DOMAIN-SPECIFIC TEXT EMBEDDING MODELS FOR INFORMATION RETRIEVAL

DOMAIN-SPECIFIC TEXT EMBEDDING MODELS FOR INFORMATION RETRIEVAL

By ALI SHIRAEE KASMAEE,

A Thesis Submitted to the School of Graduate Studies in Partial Fulfillment of the Requirements for the Degree Master of Applied Science

McMaster University
MASTER OF APPLIED SCIENCE (2025)
Hamilton, Ontario, Canada (School of Computational Science and Engineering)

TITLE:	Domain-Specific Text Embedding Models for Information Retrieval
AUTHOR:	Ali Shiraee Kasmaee
SUPERVISOR:	Dr. Hamidreza Mahyar
NUMBER OF PAGES:	xii, 116

Abstract

Large Language Models (LLMs) have shown advanced capabilities across various fields. However, using these models out of the box, especially in specialized domains like chemistry, often leads to issues such as context limitations, hallucinations, difficulty updating their parametric knowledge, and unclear sources for generated responses. To tackle these challenges, Retrieval-Augmented Generation (RAG) enables language models to use external knowledge sources during inference, improving factual accuracy and allowing dynamic knowledge retrieval without costly retraining. A critical part of any RAG system is the text embedding model, which searches for the most relevant documents in a knowledge base given a query. However, standard embedding models trained on general datasets perform poorly in chemistry because of the field's unique vocabulary, specialized terms, and complex semantics. This thesis introduces the *Chemical Text Embedding Benchmark (ChemTEB)*, designed specifically to evaluate embedding models on chemical tasks. ChemTEB systematically measures models' performance, clearly identifying their strengths and weaknesses when handling chemistry-related texts. Using insights from ChemTEB, we developed *ChEmbed*, a family of text embedding models fine-tuned specifically for chemistry. To achieve this, we gathered domain-specific text from chemistry research articles and public resources. We then generated synthetic queries for these texts using LLMs, creating realistic query-passage pairs for training. This approach effectively addresses issues of limited data availability and improves the model's ability to represent chemistry-specific language. Additionally, we introduced a new domain-specific tokenizer that efficiently integrates chemical terms into an existing pretrained model, enhancing the accuracy of text representations. Together, ChemTEB and ChEmbed offer the first domain-adapted solution for chemical text retrieval, overcoming the performance limitations of general embedding models. This contributes to more accurate and interpretable AI-based chemical research and discovery. Although the focus here is chemistry, the methods can serve as a practical framework for adapting embedding models to other specialized fields.

Acknowledgements

First and foremost, I would like to extend my sincere gratitude to my supervisor, **Dr. Hamidreza Mahyar**, for their invaluable guidance, unwavering support, and exceptional mentorship throughout my academic journey. Their thoughtful advice and continuous encouragement have shaped my academic perspective and opened doors to numerous research opportunities, enriching my understanding of the field.

I am deeply thankful to **Dr. Soheila Samiee** from BASF Canada, whose insightful feedback and valuable suggestions significantly contributed to shaping this research. Their practical perspectives provided clarity and enhanced the quality of my work.

My appreciation also goes to my colleagues and fellow lab members for their stimulating technical discussions and insightful exchanges, which continuously inspired and refined my ideas.

I am grateful to **McMaster University** for fostering an excellent academic environment that has encouraged my learning, personal growth, and research development.

Special thanks to the **Digital Research Alliance of Canada** and **BASF Canada** for providing essential computational resources; this research would not have been possible without their generous support. This research was also funded by **Mitacs** through grant IT32409.

Finally, my deepest gratitude goes to my family: my mother, father, brother, and my partner, whose unwavering emotional support and encouragement have been instrumental in my academic journey.

To all individuals and institutions mentioned, and others who supported this endeavour in any way, thank you. Without your contributions, guidance, and encouragement, this work would not have been achievable.

Table of Contents

List of Figures

List of Tables

Notation and Abbreviations

AI	Artificial Intelligence
AMP	Automatic Mixed Precision
ANN	Approximate Nearest Neighbour
API	Application Programming Interface
AVocaDo	Adapt the Vocabulary to Downstream Domain
BEIR	Benchmarking Information Retrieval
BERT	Bidirectional Encoder Representations From Transformers
BGE	BAAI General Embedding
BLUE	Biomedical Language Understanding Evaluation
BoW	Bag-of-Words
BPE	Byte Pair Encoding
CBOW	Continuous Bag-of-Words
CHEMDNER	Chemical Disease/Drug Named Entity Recognition
ChEMU	Chemical Information Extraction
ChemTEB	Chemical Text Embedding Benchmark
CLUE	Chinese Language Understanding Evaluation
CNN	Convolutional Neural Networks
CPC	Contrastive Predictive Coding

DPR	Dense Passage Retriever
DSSM	Deep Structured Semantic Model
FFN	Feed-Forward Network
GloVe	Global Vectors For Word Representation
GLU	Gated Linear Units
GLUE	General Language Understanding Evaluation
GPT	Generative Pretrained Transformer
GPU	Graphics Processing Unit
GROBID	GeneRation Of BIbliographic Data
GTE	General Text Embeddings
HBM	High-Bandwidth Memory
InfoNCE	Noise Contrastive Estimation
IR	Information Retrieval
IUPAC	International Union Of Pure And Applied Chemistry
KDE	Kernel Density Estimation
K-NRM	Kernel-based Neural Ranking Model
LLM	Large Language Model
LoCo	Long-Context Benchmark
LSTMs	Long Short-Term Memory Networks
LTR	Learning-to-Rank
MAE	Masked Autoencoders
MAP	Mean Average Precision
MLM	Masked Language Modeling
MLP	Multi-Layer Perceptron

MoE	Mixture of Experts
MRL	Matryoshka Representation Learning
MRR	Mean Reciprocal Rank
MSE	Mean Squared Error
MTEB	Massive Text Embedding Benchmark
nDCG	Normalized Discounted Cumulative Gain
NER	Named Entity Recognition
NLI	Natural Language Inference
NLP	Natural Language Processing
NQ	Natural Questions
NSP	Next Sentence Prediction
NTK	Neural Tangent Kernel
NT-Xent	Normalized Temperature-scaled Cross-Entropy Loss
PDF	Portable Document Formats
PI	Position Interpolation
PUG	Power User Gateway
RAG	Retrieval-Augmented Generation
RetroMAE	Retrogressive Masked Auto-Encoder
RLHF	Reinforcement Learning From Human Feedback
RNN	Recurrent Neural Networks
RoPE	Rotary Position Embeddings
RRF	Reciprocal Rank Fusion
S2ORC	Semantic Scholar Open Research Corpus
SBERT	Sentence-BERT

SDS	Safety Data Sheets
SELFIES	Self-Referencing Embedded Strings
SMILES	Simplified Molecular-Input Line-Entry System
SRAM	Static Random-Access Memory
SSL	Self-Supervised Learning
STS	Semantic Textual Similarity
TF-IDF	Term Frequency-Inverse Document Frequency
XGLUE	Cross-lingual General Language Understanding Evaluation
XTREME	Cross-lingual TRansfer Evaluation of Multilingual Encoders
YaRN	Yet another RoPE extensioN

Declaration of Authorship

I, Ali Shiraee Kasmaee, hereby declare that the work titled "Domain-Specific Text Embedding Models for Information Retrieval" is the result of my own efforts and contributions.

This thesis includes material from two co-authored works that have been submitted or published.

1. **Chapter 3:** A. Shiraee Kasmaee, M. Khodadad, M. Arshi Saloot, N. Sherck, S. Dokas, H. Mahyar, and S. Samiee. ChemTEB: Chemical text embedding benchmark, an overview of embedding models performance & efficiency on a specific domain. In Proceedings of the 4th NeurIPS Efficient Natural Language and Speech Processing Workshop, PMLR 262:512-531, 14 Dec 2024.

2. **Chapter 4:** A. Shiraee Kasmaee, M. Astaraki, M. Khodadad, M. A. Saloot, N. Sherck, H. Mahyar, and S. Samiee. Chembed: Enhancing chemical literature search through domain-specific text embeddings. arXiv:2508.01643, 2025.

For each of these chapters, a statement of authorship and division of labour is provided at the chapter opening. The versions here include formatting and editorial changes.

Chapter 1

Introduction

Natural Language Processing (NLP) is an interdisciplinary field at the intersection of artificial intelligence, computer science, and computational linguistics, enabling computers to process, comprehend, and generate human language. Its core aim is to bridge the communication gap between humans and machines by transforming unstructured linguistic data into structured information for diverse applications. Early NLP relied on rule-based methods, where language behaviour was encoded via handcrafted grammars and heuristics. Notable examples include ELIZA [192], which simulated dialogue through pattern matching, and SHRDLU, which introduced a "blocks world" interface where users manipulated objects via natural language commands. These systems, however, relied on manually defined rules, making them labour-intensive and poorly suited to handling real-world variability. In the 1980s and 1990s, statistical approaches [110] emerged, driven by the growing availability of text corpora. Models such as the IBM translation systems [20] learn linguistic patterns from data rather than relying on explicit rules. These methods powered tasks such as machine translation, part-of-speech tagging, and sentiment analysis; yet, they still struggled with long-range dependencies, deep context, and semantic nuance, treating words largely as isolated units [59]. Such limitations underscored the need for better representations, setting the stage for neural approaches. The deep learning revival of the early 2010s reshaped NLP. Inspired by AlexNet's [91] success on ImageNet [41] and advances in GPU computing, researchers applied neural networks to language [34, 12]. Recurrent Neural Networks (RNNs) [45], particularly Long Short-Term Memory networks (LSTMs) [67], have proven effective in modelling text as a sequence and capturing longer dependencies than statistical models. At the same time, distributed word representations (word embeddings) such as Word2Vec [114] and GloVe [134], enabled dense, semantically rich encodings that surpassed bag-of-words [153] and n-gram schemes [160]. While recurrent architectures such as RNNs and LSTMs improved representational power, they compute one token at a time.

Each step depends on the previous one, which prevents parallelization across time and slows training and inference, and they still struggle with long-range dependencies [11]. These challenges ultimately motivated the search for architectures that could handle longer context more efficiently.

Two key drawbacks of recurrent models, difficulty in modelling long-range dependencies and the lack of parallel computation, motivated researchers to explore alternatives. This effort led to the introduction of the Transformer architecture, as described in Attention Is All You Need [180]. At its core, the self-attention mechanism enables the model to focus on any part of a sequence, regardless of position, while processing all tokens in parallel with simple feed-forward layers, rather than recurrent units. The original Transformer, designed for machine translation, used an encoder-decoder layout; its strong results quickly encouraged the community to adopt and refine the idea. Encoder-only variants, most notably BERT [43] and RoBERTa [106], excelled at classification, sentiment analysis, and named entity recognition, while decoder-only models such as the GPT family [139, 140, 21] and the Llama family [177, 178, 55] demonstrated powerful generative ability. Continuous improvements on these foundations have produced the large language models that dominate modern NLP research and applications today.

Large Language Models (LLMs) then quickly became influential artificial intelligence tools. They are built as unidirectional, autoregressive Transformers: at each step, the model considers only the preceding tokens and selects the next one according to a learned probability distribution. At each step, the model estimates a probability distribution over all possible next tokens, conditioned on the preceding context, and selects the most likely token. [139]. They are first pretrained on massive text corpora, growing from GPT-1's 985 million words to GPT-2's tens of billions of tokens and GPT-3's hundreds of billions, with modern models such as LLaMA 3 trained on up to 15 trillion tokens [55]. A pivotal discovery was that these autoregressive models could perform a wide range of tasks without explicit fine-tuning: GPT-2 exhibited strong zero-shot abilities across diverse NLP benchmarks [140], and GPT-3 further showcased robust few-shot learning, mastering new tasks from just a few in-context examples and often matching fully fine-tuned systems without requiring gradient updates [21]. As a result, LLMs now support sophisticated classification, summarization, coding assistance [28], and complex agentic workflows involving planning and tool use [205, 156]. This versatility enables domain-specific applications, accelerates scientific discovery [168, 108, 169], supports decision-making in finance, and encourages innovation in various industries.

Retrieval-Augmented Generation Large Language Models (LLMs), though powerful, have built-in limitations due to their design, mainly their dependence on "parametric memory", where knowledge acquired during training is stored within the model's parameters. [146, 137]. This means their knowledge is limited to the data they were trained on, making them unable to access or reason about new information or events that happened after training without external sources. As a result, LLMs often "hallucinate"; they produce answers that sound convincing but are wrong or meaningless, particularly when questions go beyond what they "know". Instead of relying on up-to-date and reliable sources, they use their built-in parametric memory to address queries. This presents a significant challenge in cases where it is crucial to feed them new or completely fresh information [70]. While fine-tuning is an option, it is often a complex and resource-intensive task for general users; for instance, training a 175 billion version of the GPT-3 model with a single V100 NVIDIA GPU takes approximately 288 years [120]. One effective approach to address these limitations is **Retrieval-Augmented Generation** (RAG), which allows new knowledge to be provided to the LLM as part of its input context rather than through retraining [99]. RAG enhances LLMs by enabling them to access external knowledge sources, making them more robust, reliable, and less prone to hallucination. A RAG pipeline typically involves using a query to search an existing external knowledge base; the most relevant pieces of information are then retrieved and augmented with the original input query, creating a richer, fact-grounded context for the LLM to generate its response.

Text Embedding Models One of the most essential components in a Retrieval-Augmented Generation (RAG) pipeline is a text embedding model, which determines how accurately the system retrieves relevant documents in response to a query. These models convert text into dense, high-dimensional numerical representations, allowing semantic similarity between texts to be efficiently measured using metrics like cosine similarity [145, 185]. Unlike encoder-based models, which are trained with masked language modelling objectives [43], text embedding models are typically built upon pretrained encoders and further trained using **Contrastive Learning** [126]. This training paradigm encourages the model to generate similar embeddings for semantically related texts and distinct embeddings for unrelated ones, effectively capturing the semantic essence of the sequences. Beyond their role in RAG systems, text embedding models are versatile tools applicable to various search-based tasks, including web and patent retrieval, recommendation systems, and tasks like classification and clustering that benefit from contextual vector representations.

Domain Gap Chemical sciences feature a unique and complex language, characterized by specialized nomenclature systems such as IUPAC, SMILES, and SELFIES, as well as detailed descriptions of reactions and experimental conditions.

These domain-specific terminologies and representations are rarely found in the general or web-scale corpora typically used to train general language models. Although attempts have been made to adapt encoder-based transformer models to chemical data [10, 57, 31], these models are typically not trained with contrastive objectives, and as a result, often exhibit suboptimal performance in tasks that require semantic understanding, such as information retrieval within chemical contexts. Moreover, there is a significant gap in open-source embedding models specifically designed for chemical text, as well as an absence of publicly available datasets suitable for training or fine-tuning such models. Although benchmarks like the Massive Text Embedding Benchmark (MTEB) [118] are available to evaluate text embedding models across general tasks and specific domains, such as medicine and law, no specialized benchmark exists for assessing the performance of text embedding models on chemical literature. To fill these gaps, this thesis presents **ChemTEB**, a benchmark designed to evaluate text embedding models on chemical tasks [164], and introduces the **ChEmbed** family of chemical-specific text embedding models [165]. These contributions aim to provide insights into domain adaptation and dataset construction, with methodologies that can be generalized to other specialized disciplines.

Contributions This thesis advances the field of chemistry information retrieval through the development and evaluation of domain-specific embedding models and benchmarks. The key contributions of this work are as follows:

1. **Evaluation Benchmarks for Chemistry:** Developed a comprehensive suite of evaluation tasks for chemistry information retrieval, including: (a) *ChemTEB*, a **Chem**ical **T**ext **E**mbedding **B**enchmark designed to assess the performance of open-source and proprietary general text embedding models on a broad set of chemistry-specific NLP tasks, and (b) *ChemRxiv Retrieval*, a literature-centric evaluation task explicitly tailored for information retrieval in chemical texts, closely aligned with real-world use cases in scientific research.

2. **Scalable Synthetic Data Generation Framework:** Designed a robust and scalable framework for generating high-quality synthetic query-passage pairs from raw chemical paragraphs, leveraging large language models to overcome the scarcity of annotated data necessary for training domain-specific text embedding models.

3. **ChEmbed Model:** Introduced *ChEmbed*, the first text embedding model specifically trained and optimized for retrieval-augmented generation (RAG) pipelines and chemical literature retrieval, addressing the unique challenges posed by chemical scientific language.

4. **Tokenizer Adaptation for Chemistry:** Proposed and validated a practical tokenizer adaptation approach, augmenting the model's vocabulary with chemistry-specific tokens, leading to enhanced capacity to capture domain-specific nuances such as IUPAC nomenclature and chemical identifiers.

Thesis Organization This thesis is structured into six chapters as follows:

1. **Chapter 1: Introduction**
   Introduces the motivation, objectives, and scope of the thesis, outlining the challenges of domain-specific information retrieval in chemistry and presenting the main contributions of the work.

2. **Chapter 2: Background and Related Work**
   This chapter begins with the general information-retrieval problem and illustrates how early lexical methods attempted to address it. It then follows a gradual improvement: first with weighting schemes, then with learning-to-rank ideas, and finally with neural retrievers, which are now replaced by dense text-embedding models. After that, the chapter introduces contrastive learning as the primary method for training these models and briefly presents the transformer refinements that enhance their speed and accuracy. The chapter concludes by summarizing the primary benchmarks used to evaluate embedding quality and reporting the most relevant language models that have already been adapted for chemical text.

3. **Chapter 3: ChemTEB; evaluation text embedding models in chemistry**
   Presents the ChemTEB benchmark, a comprehensive suite of chemistry-specific evaluation tasks. This chapter systematically evaluates the performance of general-purpose text embedding models across these tasks, providing an in-depth analysis of their strengths and limitations within the chemical domain.

4. **Chapter 4: ChEmbed; domain-specific text embeddings model for chemistry information retrieval**
   Details the development of ChEmbed, including data preparation (gathering raw chemical text from multiple sources and preprocessing it), synthetic query generation, model training, and tokenizer adaptation. This chapter also presents experimental results benchmarking ChEmbed against baselines and analyzes its performance in realistic chemical retrieval scenarios.

5. **Chapter 5: Discussion and Conclusions**
   Summarizes the key findings of this work, discusses challenges, limitations,

and lessons learned, and outlines promising directions for future research in domain-specific text embedding models.

Chapter 2

Background and Related Work

2.1 Traditional Information Retrieval

Information Retrieval (IR) involves finding relevant information in large datasets. Early IR methods were based on lexical matching (using keywords and term frequencies), while later approaches moved to semantic matching with learned representations. This section provides a theoretical background, covering both classical IR methods to the first generation of neural retrieval ones.

2.1.1 Definitions

In information retrieval, a **document** is a piece of text (such as an article) found in a collection of documents D. A **query** q is the search input of one or more words that a user provides, which describes what they are looking to find. The IR system aims to retrieve a ranked list of documents $d \in D$ that are **relevant** to the query, documents that contain the information the user is looking for. In formal terms, relevance refers to evaluating how well a document's content addresses the query, typically using a binary (e.g., relevant / not relevant) or graded/ordinal (e.g., 0-3, 0-5) scale, which are the most common forms in IR [111]. Other formulations also appear, such as pairwise preferences [79], listwise judgments [24], and implicit behavioural signals (e.g., clicks, dwell time) used as noisy proxies for relevance. The process of **retrieval** involves calculating a **score** $s(d, q)$ for each document with respect to the query, and ordering the documents based on these scores so that the most relevant ones appear first [111].

2.1.2 Bag-of-Words Model

The bag-of-words (BoW) model is a simple but foundational model for representing documents in Information Retrieval. In this model, a document is seen as an unordered

set of words, where only the frequencies of words matter and their ordering is completely ignored. A vocabulary (dictionary) of all unique terms in a corpus is defined, and each document is represented as a vector of term frequencies. Each component of this vector corresponds to a term in the vocabulary, where its value is the number of times that term has appeared in the document. In general, if two sentences have similar BoW representations, one would expect them to be similar in content. However, that is not always the case with BoW. For example, the sentences "Mark is smarter than Alex" and "Alex is smarter than Mark" both have the same BoW representations, because they have the same terms with the same frequencies, even though the word order is different. The BoW is simple and convenient for mathematical modelling, but it ignores context and grammar [111].

2.1.3 Term Weighting and TF-IDF Scheme

One drawback of the basic BoW model with raw word counts is that it treats every word the same. In practice, not all words carry the same importance in conveying what a document is about. Common words like "the" or "is", and domain-specific words that appear in almost every document (e.g., "compound" in a collection of chemistry articles) have little discriminative power to distinguish one document from another. However, a word that appears frequently in one document but rarely in others is often a good indicator of that document's specific topic. To handle this issue, IR systems use **term weighting** methods to reduce the importance of common words and increase the influence of less common, more informative words.

The most common weighting approach is based on term frequency and inverse document frequency, known as the **TF-IDF** scheme. First, we define the document frequency $\mathrm{df}(t)$ of a term as the number of documents in the collection that contain that term. The idea is that words that appear in many documents (high df) are common words, while words with low *df* are more unique. Next, we define the **inverse document frequency** (idf) of term t as:

$$\mathrm{idf}(t) = \log\left(\frac{N}{\mathrm{df}(t)}\right)$$

where N is the total number of documents in the corpus. The logarithmic idf factor is high for rare words (when $\mathrm{df}(t)$ is low) and low for frequent words (when $\mathrm{df}(t)$ is high). The idf reduces weight for words that appear in many documents and increases weight for words that appear in only a few. Finally, the **tf-idf weight** for a term t in document d is calculated by the product of its term frequency and inverse document frequency:

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, D) \cdot \log(\frac{N}{\mathrm{df}(t)})$$

where $\text{tf}(t, d)$ is the term frequency of t in document d, $\text{df}(t)$ is the document frequency of t in the corpus D, and $N = |D|$ is the total number of documents. In other words, $\text{tf}(t, d)$ shows how important term t is in this particular document (by counting occurrences), while the $\log(\frac{N}{\text{df}(t)})$ factor shows how informative t is in the entire collection (reducing the weight of common words). The tf-idf approach thus assigns a higher weight to:

- A term that appears often in a given document.
- A term that appears in just a few documents overall.

And it gives near-zero weight to terms that appear in almost every document, since such terms lack discriminative value. To summarize:

- **Term frequency:** $\text{tf}(t, d)$ increases with the number of times term t appears in document d. A higher tf indicates that t is more related to the document's topic.
- **Document frequency:** $\text{df}(t)$ is the number of documents that have term t. A high df means t is common in the corpus (lower importance), while a low df means t is rare (higher importance).
- **Inverse document frequency:** $\log(\frac{N}{\text{df}(t)})$ is large for rare terms and small for common terms, reducing the weight for very common words.
- **Tf-idf weight:** $\text{tf}(t, D) \cdot \log(\frac{N}{\text{df}(t)})$ balances these factors, emphasizing words that are frequent in d but rare in the whole corpus.

2.1.4 Sparse Lexical Methods

Building on the bag-of-words representation and tf-idf weighting introduced earlier, classical IR systems compare queries and documents in a sparse lexical space. In this section, we will cover three flagship models that differ mainly in how they compute a relevance score $s(d, q)$ after converting text into tf-idf or similar weight vectors.

Vector Space Model is a foundational model introduced by Salton in 1975 [153], in which each document d and query q are represented as high-dimensional vectors of term weights (often using tf-idf). It is the first formal IR approach to represent text as high-dimensional vectors, allowing algebraic similarity calculations. The relevance score is calculated as the cosine similarity between query and document vectors.

$$s(\mathbf{d}, \mathbf{q}) = \frac{\mathbf{d} \cdot \mathbf{q}}{\|\mathbf{d}\| \, \|\mathbf{q}\|}$$

where $\mathbf{d} \cdot \mathbf{q} = \sum_t \text{tf-idf}(t, d)\,\text{tf-idf}(t, q)$ is the dot product of the two weight vectors, and $\|d\|$ and $\|q\|$ are their Euclidean lengths. Cosine similarity measures the angle between the query vector and the document vector. A value of 1 means the vectors point in the same direction (the document's term distribution exactly matches the query's terms), and zero means they share no common terms. Ranking documents by decreasing cosine similarity is a common core mechanism in IR.

BM25 One of the limitations of the previous method, primarily when used with standard tf-idf weighting, is that the term frequency often increases linearly with the number of terms appearing in a document, and can be skewed by documents that have a high frequency of a particular term, even if the overall relevance is not proportionally higher. Another influential lexical method that addresses this challenge is Okapi BM25 [147]. BM25 scores a document by summing term contributions for each query term $t \in q$. Each term's contribution grows with its term frequency $tf(t, d)$ in the document adjusted by a **saturation parameter** k_1 which controls how much the contribution of a term saturates as its frequency increases. The intuition behind this parameter is that the first few occurrences of a query term in a document are very important. Additionally, the score is adjusted by document length normalization, with the intuition that longer documents are naturally more likely to contain query terms than shorter ones, even if they are not more relevant. A simplified BM25 scoring function is:

$$s(d, q) = \sum_{t \in q} \text{idf}(t) \frac{\text{tf}(t, d)(k_1 + 1)}{\text{tf}(t, d) + k_1 \left(1 - b + b\frac{|D|}{L_{\text{avg}}}\right)}$$

where $|D|$ is the length of document d, L_{avg} is the average document length in the corpus, and k_1 and b are parameters to calibrate term saturation and document length. In simple terms, BM25 rewards repeated appearances of a query term in d but with a saturation effect, and it penalizes very long documents using b to avoid length bias.

Limitations Despite their effectiveness and simplicity, these lexical methods expect exact word matches, so they struggle with lexical mismatch (e.g., a document about 'sodium chloride' might not rank for the query 'table salt' if the term 'table salt' isn't present). They also cannot understand the notion of semantic similarity or context; each term is treated independently, ignoring word meaning and order. The vector representations are high-dimensional and sparse (one dimension per vocabulary term, with most entries being zero). This sparsity is computationally convenient for inverted indexing, but it makes similarity rely entirely on exact term overlap: related texts that use different words often receive near-zero similarity, which limits generalization. In multilingual settings, separate vocabularies and indexes further increase storage and maintenance costs.

2.1.5 Distributed Embeddings

To bridge the semantic gap, IR approaches started using **distributed representations** of text, following progress in neural language models. These representations map words or documents to low-dimensional, dense vectors (also called *embeddings*) such that vectors with similar meaning appear close in the vector space. This is in contrast to sparse representations, such as one-hot encoding, where each word is represented as an isolated dimension.

word2vec is a major advance in this category [114], which, by observing which words tend to appear in the same contexts, learns embeddings that place semantically similar terms close together in the vector space, which uses two complementary methods to train embeddings:

- **Skipgram:** The Skip-gram model takes a single target word as input and learns to predict each of its surrounding context words w_{t+j} $(-c \le j \le c,\ j \ne 0)$ independently. Its objective is to maximize the log-likelihood of the actual context words given the target:

$$\max \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log P\big(w_{t+j} \mid w_t\big)$$

  A shallow neural network with a one-hot input layer, a single hidden projection layer (of dimension d), and a softmax output layer [18] is trained via backpropagation [151]. After training, the row of the projection weight matrix corresponding to each word serves as its d-dimensional embedding.

- **Continuous Bag-of-Words (CBOW):** it takes the $2c$ context words $\{w_{t+j}\}_{-c \le j \le c,\ j \ne 0}$ as input (their embeddings are typically averaged or summed) and predicts the centre word w_t where the training goal is:

$$\max \sum_{t=1}^{T} \log P\big(w_t \mid w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}\big).$$

  Like Skipgram, CBOW employs a single hidden layer whose weights become the learned embeddings, but benefits from averaging multiple context signals, often resulting in smoother vector estimates for rare words.

Both architectures rely on optimization tricks (e.g., negative sampling [115] or hierarchical softmax) to avoid the computational cost of a full-vocabulary softmax at each update. By adjusting the embedding weights to improve prediction accuracy, these models capture rich semantic patterns.

GloVe is a *count-based* approach (in contrast to the *predictive* nature of Word2Vec) which builds on global co-occurrence information across the entire corpus [134]. The core idea is that **ratios** of word-word co-occurrence probabilities between words can capture semantic meaning. For example, considering the words "ice" and "steam", the ratio of their co-occurrence probabilities with a word like "solid" will be large. Conversely, for a word like "gas", this ratio will be small.'

$$\frac{P(\text{solid} \mid \text{ice})}{P(\text{solid} \mid \text{steam})} \gg \frac{P(\text{gas} \mid \text{ice})}{P(\text{gas} \mid \text{steam})}$$

These kinds of ratios are large when one word strongly signals another, small when it does not. For training the model, GloVe first constructs a full word-word co-occurrence matrix X, where each entry X_{ij} records the frequency with which word j appears in the context of word i across the entire corpus. The model aims to learn two sets of vectors: word vectors w_i and context vectors $\tilde{w}_j$, along with biases b_i and $\tilde{b}_j$, such that their dot product approximates the logarithm of their co-occurrence matrix:

$$w_i^\top \tilde{w}_j + b_i + \tilde{b}_j \approx \log\big(X_{ij}\big)$$

The training objective is a weighted least-squares problem, where the weighting function assigns less weight to highly frequent co-occurrences (often less informative, such as "the" and "is") and also ensures that zero co-occurrences do not dominate the cost.

$$J = \sum_{i,j=1}^{V} f\big(X_{ij}\big)\left(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log(X_{ij})\right)^2$$

By optimizing this objective, GloVe learns word embeddings that encode relationships between words as linear patterns in the vector space.

2.1.6 Learning-to-Rank (LTR)

As the number of hand-crafted IR features and heuristics grew, a paradigm shift occurred: instead of manually designing ranking functions such as BM25, allow systems to learn ranking from data. Learning-to-Rank (LTR) methods use machine learning on labeled examples of queries and documents (with relevance judgments) to learn a scoring function $s_\theta(d, q)$ with parameters θ [105]. This data-driven approach can leverage many features and optimize directly for ranking performance. These methods are typically classified into three groups based on their objective formulation:

- **Pointwise LTR:** Consider ranking as a regression or classification task on single query-document pairs. For example, given a query and a document, it predicts a relevance score or label (such as "relevant" vs. "not relevant").

The model is trained to output a value close to the human-judged relevance. This approach does not consider document-to-document interactions, effectively learning a scoring function like $s_\theta(d, q) \approx \text{rel}(d, q)$.

- **Pairwise LTR:** Transform ranking into a binary classification task on document pairs. In this setup, the model learns to prefer a relevant document over a non-relevant one for the same query. Training samples are document pairs (d^+, d^-), with d^+ being more relevant than d^-. The loss function encourages $s_\theta(d^+, q)$ to be higher than $s_\theta(d^-, q)$ by a margin. For instance, a common pairwise loss is a hinge loss on the score difference:

$$\min_\theta \sum_{(d^+,d^-)} \max\{0,\ 1 - s_\theta(d^+, q) + s_\theta(d^-, q)\}.$$

  This loss is zero only when the score of every preferred document exceeds that of any less-relevant document by at least 1, thereby optimizing the document ranking directly [23, 22, 78].

- **Listwise LTR:** Takes the entire ranked list into account when learning. These methods define a loss function on a permutation of documents, aiming to optimize metrics such as Mean Average Precision or nDCG directly. Such methods can use a probabilistic model over permutations or differentiate through a ranking metric. While more complex, listwise methods can capture interactions among all results for a query and directly target the true evaluation measure.

These approaches significantly improved ranking performance by using a wide range of features (e.g., BM25 score, query term frequencies) and optimizing for user-centric metrics. However, they relied on hand-engineered features. Designing and computing features for each query-document pair was slow and error-prone, and coverage was often incomplete, so subtle relevance signals were missed. These limitations paved the way for neural networks to learn representations and features automatically and directly from raw text.

2.1.7 First Neural Rankers

The next step was to leverage neural networks for ranking, going beyond static and hand-crafted features. Neural information retrieval models can be categorized into two categories [117]: **representation-based** models and **interaction-based** models. Representation models encode queries and documents independently into semantic vectors, whereas interaction models aim to directly learn detailed term-by-term interactions between queries and documents.

Representation-based neural rankers. The main idea is that a neural network learns to map text into a vector space where the distance or angle between these vectors shows how well a document matches a query. While some later models in this category may use pretrained word embeddings, such as Word2Vec or GloVe, as a starting point for representing words, early influential models often learned their own representations directly from raw text input. The key is that the query and the document are processed independently by the neural network before their final representations are compared. An early example of this approach is the **Deep Structured Semantic Model (DSSM)** [71]. DSSM uses a multi-layer perceptron (MLP) [149, 151] to project both the query and the document into a shared low-dimensional vector space. Notably, instead of directly using raw words, it employs letter-trigram hashing to create features that handle large vocabularies and words not encountered during training. These features are then fed through multiple layers of the network. The relevance between a query and a document is then calculated simply as the cosine similarity between their output vector representations. DSSM was trained on large amounts of click-through data from search engines, learning to make the vectors of queries and clicked documents more similar. After DSSM, other notable representation-focused models emerged, such as **C-DSSM** [163], which used Convolutional Neural Networks (CNNs) [51, 91, 96] to better capture word sequences, and **ARC-I** [69], which also used CNNs for generating sentence representations.

Interaction-based neural rankers. These models let the query and the document interact with each other earlier in the process. Instead of first condensing each text into a single vector, they typically start by constructing an interaction matrix that captures the similarities between individual terms or phrases in the query and those in the document. Neural networks, often CNNs, are then employed to analyze this matrix and learn meaningful matching patterns directly from these detailed term-by-term interactions. This approach enables the model to capture subtle details that may be overlooked when the texts are encoded independently. One of the early influential models in this category is the **Kernel-based Neural Ranking Model (K-NRM)** [201], which first calculates pairwise similarities (e.g., cosine similarity) between the embeddings of each query term and each document term. It then applies a bank of Gaussian (RBF) kernels to these scores to build soft-match histograms that count term pairs at various similarity levels (exact, strong, weak). The histogram features are then fed into a network to output a final relevance score. Another significant model, **Duet** [116], combined both interaction and representation approaches: it has a local sub-network that uses a CNN on a query-document interaction matrix to find precise, local matching patterns, and a parallel distributed sub-network that learned to match based on the global semantic representations of the query and document, similar to representation-based models. Other earlier examples include ARC-II

[69] and MatchPyramid [129]. These interaction-based methods excel at capturing fine-grained relevance signals by directly focusing on how query terms relate to document terms. These fundamental architectural choices, encoding inputs separately or together, are conceptually analogous to the modern paradigms of **bi-encoders** and **cross-encoders**, respectively, a distinction illustrated in Figure 2.1). Each has its own advantages and drawbacks, helping advance neural IR methods. While this first

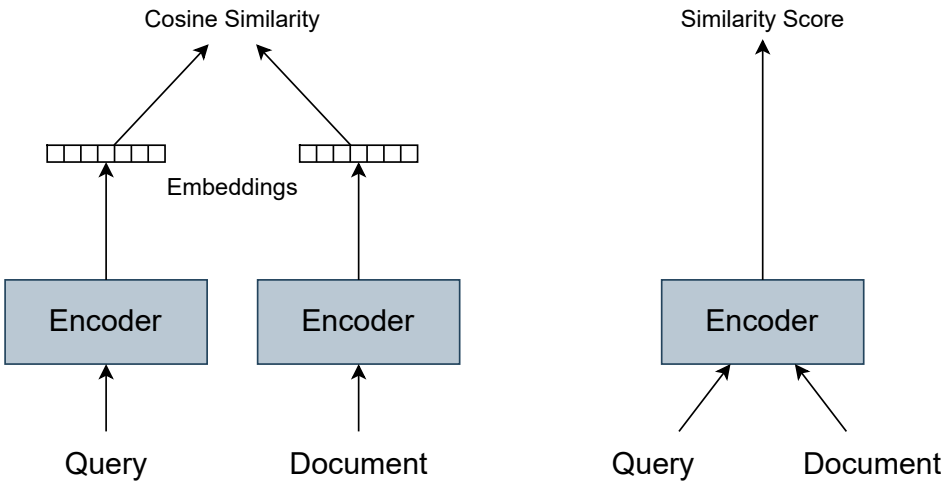

Figure 2.1: Bi-encoders (left) vs cross-encoders (right)

wave of neural rankers showed significant promise for information retrieval, they also revealed several challenges. One major issue is context handling. Many early models treated text as a *bag of features*, even though those features were learned. Models that did use word embeddings, like K-NRM or the distributed part of Duet, often relied on static embeddings, meaning that each word had a single vector representation, regardless of its surrounding words. Another essential drawback in data efficiency and generalization is that neural networks typically require large training datasets to learn effectively [97, 41]. However, large datasets to train these models, often consisting of click logs or human relevance judgments, are not always publicly available, making it difficult for the research community to consistently build upon these models. Finally, another practical concern is scalability. These limitations set the stage for the next generation of IR models. In particular, the inability of early neural methods to fully capture context and language nuances motivated the adoption of transformer-based encoders (such as BERT-based models [43, 106]) for retrieval. Transformer models

introduced context-aware embeddings and knowledge from pretraining on vast text corpora, addressing many of the limitations of early neural rankers.

2.2 Transformers

In the early years of deep learning, recurrent neural networks were the primary choice for solving problems involving sequential data, such as text, speech, and time series. However, several limitations of these networks led to the development of more advanced concepts, such as the attention mechanism and transformers. One major drawback of RNNs was the difficulty in capturing long-range dependencies, primarily due to the **vanishing gradient** problem [11, 66], where gradients diminish exponentially during backpropagation. Additionally, in RNNs, computations in each step depend on the previous steps, so their inherent sequential nature prevents effective parallelization, resulting in increased training time and inefficient utilization of modern processing hardware, such as GPUs.

2.2.1 Attention Mechanism

Over the past decade, sequence-to-sequence models powered by recurrent networks (e.g., LSTMs [67]) transformed tasks like machine translation by learning to encode an input sequence into a fixed-length context vector and then decode it into another language [32, 173]. However, squeezing an entire sentence into a single vector creates a severe information bottleneck: long inputs are compressed in a lossy manner, and gradients struggle to flow back through multiple timesteps. The attention mechanism [8] was introduced to overcome this limitation by allowing the decoder to look back at all encoder states, assigning each a learned weight, or attention score, based on its relevance to the current decoding step. This simple but powerful idea not only expanded a model's effective memory without swelling its parameter count but also provided a direct shortcut for gradient flow, significantly improving performance on long or complex sequences. Figure 2.2 illustrates the difference between the **vanilla** and **attention-based** encoder-decoder architectures. In the **vanilla model** (upper diagram), the encoder processes the input sequence $X = (x_1, \dots, x_T)$ and passes only its last hidden state $s = h_T$ to the decoder. Every output y_t, therefore, depends on this single, fixed-length summary. The **bottom diagram** illustrates additive attention, which removes this bottleneck by providing a step-specific summary. After the encoder has generated all hidden-states $H = \{h_1, \dots, h_T\}$ calculates a scalar relevance score, just before generating token y_t, which measures how well the current decoder state s_{t-1} matches each encoder state h_i:

$$e_{t,i} = v_a^\top \tanh\big(W_s s_{t-1} + W_h h_i\big), \tag{2.2.1}$$

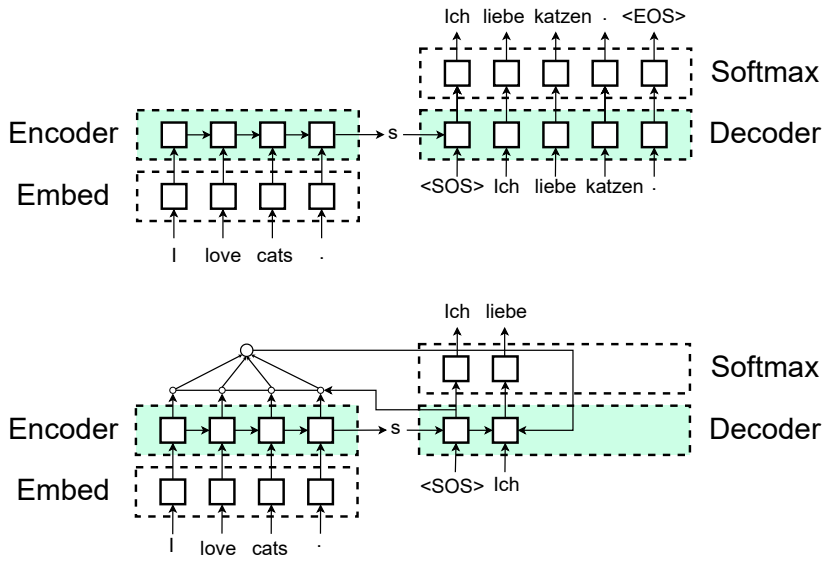

Figure 2.2: Top: vanilla encoder-decoder; Bottom: encoder-decoder with attention. Adapted from Smerity.com.

A softmax then turns these scores into *attention weights*

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{T} \exp(e_{t,j})}, \qquad \sum_{i=1}^{T} \alpha_{t,i} = 1, \tag{2.2.2}$$

which results in the *dynamic context vector*:

$$c_t = \sum_{i=1}^{T} \alpha_{t,i}\, h_i, \tag{2.2.3}$$

a weighted average emphasizing the most relevant source positions. This context augments the next decoder hidden state:

$$s_t = \mathrm{RNN}(y_{t-1},\, s_{t-1},\, c_t), \tag{2.2.4}$$

and finally feeds the output to the softmax

$$p(y_t \mid y_{<t}, X) = \mathrm{softmax}(W_o s_t). \tag{2.2.5}$$

Because c_t is recalculated at each step, the decoder can focus on different parts of the input sentence as it is translating, so it does not have to compress all the information

into a single vector, and the model stays small in terms of parameter count.

2.2.2 Transformer Architecture

Adding attention significantly improved recurrent networks, because the decoder could access all encoder states rather than depending on a single fixed vector. Translation quality improved, training with long sentences became easier, and the vanishing gradient was decreased. However, the model still processed tokens sequentially, so training remained slow on large corpora. Building directly on the success of attention, Vaswani *et al.* [180] proposed an architecture without any recurrent units. Each layer has only linear projections and *scaled dot-product attention*, allowing all tokens in a sequence to be processed in parallel. This model, called the **Transformer**, keeps the benefits of attention and eliminates the sequential bottleneck of RNNs.

Architecture. Figure 2.3 illustrates a classic encoder-decoder architecture, which is built solely with attention and feed-forward layers, without any recurrent or convolutional units. Both encoder and decoder are stacks of N identical layers ($N = 6$ in the original variant). Every sub-layer uses a residual connection [61] followed by layer normalization [6]. Given an input x, the sub-layer output is:

$$\mathrm{LayerNorm}\big(x + \mathrm{Sublayer}(x)\big), \tag{2.2.6}$$

where $\mathrm{Sublayer}(x)$ denotes that layer's transformation. To simplify residual connections, every sub-layer and embedding layer outputs vectors of the same size, $d_{\text{model}} = 512$.

Encoder stack. Each encoder layer uses a multi-head **self-attention** module, letting each word look at all other words in the sequence at the same time and in parallel. The attention output then passes through a **position-wise feed-forward** network. After N layers, the encoder has built a contextual representation H that the decoder can use.

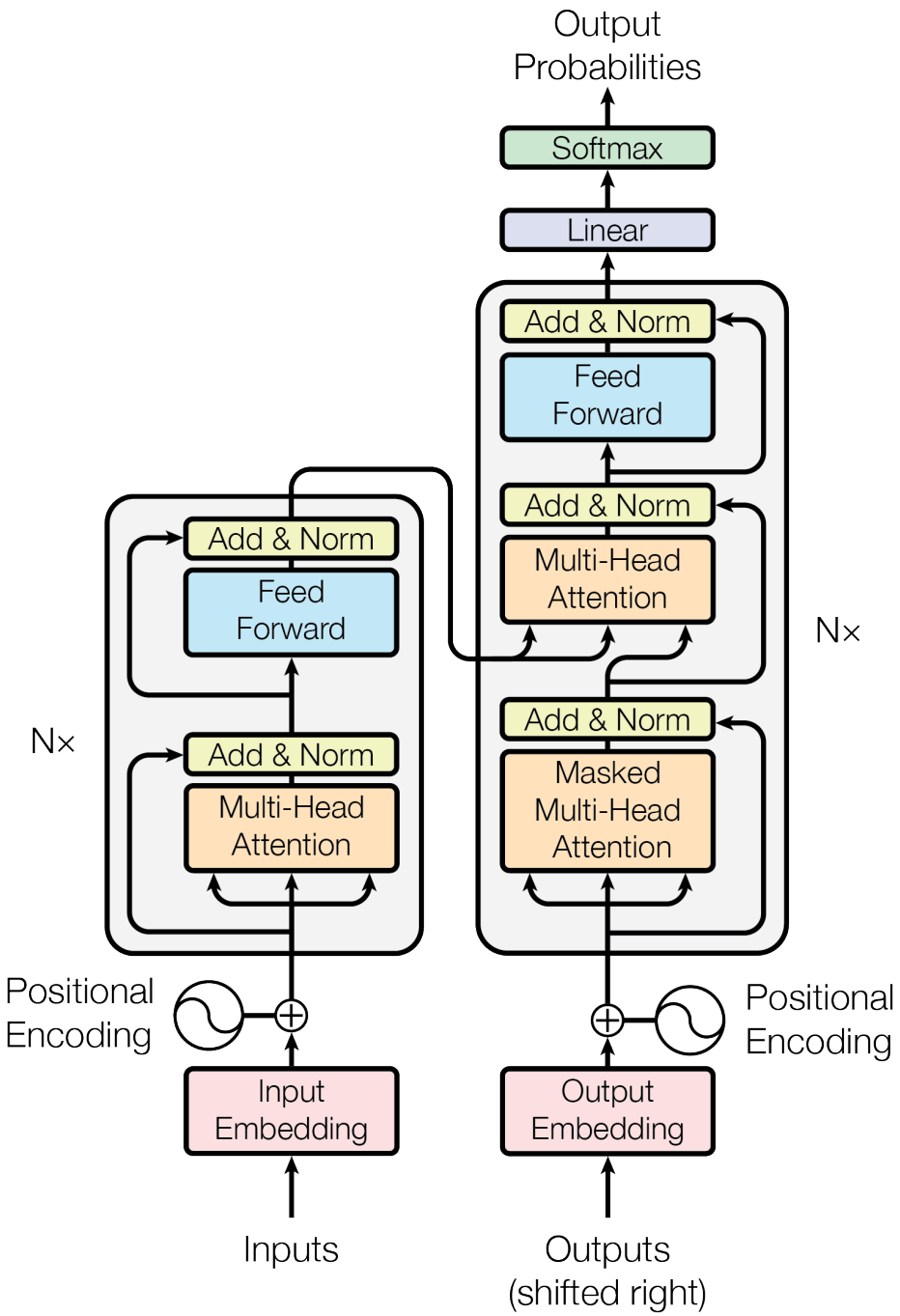

Figure 2.3: The Transformer architecture. Reprinted from Vaswani et al. [180]

Decoder stack. Each decoder layer begins with a **masked** self-attention block that can look only at earlier target tokens. This preserves the autoregressive property: when generating y_t, the model cannot peek at y_{t+1} or beyond. A second attention block (often called **encoder-decoder attention** or **cross-attention**) takes its queries from the decoder and its keys and values from the encoder's H. This lets every target position attend to the full source sequence. Finally, a feed-forward network processes the attended information before moving to the next layer. At inference time, the decoder runs step by step, but within each step, all its matrix operations are fully parallel.

Scaled dot-product attention. Figure 2.4 (left) illustrates this operation. Inside every head the model first builds three matrices; queries $Q \in \mathbb{R}^{T_q \times d_k}$, keys $K \in \mathbb{R}^{T_k \times d_k}$, and values $V \in \mathbb{R}^{T_k \times d_v}$ by multiplying the input matrix X with learned weights W^Q, W^K and W^V. We can think of it as an information retrieval scenario; keys act like database indices, queries denote search strings, and values are like the records we want to retrieve. The attention, which is a notion of relevance, is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.2.7}$$

Where $\sqrt{d_k}$ is the key/query dimensionality. If the components of q and k are zero-mean and unit-variance (as is often assumed, e.g., for normalized random vectors), then $\text{Var}(q \cdot k) = d_k$. Scaling by $1/\sqrt{d_k}$ keeps these dot-product values around order 1, which prevents the softmax from producing extremely peaked or flat distributions and helps maintain stable gradients. The resulting weights select a weighted sum of the values, giving a context vector for each query position.

Multi-head attention. Instead of running a single attention with d_{model}-dimensional vectors, we project the queries, keys, and values h times with learned linear maps, producing smaller dimensions d_k and d_v. With $h = 8$ in the original model, we have $d_k = d_v = d_{\text{model}}/h = 64$, so the total cost stays the same. The h attention outputs are concatenated and passed through a final linear layer (Figure 2.4, right).

Multi-head attention is used in three places:

1. **Encoder-decoder attention**: the decoder supplies the queries, while the keys and values originate from the encoder.
2. **Encoder self-attention**: all queries, keys, and values are provided by the previous encoder layer.
3. **Decoder self-attention**: same as above, but masked to preserve autoregression.

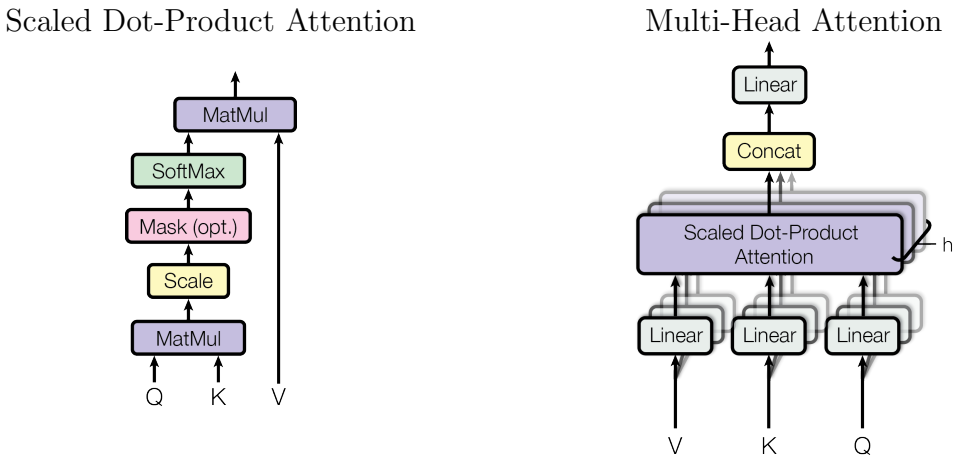

Figure 2.4: Scaled Dot-Product Attention computes attention weights by comparing queries and keys, producing a weighted sum of the values. (left) Multi-Head Attention applies several independent attention mechanisms in parallel, enabling the model to capture information from multiple representation subspaces at once (right). Reprinted from Vaswani et al. [180]

Feed-forward layers and output. After every attention sub-layer, the information at each sequence position runs through the same two-layer feed-forward network (FFN). This block is applied independently to every position, so its weights are shared across time but differ from layer to layer. The first linear map expands the hidden size from d_{model} to a larger inner dimension d_{ff} (2048 in the original paper), a non-linear activation (ReLU or, in newer models, GELU [64]) adds capacity, and the second linear map projects back down to d_{model}. Dropout is typically added after the activation layer to prevent overfitting. A residual connection and layer normalization wrap the whole FFN, precisely as in the attention sub-layers, so gradients can bypass the non-linearity when helpful. Once the decoder stack has produced its final hidden state matrix $Z \in \mathbb{R}^{T \times d_{model}}$, each row z_t passes through a learned output matrix W^O whose width is equal to the vocabulary size. The resulting logits are turned into probabilities with a softmax, and training minimizes cross-entropy against the reference tokens (teacher forcing [194]). At inference time the model generates one token at a time, feeds it back through the masked self-attention, and continues until an end-of-sentence symbol is emitted or a length cap is reached.

2.3 Self-Supervised Learning

The introduction of the transformer architecture marked a pivotal moment in natural language processing, paving the way for the era of pretraining. A key concept in this era is **self-supervised learning** (SSL), which, unlike supervised learning that relies on human-annotated labels and unsupervised learning that finds patterns in data without explicit labels, uses supervision signals directly derived from the input data itself. This is done by designing pretext (or proxy) tasks, which may not be inherently meaningful for any final application, but can guide the model to learn rich, generalizable representations of the data [54, 62]. Once pretrained on such a task, the model can then be fine-tuned to perform effectively on a variety of downstream tasks, even when labeled data is scarce [30, 43].

Self-supervised learning approaches can be categorized into two groups: **intra-sample** and **inter-sample** methods. Intra-sample methods obtain the supervision signal directly from a single sample. For example, in computer vision, this could involve predicting the relative position of patches within an image [44], learning to colourize a grayscale image [207], or inpainting masked patches in an image [130]. In NLP, a classic example is predicting masked words of a text based on its surrounding context [43]. Inter-sample methods, on the other hand, get supervision from relationships between multiple samples, such as learning to pull similar samples closer together in an embedding space while pushing dissimilar ones apart.

2.3.1 Rise of Pretrained Language Models

Following the success of transformers, a dominant approach emerged: pretraining a large model on a huge amount of general-domain text data and then fine-tuning it on smaller, domain-specific datasets for particular tasks. Two of the most influential architectures in this regard are **BERT** (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) [43] and GPT (**G**enerative **P**re-trained **T**ransformer) [139].

BERT Introduced by Devlin et al. [43], BERT uses the encoder part of the transformer architecture. Its primary pretraining objective is **Masked Language Modeling (MLM)**, where a certain percentage of tokens in the input text are randomly masked (e.g., replaced with a special [MASK] token), and the model is trained to predict the original tokens based on the unmasked context. With this objective and a bidirectional nature (considering both left and right context), BERT can learn deep contextual relations between words and the overall structure of language. Another pretraining task used in the original BERT was **Next Sentence Prediction (NSP)**, where the model predicted whether two input sentences were consecutive. However, later work found that MLM is a more effective pretraining objective than

NSP, yielding better downstream performance [106]. Due to its strong language understanding capabilities, a pretrained BERT model can be easily used for a wide variety of downstream tasks, including text classification, sentiment analysis, Named Entity Recognition (NER), and question answering, typically by adding a small task-specific layer and fine-tuning the entire model. The success of BERT led to many variants, such as RoBERTa [106], which optimized pretraining by using dynamic masking and training on more data for longer, and DistilBERT [154], a smaller, faster, and lighter version that preserves most of BERT's performance.

GPT First introduced by Radford et al. [139], GPT uses the decoder part of the transformer architecture. The core self-supervised objective here is **Next Token Prediction** (also known as **Causal Language Modeling**). Given a sequence of tokens, the model is trained to predict the next token in the sequence. This auto-regressive process allows the model to learn how to complete text or generate entirely new content that is coherent and contextually relevant. Scaling up these models (e.g., GPT-2 [140], GPT-3 [21], and subsequent iterations by OpenAI) along with integrating techniques such as instruction fine-tuning and reinforcement learning from human feedback (RLHF) has led to the powerful Large Language Models (LLMs) we see today [127]. These LLMs can perform many tasks in a zero-shot or few-shot manner, meaning they can follow instructions and complete tasks that they were not explicitly trained for, simply by being prompted appropriately.

2.3.2 Contrastive Learning

Contrastive learning is a widely used inter-sample self-supervised method. The main idea is to learn representations by contrasting positive pairs ("similar" samples) against negative pairs ("dissimilar" samples). The model is trained to map these positive pairs close to each other in an embedding space, while simultaneously pushing the representations of negative pairs further apart. This process encourages the model to capture the important underlying features that are most relevant to the notion of similarity defined when pairs are formed.

Contrastive Predictive Coding An early influential work in contrastive methods is CPC [126]. The main idea is to learn rich, useful, and low-dimensional representations by predicting future or contextual information in a sequence or high-dimensional signal, rather than directly predicting future samples in the input space, which can be complex. CPC frames this prediction task as a form of contrastive learning. The model first encodes the input sequence x up to a certain time step t, $x_{\leq t}$, into a context vector $c_t = g_{ar}(x_{\leq t})$, using an autoregressive model g_{ar} (like an RNN, a causal CNN [179] or a masked transformer). Then, separate encoders g_k are used to encode future or target

samples x_{t+k} into representations $z_{t+k} = g_k(x_{t+k})$, where k means steps in the future. The model then predicts these future vectors, z_{t+k}, based on the context, c_t. Rather than directly predicting z_{t+k}, CPC uses a density ratio method. A scoring function

$$f_k(x_{t+k}, c_t) \propto \frac{p(x_{t+k} \mid c_t)}{p(x_{t+k})} \tag{2.3.1}$$

shows how well the context c_t predicts the future sample x_{t+k}, relative to its overall probability. This score is often modelled using a simple log-bilinear model:

$$f_k(x_{t+k}, c_t) = \exp(z_{t+k}^T W_k c_t) \tag{2.3.2}$$

Where W_k is a trainable weight matrix. To learn these representations, CPC uses a contrastive loss called **InfoNCE** (Noise Contrastive Estimation). Given a context c_t and a set of N random samples $\{x_j\}_{j=1}^N$ which has one positive sample (the actual future sample x_{t+k}) and $N-1$ negative samples (randomly drawn from the data distribution), the InfoNCE loss is defined as:

$$\mathcal{L}_{\text{InfoNCE}} = -\mathbb{E}_X \left[\log \frac{f_k(x_{t+k}, c_t)}{\sum_{j=1}^N f_k(x_j, c_t)} \right] \tag{2.3.3}$$

Minimizing this loss is equivalent to maximizing the lower bound on the mutual information between c_t and x_{t+k}, and this leads $f_k(x_{t+k}, c_t)$ to estimate the density ratio in Equation 2.3.1. In simple terms, the model is trained to pick the true future sample x_{t+k} from a set of distractors (negative samples) given the context c_t. By doing so, the model learns powerful representations c_t that have information useful for predicting future states, effectively learning about the underlying structure and dynamics of the data.

SimCLR An influential architecture, especially in computer vision, is **SimCLR** by Chen et al. [30]. SimCLR learns representations by maximizing agreement between two augmented views of the same sample (an image in the original variant) using a contrastive loss. The framework has the following key steps (illustrated in Figure 2.5):

1. **Data Augmentation:** For each data sample x_i in a mini-batch, two related views, $\tilde{x}_i$ and $\tilde{x}_j$, are generated by applying some random data augmentations (e.g., random cropping, resizing, colour jittering, Gaussian blur, sampled from $\mathcal{T}$). These two views $(\tilde{x}_i, \tilde{x}_j)$ form a positive pair.

2. **Encoder Network:** A base encoder $f(\cdot)$ (e.g., a ResNet [61]) creates representation vectors from each augmented samples. Let $h_i = f(\tilde{x}_i)$ and $h_j = f(\tilde{x}_j)$.

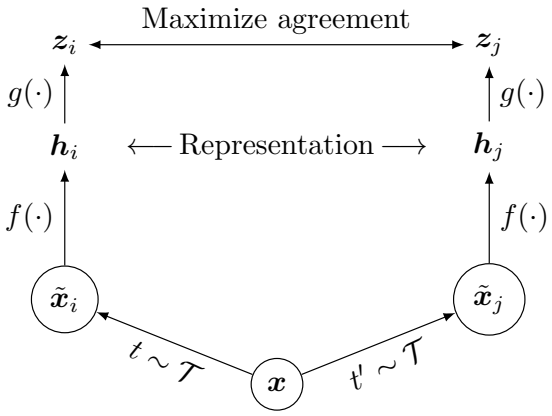

Figure 2.5: Illustration of a basic architecture for contrastive learning that enables the model to produce effective visual embeddings. Reprinted from Chen et al. [30]

3. **Projection Head:** A small neural network, the projection head $g(\cdot)$ (typically an MLP), maps these representations h to a lower-dimensional space where the contrastive loss is applied. So, $z_i = g(h_i)$ and $z_j = g(h_j)$. The authors found that applying the contrastive loss on z rather than h leads to better representations h for mainstream tasks.

4. **Contrastive Loss Function:** Given a positive pair (z_i, z_j), all other $2(N-1)$ augmented examples in the mini-batch (where N is the batch size) serve as negative examples. The loss function is the normalized temperature-scaled cross-entropy loss (NT-Xent):

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)} \tag{2.3.4}$$

Here, $\text{sim}(u, v) = \frac{u^T v}{\|u\|\|v\|}$ means the cosine similarity between vectors u and v, τ is a temperature parameter that adjusts how peaked or smooth the similarity distribution becomes, and $\mathbb{1}_{[k \neq i]}$ is an indicator function that ensures the positive sample z_i is not counted among an examples negatives. The final loss is computed across all positive pairs in the mini-batch (i.e., (i, j) and (j, i)).

2.4 Text Embedding Models

Early neural IR systems, built on word2vec and GloVe, assigned a single, context-free vector to every word; thus, homonyms such as "bank" (financial vs. river edge) mapped to the same point in space. Sentence or document representations were built by aggregating word vectors (e.g., simple averaging or addition). This was computationally attractive, yet the resulting sentence embeddings were semantically too general and insensitive to word order. When transformer encoders like BERT were introduced, they solved polysemy by computing contextual token embeddings, and each word's vector is conditioned on its entire sentence. When used in a cross-encoder setup, BERT concatenates a query and a candidate passage in the same input, and self-attention captures detailed interactions between the two. However, comparing one query with many passages requires N full forward passes ($O(N)$), which is impractical for large-scale retrieval. This motivated the bi-encoder paradigm: encode the query and each document independently, store document embeddings in advance, and perform retrieval with vector similarity search. Early naïve bi-encoders that simply reused BERT's raw [CLS] vector (or mean pooling) performed worse than even averaged GloVe on STS tasks [145], showing the need for task-specific training that could learn contextual embeddings with meaningful geometry for similarity.

2.4.1 Siamese & Triplet Network - Sentence-BERT

Sentence-BERT (SBERT) [145] uses a Siamese architecture that consists of two identical transformer encoders that share the same weights during training. At inference time, a single encoder can convert any text (query or passage) into a fixed-size vector, then compare it using cosine similarity or dot product. This architecture enables $O(N)$ retrieval: all documents can be pre-encoded and indexed, and each query requires only a single forward pass to obtain its embedding, which is then matched against the index. By contrast, a BERT cross-encoder must process each query-document pair individually, resulting in $O(N)$ passes and $O(N)$ per query. SBERT is trained with three interchangeable objectives:

1. **Classification loss (softmax):** Given a set of sentence embeddings (u, v) as input, SBERT applies a classifier to predict their class label. It uses Natural Language Inference (NLI) data (e.g., SNLI [16], MultiNLI [193]) where each sentence pair is labeled as **entailment**, **neutral**, or **contradiction**. The two embeddings u and v are combined (e.g., concatenation $[u; v; |u - v|]$) and fed into a softmax classifier to predict the NLI label. A cross-entropy loss

$$L_{cls} = -\sum_{c} y_c \log \hat{y}_c \tag{2.4.1}$$

encourages the model to assign high similarity to entailment pairs and low similarity to contradictions. This classification objective teaches the model useful semantic distinctions: entailment pairs are pulled closer in vector space, while contradictory pairs are pushed apart.

2. **Regression loss (MSE):** Model is directly trained to output cosine similarities that match human judgments. Using labeled Textual Similarity (STS) data (e.g., STS benchmark [25]), the model encodes two sentences into embeddings u and v and computes $cos(u, v)$ as the predicted similarity. The training objective minimizes the **mean squared error** between the predicted cosine and the true similarity score s_{true}(typically on a 0-5 scale): $L_{reg} = \big(\cos(u, v) - s_{\text{true}}\big)^2$. This continuous regression loss aligns embedding dot-products with graded semantic similarity, forcing the model to treat vector proximity as a fine-grained similarity measure.

3. **Triplet loss (margin ranking):** In this setup, training samples are triplets (a, p, n) where a is an anchor sentence, p is a positive sentence (semantically similar to the anchor), and n is a negative sentence (dissimilar). The objective is to make the distance between the anchor and positive at least a certain margin smaller than the distance between the anchor and negative. The triplet loss is formally written as:

$$L_{\text{triplet}} = \max\left\{0,\ \Delta + d(a, p) - d(a, n)\right\}, \tag{2.4.2}$$

where $d(x, y)$ is a distance metric (e.g., cosine distance or Euclidean distance in embedding space) and Δ is a margin hyperparameter. If the positive is already at least Δ closer to the anchor than the negative, the loss is zero. Otherwise, the model is penalized proportionally to the extent to which this margin is violated. Sampling **hard negatives** (negatives that are deceptively close to the anchor) encourages the model to learn finer-grained semantic distinctions.

SBERT is usually trained in two phases: first on a large NLI dataset using a classification loss to learn semantic distinctions, and then optionally on STS data with a regression loss to directly optimize similarity scores. Alternatively, NLI sentence pairs can be converted into triplets (treating entailment pairs as anchor-positive and contradiction as hard negatives) to train with a triplet objective. The overall training set often combines millions of NLI pairs and STS benchmarks or other curated paraphrase sets. When fine-tuned on SNLI and STS data, SBERT achieves near-human performance on STS Benchmark (Spearman $\rho \approx 0.85$), whereas averaged GloVe vectors only reach $\rho \approx 0.58$ and BERT-[CLS] barely $\rho \approx 0.20$. Beyond accuracy gains, SBERT enables efficient retrieval. The bi-encoder maps any text to a fixed 768-dimensional vector (for BERT-base models), allowing retrieval to be performed as

a nearest-neighbour search in this vector space. Computing m document embeddings and n query embeddings is $O(m+n)$, and similarity search via dot-product or cosine can be accelerated with **Approximate Nearest Neighbour** indexes. In contrast, a cross-encoder would need $O(m \cdot n)$ operations to compare n queries with m documents. The original article reports that SBERT-based semantic search is more than 100 times faster than BERT cross-encoders on large corpora, with only a slight drop in accuracy. SBERT provides efficient, scalable semantic search by training sentence embeddings with a Siamese architecture and similarity-based losses, where vector proximity directly reflects semantic similarity.

2.4.2 Two-Stage Contrastive Training

Recent advances show that large-scale contrastive learning on unlabelled text pairs can result in extremely powerful embeddings. Contrastive learning aims to teach the model which texts to bring together or push apart in vector space, often using only weak or automatically generated pairs. Early methods, such as CPC and SimCLR, laid the foundation by introducing the InfoNCE loss to train encoders without explicit labels. The InfoNCE loss operates on pairs of positive examples and a set of negative examples. In the context of text embedding models, a positive pair might be a (query, relevant passage) or two paraphrased sentences, and negatives are unrelated texts. Given a query (anchor) q_i and its true p_i (positive), along with m negative passages $\{p_{ij}^-\}_{j=1}^m$ for that same query, the InfoNCE objective for a batch of N examples is defined as:

$$L_{\text{InfoNCE}} = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{s(q_i,p_i)}}{e^{s(q_i,p_i)} + \sum_{j=1}^{m} e^{s(q_i,p_{ij}^-)}}\,, \tag{2.4.3}$$

where $s(p,q) = \cos(\mathbf{E}_q, \mathbf{E}_p)/\tau$ is a scoring function (typically the cosine similarity between the query embedding E_q and passage embedding E_p, optionally scaled by a temperature τ). Intuitively, the loss forces $s(q_i,p_i)$ to be higher than $s(q_i,p^-)$ for any negative p^-. This way, the model learns to tell true associations from random ones. The InfoNCE formulation is implemented efficiently with in-batch negatives: when a batch contains many (query, positive) pairs, each pair's positive passage serves as a negative for the others. A large batch size is critical and has been shown to greatly improve representation quality by providing more negative examples for each update [30, 126].

Modern embedding models typically use a two-stage training recipe with contrastive learning. An example is the E5 model family [185], which introduced this training process in two stages: (1) large-scale contrastive pretraining on weak text pairs, and (2) supervised fine-tuning on high-quality data with hard negatives. In the first stage, a massive corpus of unlabelled text pairs (1.3 billion) are gathered from sources

that naturally form "query-passage" pairs, such as (question, answer) from Reddit or StackExchange, (article title, section text) from Wikipedia, (paper title, abstract) from academic papers, and (article headline, content) from news or Common Crawl web pages. Although these pairs often share a topic, they are not manually labeled and are therefore quite noisy. To further improve data quality, a **consistency-based filtering** step is applied. After initial cleaning, they train a draft bi-encoder on 1.3 billion candidate pairs. Next, for each query-document pair, they use the trained model to score the query against its true document and a pool of one million randomly sampled negatives. A pair is kept only if, according to the model's similarity scores, the true document ranks within the top 2 results for its query, meaning the model itself is confident this association is unusually strong, even compared to a vast set of distractors. This filtering step removes noisy pairs with weak semantic ties, reducing the training set to approximately 270 million high-consistency pairs.

Contrastive Pretraining Using this refined corpus (called CCPairs), the first stage of E5 training begins. Each text pair is treated as a positive q-p while all other texts in the batch serve as negatives (often called **in-batch negatives** [30]). The bi-encoder (initialized from a pretrained encoder model like BERT) is trained with the InfoNCE loss described above. Notably, E5 prepends special prefix tokens **"query:"** or **"passage:"** to each input to break symmetry and help a single encoder distinguish between query-like and passage-like inputs, effectively encoding them in slightly different subspaces. By the end of stage 1, the model (called E5-PT) has learned a general-purpose embedding space from hundreds of millions of noisy pairs. This contrastive pretraining already results in strong unsupervised embeddings, outperforming earlier models on retrieval benchmarks like BEIR [175] (e.g., E5-PT large scored nDCG@10=44.2%, vs. 36.0% for earlier Contriever [74]). This result highlights how both scale and data quality in contrastive learning result in substantial gains in retrieval effectiveness.

Contrastive Fine-tuning The second stage often involves using triplets and hard negatives to improve the model for real-world retrieval. Training triples are drawn from MS MARCO Passage Ranking [121], Natural Questions [94, 82], and NLI [16], each query paired with one positive passage and up to seven hard negatives mined by a prior retriever (e.g., SimLM [184]). Hard negatives are passages that closely match the query but are labeled as incorrect, typically top-ranked false positives from other systems. They serve as challenging decoys, helping the model learn fine-grained distinctions. The bi-encoder is optimized with a two-term objective:

$$L = L_{\text{InfoNCE}} + \lambda L_{\text{dist}}, \quad \lambda \approx 0.5, \tag{2.4.4}$$

where L_{InfoNCE} is the InfoNCE contrastive loss defined earlier, which uses a predefined list of negatives instead of in-batch negatives, and the *cross-encoder distillation* term

$$L_{\text{dist}} = \text{KL}\left(\text{softmax}(t/\tau) \,\|\, \text{softmax}(s/\tau)\right) \tag{2.4.5}$$

matches the bi-encoder scores s_i to teacher cross-encoder scores t_i (with temperature τ). Many models in the literature skip this distillation step and rely solely on a contrastive objective with hard negatives. Fine-tuning on roughly 30 million triplets for a few epochs improves E5-large to ≈ 58 nDCG@10 on BEIR, outperforming earlier bi-encoders like GTR and Contriever; omitting hard negatives or distillation causes performance to plateau. Fine-tuning E5-large on approximately 30 million triplets for a few epochs raises its BEIR nDCG@10 to about 58, outperforming earlier bi-encoders such as GTR [122] and Contriever [74].

It is worth noting that contrastive pre-training followed by hard-negative fine-tuning has become the prevailing approach for training text embedding models. Most leading models adopt this two-stage approach. First, they leverage weak supervision at scale (e.g., mining pairs from forums, wiki, news) to train a strong general-purpose initial encoder; then they apply supervised fine-tuning on carefully curated data with hard negatives to refine the encoder specifically for retrieval tasks.

2.4.3 Modern Encoder-Based Families

Following the foundational work in bi-encoder architectures and two-stage training, recent research has introduced several new families of embedding models that push the boundaries of performance and capability. These modern encoders still employ the same core ideas but incorporate new architectural tweaks, training methods, and significantly larger datasets.

E1 (v1 & v2) Developed by Microsoft, the E5 models were among the first to show how two-stage training could work in practice [185]. E5 v1 was released in `small`, `base`, and `large` sizes, its embeddings ranged from 768 to 1024 dimensions, and used a single BERT encoder for both queries and documents by simply prepending "query:" or "passage:" to each input. The next version, E5 v2, kept the same core architecture but improved performance by training on larger, more diverse data and refining the fine-tuning steps. The improvements in E5 v2 suggest that, even without major architectural changes, scaling data and fine-tuning can lead to significant gains.

BGE (v1 & v1.5) The BAAI General Embedding (BGE) [200] introduced a novel pretraining technique called **RetroMAE** (Retrogressive Masked Auto-Encoder). Inspired by **Masked Autoencoders (MAE)** [63] in computer vision, RetroMAE first pollutes text embeddings and then trains a lightweight decoder to reconstruct the original embeddings. After this pretraining, BGE follows the same two-stage process of large-scale contrastive learning and instruction tuning. BGE v1.5 further refined this approach, resulting in strong performance without altering the core model architecture.

GTE (v1 & v1.5) Alibaba's DAMO Academy developed the GTE models with a focus on building versatile embeddings. The original GTE used a multi-stage contrastive learning process over various public datasets, all within a typical BERT-style context window of 512 tokens [102]. These early versions already showed strong results. With GTE v1.5, they introduced an encoder capable of handling much longer inputs, up to 8,192 tokens, by adding **Rotary Position Embeddings (RoPE)** [172] and **Gated Linear Units (GLU)** [161] in the feed-forward layers. These enhancements make GTE v1.5 especially effective for tasks involving very long documents [208].

Nomic (v1, v1.5, v2) Nomic AI has initially released `nomic-embed-text-v1`, which is notable for its transparency, with the code, data (235 million text pairs), and model weights being publicly available [124]. To train their model, they initially used a long-context BERT architecture called `nomic-bert-2048`, enhanced with improvements such as RoPE, GLU, and FlashAttention, which integrate successful elements from other models to facilitate more efficient training on long sequences. This BERT model has a 2048 context length, which can then be interpolated to handle up to 8192 tokens at inference. Similar to E5, Nomic employs a two-stage training process and uses prefixes. Notably, their open-source `nomic-embed-text-v1` has been shown to outperform some proprietary models, such as OpenAI's `text-embedding-3-small` and `text-embedding-ada-002`, in certain benchmarks. Building on this success, `nomic-embed-text-v1.5` integrated **Matryoshka Representation Learning (MRL)** [93], which allows a single model to output embeddings of different dimensions (e.g., from 768 down to 64) by ensuring that shorter prefixes of the full embedding are themselves effective representations. This offers flexibility in balancing performance and resource usage. More recently, Nomic released `nomic-embed-text-v2-moe`, which is the first open-source multilingual **Mixture of Experts (MoE)** embedding model [162, 76]. MoE architectures comprise multiple expert sub-networks and a gating mechanism that determines which experts to activate for a given input. This enables models to scale capacity efficiently, often resulting in better performance with a similar computational cost during inference

compared to dense models of equivalent size.

BGE M3 BAAI's BGE M3 (Multi-lingual, Multi-function, Multi-granularity) represents a significant step towards a unified embedding model [27]. It supports over 100 languages and has a context length of up to 8192 tokens. Its main innovation is the ability to perform dense retrieval, sparse retrieval (like SPLADE [49]), and multi-vector retrieval (like ColBERT [84]) in one model, which is achieved through self-knowledge distillation, where the model learns to make its different output representations consistent with each other. BGE M3 also excels in cross-lingual retrieval capabilities, allowing a query in one language to retrieve documents in another language.

Multilingual E5 To address the need for strong embedding models beyond English, Multilingual E5 models were developed [187]. These models typically use a multilingual BERT (such as XLM-R [35]) as their base backbone and are trained on large-scale, multilingual text pairs. They demonstrate strong performance in both monolingual retrieval across multiple languages and cross-lingual retrieval tasks, making them valuable for multilingual search applications.

Decoder-based models Researchers have recently shown how to turn decoder-only LLMs into efficient bi-encoders by reusing their existing weights [9]. Starting from a causal transformer, one option is to keep the mask in place but simply take (pool) the final <EOS> hidden state as that text's embedding [186]. Another option is to remove the causal (triangular) attention mask, allowing every token to attend to every other token, and making attention bidirectional. Then, standard pooling choices can apply, such as mean, max, first-token, or EOS, which result in a single vector. Then a pair of towers can be fine-tuned with a contrastive InfoNCE loss and mined hard negatives, exactly as in E5. This exact procedure lies behind models like `E5-Mistral-7B-instruct`, `GTE-Qwen2-7B-instruct`, `GTE-Qwen1.5-7B-instruct`, and `multilingual-E5-large-instruct`. Despite having between 550 million and 7 billion parameters, these models now rank among the top performers on MTEB [118]. Their strength stems from the extensive causal-LM pretraining, which equips the encoder with broad world knowledge and even basic reasoning skills. The downside is that they are expensive: in FP16 precision, each model requires over 14 GB of GPU memory, and only a handful of queries can be processed per second on a single A100 GPU.

Proprietary Models Several vendors offer proprietary embedding services that can only be used via hosted APIs. OpenAI provides `text-embedding-ada-002` and the newer `text-embedding-3-small` and `text-embedding-3-large`. Amazon Bedrock serves `amazon.titan-embed-text-v1` and its 2025 upgrade

`amazon.titan-embed-text-v2.0`. Cohere's lineup includes `embed-english-v3` and `embed-multilingual-v3`. Google Vertex AI has released `gemini-embedding-001` and related experimental variants. While all of these models achieve strong zero-shot performance on public benchmarks, their internal architectures and training procedures remain undisclosed.

Bi-encoders today range from lightweight 100-million-parameter models that can handle tens of queries per second to multi-billion-parameter decoder hybrids that improve benchmarks only marginally but run two orders of magnitude slower. In practice, production systems utilize efficient, mid-sized open-source checkpoints for first-pass retrieval and reserve a cross-encoder model for a second pass of reranking to further improve accuracy, where extra compute is justified when dealing with a short list of candidates.

2.4.4 Late-Interaction & Hybrid Dense-Sparse Models

ColBERT Late-interaction models aim to blend the lexical exactness of sparse search with the contextual richness of dense encoders while preserving offline indexing. Unlike a standard bi-encoder that pools a document into one vector, ColBERT [84, 155] encodes every token in a query and a document into separate low-dimensional vectors and scores a pair by summing token-wise maxima, for each query token vector q_i, it computes the cosine similarity with every document token vector d_j, takes the maximum similarity $\max_j \cos(q_i, d_j)$, and then sums these maxima over all query tokens. More formally, if $Q = q_1, \ldots, q_m$ and $D = d_1, \ldots, d_n$ are the sets of embeddings for the query and document, the ColBERT score is:

$$\text{score}(Q, D) = \sum_{i=1}^{m} \max_{1 \le j \le n} \text{sim}(q_i, d_j). \tag{2.4.6}$$

Because the document is still pre-encoded, retrieval scales linearly with corpus size, yet the token-level comparison recovers nuanced matches. For example, consider the query "dog training tips". A ColBERT model might use one vector for "dog" to find related words like "puppy", and another for "training" to find words like "commands" in the document, combining their scores. A single dense vector might not pick up on these specific word matches. A ColBERT store remains an order of magnitude larger than a dense index, and each query incurs dozens of dot products per token, so the method is typically used as a second-stage re-ranker.

SPLADE A complementary method retains the familiar inverted index but utilizes a transformer to determine which terms to include [49]. It utilizes the hidden states

of a BERT model and employs its masked language model head to score every word in the vocabulary (approximately 30,000 terms). Then, by applying L1 or FLOPS regularization, it forces almost all scores to zero, leaving only a few active terms per document or query, which can be indexed and scored exactly like TF-IDF. Instead of relying on fixed query expansion, SPLADE learns to activate semantically related terms. For example, given "smartphone battery life", it might choose "battery", "longevity", and "phone". This learning happens by distilling knowledge from a strong cross-encoder and by tuning separate sparsity levels for queries versus documents. SPLADE v2 [48] adds hard negatives during training and adjusts activation weights based on document length, reaching BEIR performance on par with dense models while remaining fully interpretable and indexable. The primary overhead is computing scores for every vocabulary term in each query; however, aggressive pruning and caching make SPLADE fast enough to handle web-scale collections.

Most production systems now combine dense, sparse, and late-interaction signals. First, a medium-sized bi-encoder (e.g., E5, GTE, etc) retrieves a few hundred candidates via approximate nearest-neighbour search. Next, a sparse model (e.g, BM25 or SPLADE) finds additional documents based on exact term matches. Finally, the combined set is re-ranked using a late-interaction model, such as ColBERT or a cross-encoder. In practice, these three views cover different weaknesses: dense embeddings catch paraphrases, sparse scores handle out-of-vocabulary phrases, and token-level MaxSim handles precise intent. Top systems in the BEIR challenge and default settings in toolkits like Pyserini [104] employ this layered approach, achieving cross-encoder accuracy while keeping latency and hardware costs manageable. As a result, hybrid search has become the standard for large-scale retrieval, enabling the balance of recall, precision, and speed.

2.5 Retrieval Evaluation Metrics

In this section, we introduce standard metrics for evaluating retrieval and ranking models. Modern retrieval research relies on several rank-focused measures, each highlighting a different aspect of system quality. Precision@K and Recall@K measure result purity and coverage, but do not consider rank order. MAP@K and MRR@K reward obtaining correct results early, assuming binary relevance. nDCG@K handles graded relevance, making it the single metric reported in large comparative studies, such as BEIR. Below, we define each metric formally and explain how it can both clarify and sometimes distort experimental outcomes.

Precision @ K It is defined as the ratio of relevant documents among the top K results:

$$P@K = \frac{1}{K}\sum_{i=1}^{K} \text{rel}_i \tag{2.5.1}$$

where rel_i is 1 if the i-th document is relevant and 0 otherwise.This metric is easy to understand and aligns with how users typically view only the first few results on a page. However, it has two main drawbacks. First, it does not care about the order of those top K results; swapping a relevant document at rank 1 with one at rank 20 does not change $P@K$. Second, it ignores any documents ranked lower than K, which means it does not reflect performance when users need to find many relevant documents. Finally, by treating relevance as just relevant or not, it misses cases where some documents are more useful than others.

Recall @ K It measures how many of the total relevant documents appear in the top K results:

$$R@K = \frac{1}{|R|}\sum_{i=1}^{K} \text{rel}_i \tag{2.5.2}$$

where $|R|$ is the total number of relevant items. This metric is crucial when missing even one relevant document is unacceptable, such as in legal discovery or literature reviews. However, it shares Precision@K's blind spot to ordering: swapping a relevant result at rank 1 with one at rank K does not change Recall@K. It also relies on knowing every relevant document in advance; if some relevant items are unjudged (i.e., not counted in $|R|$) and appear below K, the score appears artificially higher. Finally, on simple queries with only a few relevant documents, recall can reach 100% very quickly, which makes comparing results across different datasets unreliable.

MAP @ K (Mean Average Precision) This metric finds the precision at each position where a relevant document appears, up to rank K, and then averages those precision values (dividing by the smaller of K or the total number of relevant documents). Mean Average Precision (MAP@K) is simply the average of AP@K across all queries:

$$AP@K = \frac{1}{\min(|R|, K)}\sum_{i=1}^{K} P@i \,\text{rel}_i \tag{2.5.3}$$

Because AP gives a reward each time a relevant document appears, MAP favors systems that place many good documents near the top. However, MAP treats relevance as a yes/no decision, so it ignores any differences in how relevant documents might be on a scale of relevance. Studies also show that MAP can fluctuate significantly if

some relevant documents are missing from the relevance set; small changes in which documents are judged can substantially alter rankings [196, 197]. Finally, MAP averages every query equally, regardless of its difficulty. That means a handful of very difficult queries (with few relevant documents) can dominate the overall score and hide how the system performs on more common, everyday searches.

MRR @ K (Mean Reciprocal Rank) It focuses only on the position of the very first relevant document and assigns a score equal to the reciprocal of that document's rank, as long as it appears within the cutoff K.

$$RR@K = 1/\text{rank}_{\text{first}} \tag{2.5.4}$$

This makes MRR especially useful for tasks like factoid question answering or conversational agents, where users usually stop after finding the first satisfactory answer; an increase in MRR directly means answers are appearing earlier. However, MRR ignores any additional relevant items, offering no reward for returning a second or third correct document, and it treats relevance as a strictly binary concept. Due to its reciprocal formula, even slight shifts in rank can have a significant impact. For example, moving the first hit from position 2 to 3 reduces its contribution by half, allowing MRR scores to fluctuate significantly despite minimal perceptual differences for users.

nDCG @ K (Normalized Discounted Cumulative Gain) This metric adds up graded relevance scores but penalizes results that appear deeper in the list. Formally:

$$DCG@K = \sum_{i=1}^{K} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)} \tag{2.5.5}$$

Here, the exponential term $(2^{rel_i} - 1)$ lets us handle multi-level relevance judgments (for example, "highly relevant" versus "somewhat relevant"), and dividing by $\log_2(i+1)$ reflects the fact that users focus more on top-ranked items. To make comparisons fair, we divide DCG@K by the maximum possible DCG@K (the ideal ranking) to get normalized DCG (nDCG). Normalization enables us to compare different queries and systems on a standard scale. One useful property of nDCG, which is consistent distinguishability, is that, under broad conditions, its ordering of two systems tends to match human preferences. In other words, if people consistently like system A more than B, nDCG will usually rank A above B—even though nDCG values move closer to 1 as K increases [190]. Because nDCG combines sensitivity to both rank position and graded relevance, many benchmarks (such as BEIR) use nDCG@10 as their primary metric. However, nDCG has its own limitations: it requires knowing

the best possible ranking in advance so that any unjudged relevant documents can inflate scores. Also, choosing a different discount function (for example, $\frac{1}{i}$ instead of $\frac{1}{\log_2(i+1)}$ changes absolute values, making it hard to compare results unless everyone follows the same formula.

In summary, each metric highlights a different aspect of performance: purity, completeness, speed of finding results, or graded usefulness. A thorough evaluation, therefore, encompasses all these measures, allowing readers to judge from multiple angles instead of relying on a single number that might be misleading.

2.6 Synthetic Data Generation

Large Language Models are now used not just for performing tasks but also as data generators, helping to overcome issues like scarce or imbalanced datasets. For instance, **NVIDIA's Nemotron-4 framework** [2] relied on over 98% synthetic examples to fine-tune its instruction-following abilities. Such synthetic corpora can rephrase real text, produce labeled examples, or fill gaps in underrepresented domains. By automating prompt creation, response generation, and rigorous filtering, NVIDIA's pipeline produces high-quality training and preference modelling data without the expense of human annotation. This trend, where powerful LLMs generate data to train other models, represents a significant shift in NLP, enabling rapid and cost-effective expansion into new domains.

One particularly successful application of synthetic data is **information retrieval**, particularly in training neural rankers and retrievers, where large collections of labeled query-document pairs are scarce. Recent methods use LLMs to generate realistic queries for existing documents, yielding pseudo-labeled pairs without any human annotation. **InPars** [14] is a notable example; it prompts GPT-3 with a few demonstration examples to produce relevant queries for a given passage, creating enough synthetic pairs to train a dual encoder or reranker without any human-labeled queries. Building on this idea, **InPars-v2** [77] switches to open-source LLMs and filters the generated pairs through a strong reranker, resulting in higher-quality data and state-of-the-art zero-shot retrieval on BEIR. **Promptagator** [38] takes a similar approach with as few as eight examples and task-specific prompts to elicit queries from an LLM, then applies a round of consistency checks to discard low-quality queries. Researchers have also extended this strategy to multilingual and cross-domain settings [176]; they demonstrate that a "summarize-then-ask" strategy (first having an LLM summarize a document, then generating queries for the summary) yields rich training sets in dozens of languages. LLM-generated query-document pairs provide abundant,

high-quality training samples that rival traditional supervised data in training neural ranking models.

Synthetic data has also become a powerful tool for training text embedding models used in semantic search. For instance, Wang et al. [186] show that you can train a general-purpose encoder almost entirely on LLM-generated pairs. The recipe is simple: first prompt a large LLM to produce diverse sentence pairs and related snippets across multiple tasks and languages. Then, use those synthetic pairs to fine-tune a smaller transformer with a contrastive loss. Despite using only synthetic data, the resulting embedding model achieves competitive performance on standard semantic textual similarity and retrieval benchmarks. Adding even a small amount of real labeled data on top pushes it to state-of-the-art on benchmarks such as BEIR and MTEB. The real advantage lies in the variety and difficulty of the generated examples, which effectively form an automatic curriculum that would be impractical to build by hand. By controlling the design of prompts, researchers can tailor the tone, complexity, and domain of the synthetic data, ensuring that the final embeddings are both robust and well-matched to their intended applications.

Recent studies highlight a clear message: Large Language Models can now produce diverse, high-quality training pairs at a scale that manual annotation cannot match. This generated data provides a practical and low-cost alternative source of supervision. This complements the neural ranking and contrastive learning techniques discussed earlier, and it is likely to become even more critical for building robust text embedding and retrieval systems in data-scarce or rapidly changing domains.

2.7 Retrieval-Augmented Generation

Large pretrained language models (LLMs) implicitly store knowledge in their parameters, but this parametric memory is static and cannot be easily updated or inspected [137, 146]. As a result, LLMs may struggle with queries about new or niche information and can confidently "hallucinate": generate outputs that sound plausible but are factually incorrect. One way to introduce new knowledge is by fine-tuning or retraining the model on updated data; however, updating the model's weights is often costly and impractical, especially for very large or closed-source language models [112]. Retrieval-Augmented Generation (RAG) has been introduced as a more practical alternative that allows LLMs to access external knowledge at generation time without changing their weights [100]. In RAG, the model is equipped with a retriever that fetches relevant text from an external knowledge base at inference time, and this retrieved context is provided to the model alongside the

query. By grounding every response in external documents, the model can incorporate up-to-date or domain-specific information on the fly. This approach has been shown to improve the factual accuracy of generated answers and reduce hallucinations, as the language model can rely on retrieved evidence rather than guessing from parametric memory. For example, Shuster et al. [166] report a 30-35% drop in fabricated answers when using RAG for open-domain tasks. Moreover, integrating external knowledge can make smaller models as effective as much larger ones; a 7-billion-parameter retrieval-augmented model can match the performance of a 175B model on knowledge benchmarks by accessing a large text index at runtime [15]. RAG proceeds in two stages:

1. **Retrieval:** An embedding model maps the query q to a vector $E(q)$ and each document d to $E(d)$. The retriever then finds the top-k documents $d_1, d_2, \ldots, d_k$ whose embeddings have the highest similarity (cosine or dot) to $E(q)$, formally we have:
$$d_i = \arg\max_{d \in \mathcal{C}} \text{sim}(E(q), E(d)) \tag{2.7.1}$$
for $i = 1, \ldots, k$, where $\mathcal{C}$ is the document corpus and $\text{sim}(\cdot, \cdot)$ is a similarity function. The set of retrieved passages $D = d_1, \ldots, d_k$ is then appended to the original query as extra context.

2. **Generation:** The language model G (e.g. a pretrained transformer decoder or seq2seq model) processes the query along with the retrieved documents to generate a final answer y. Essentially, the model now generates text conditioned on external knowledge, modelling the probability of an output y given the query and retrieved content, $P(y \mid q, D)$.

Figure 2.6 illustrates the Retrieval-Augmented Generation (RAG) architecture, which consists of two primary stages. In the first stage, documents are processed for indexing. They are typically segmented into smaller text chunks to accommodate the limited context windows of embedding models. For instance, the context window for standard BERT models is 512 tokens. These chunks are then converted into fixed-size numerical vectors by an embedding model. Finally, the resulting vectors are stored in a vector database [80], a data structure optimized for efficient similarity search in high-dimensional spaces. The **second stage** occurs at inference time. An incoming query is converted into a vector using the same embedding model. A similarity metric, in conjunction with an efficient search algorithm like Approximate Nearest Neighbours [109], is employed to retrieve the top-k most relevant document vectors from the database. The text chunks corresponding to these vectors are then used as supplementary context. This context is concatenated with the original query and provided as a single input to a large language model, which generates a response that is grounded in the retrieved information and returns it to the user.

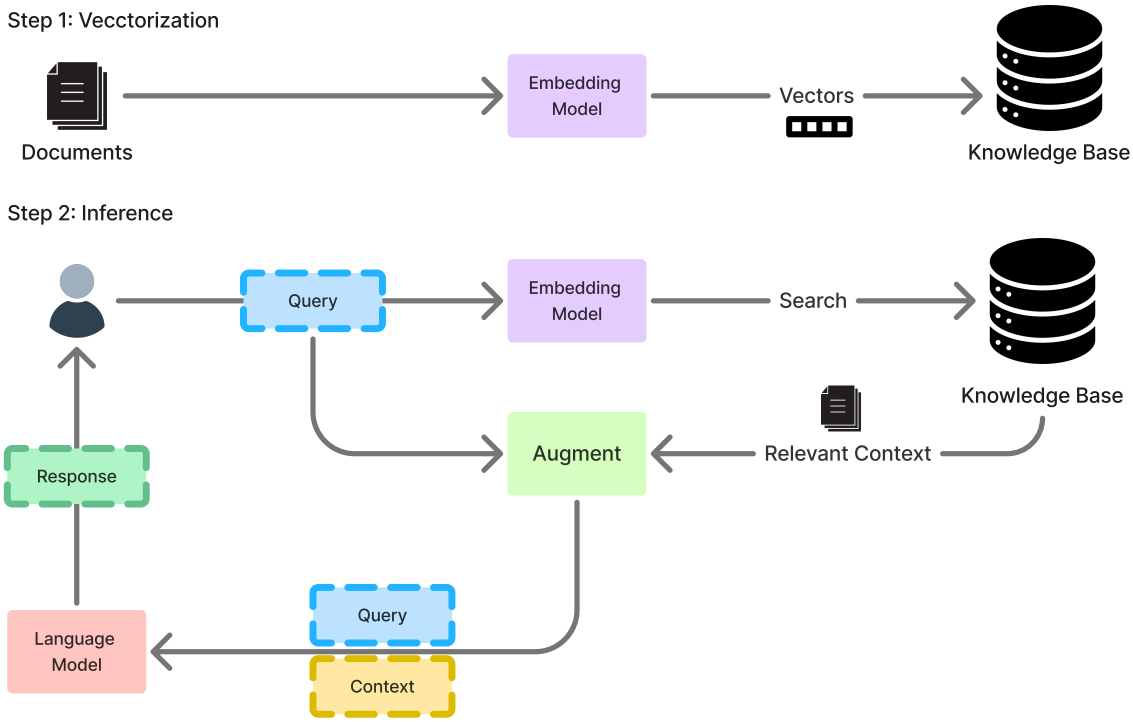

Figure 2.6: A retrieval-augmented generation pipeline

This retrieve-and-generate approach extends earlier "open-book" QA systems that used information retrieval with a reader model [26], and it builds on advances in integrating differentiable retrieval into neural networks [58]. In particular, the original RAG implementation fine-tunes a BART seq2seq model with a Dense Passage Retriever (DPR) backend [100], enabling end-to-end learning of both retrieval and generation. Overall, Retrieval-Augmented Generation provides a practical approach to incorporating up-to-date, non-parametric knowledge into LLMs, and it has become a foundational technique for knowledge-intensive tasks in NLP, where purely parametric models were previously prone to factual errors.

2.8 NLP & Text Embedding Benchmarks

2.8.1 General Encoder-Based NLP Benchmarks

General natural language understanding benchmarks, such as **GLUE** (General Language Understanding Evaluation) [181] and **SuperGLUE** [182], are widely used to evaluate encoder-based models. GLUE combines nine sentence-level or sentence-pair tasks, spanning sentiment analysis, linguistic acceptability, paraphrase detection,

natural language inference (NLI), and semantic textual similarity into a single score that measures overall natural language understanding ability. SuperGLUE builds on GLUE with a more challenging suite of tasks (e.g., commonsense reasoning, coreference resolution in Winograd schemas, multi-sentence inference), an updated private test set, and a leaderboard for more difficult language understanding problems. Additionally, multilingual extensions have been introduced. The **XTREME** [167] benchmark (Cross-lingual TRansfer Evaluation of Multilingual Encoders) covers 9 tasks across 40 languages to evaluate how well representations transfer between languages. **XGLUE** [103] offers 11 cross-lingual tasks across 19 languages, covering both understanding and generation. For Chinese, **CLUE** (Chinese Language Understanding Evaluation) [203] brings together 9 diverse single-sentence and sentence-pair classification tasks as a Chinese counterpart to GLUE. Classic task-specific evaluations are also used; for example, the TREC question classification dataset [37] (6,000 questions labeled with coarse and fine-grained categories) continues to serve as a benchmark for assessing sentence-level classification models. All being said, these benchmarks focus on classification, entailment, and sentence comprehension, and provide a standard framework for evaluating encoder models' understanding and transfer learning.

2.8.2 Text Embedding Evaluation Benchmarks

Beyond general natural language understanding, several benchmarks have been designed specifically to evaluate sentence or text embeddings and the quality of vector representations. One fundamental benchmark is the **Semantic Textual Similarity Benchmark** (STS) [25], which evaluates how well embeddings capture semantic similarity by comparing model-derived similarity scores with human judgments on sentence pairs. **SentEval** [25] is another famous evaluation framework that tests sentence embeddings on a variety of downstream tasks such as sentiment analysis, question-type classification, and NLI by feeding the embeddings into simple classifiers. One of the recent and most critical large-scale benchmarks is the **Massive Text Embedding Benchmark** (MTEB) [118], which includes eight task categories: classification, pair classification, clustering, retrieval, reranking, STS, and summarization, covering +100 tasks and +1000 languages as of mid 2025. MTEB provides a comprehensive picture of embedding performance across diverse scenarios, with results showing that no single model outperforms all tasks. In information retrieval, the **BEIR** benchmark [175] (**Be**nchmarking **I**nformation **R**etrieval) includes 18 heterogeneous retrieval datasets (spanning web search, QA, fact checking, etc.) to evaluate different types of retrievers such as lexical, dense, sparse, late-interaction, and cross-encoder. There are also specialized benchmarks, such as **LoCo** (**Lo**ng-**Co**ntext Benchmark) [152], that focus on evaluating the retrieval capabilities of embedding models on long documents, particularly in law and finance domains, which require

handling thousands of tokens.

2.8.3 Domain-Specific Benchmarks

In specialized domains such as chemistry and biomedicine, evaluation efforts have focused on the end-to-end task performance of fine-tuned encoder models rather than on the intrinsic quality of their embeddings. The **BioCreative** [65] challenge series organized a biochemical text-mining evaluation, including the CHEMDNER tasks (Chemical Disease/Drug Named Entity Recognition) [90] in BioCreative IV/V, which involved automatic recognition of chemical names in PubMed abstracts. Similarly, the **BioNLP Shared Task** [85] has provided benchmarks for extracting complex biomedical events (such as protein-protein interactions and gene/protein regulation) from research articles, representing the first large-scale, community-wide effort in fine-grained biomedical information extraction. In the chemistry domain, the ChEMU labs introduced information extraction challenges on chemical patents [60]. For example, ChEMU 2020 defined tasks on chemical reaction text: (1) chemical named entity recognition (identifying compounds, reagents, etc., and their roles) and (2) event extraction for steps in reaction procedures, resulting in a rich annotated corpora for cheminformatics. In the biomedical domain, a dedicated benchmark called **BLUE** (Biomedical Language Understanding Evaluation) [133] assembled five task types with ten datasets (covering clinical named entity tagging, relation extraction, QA, etc.) to evaluate language models on biomedicine, leading to models such as **BioBERT** [98]. Despite these successes, there remains no dedicated benchmark for assessing chemical text embeddings themselves; existing domain challenges emphasize downstream accuracy on classification, extraction, and QA tasks with fine-tuned models, leaving the intrinsic evaluation of domain-specific embeddings as an open area for research.

2.9 Domain Specific Models in Chemistry

General-purpose language models, such as BERT and GPT, excel on many tasks; however, they struggle when a domain's vocabulary, syntax, and reasoning differ significantly from everyday language. Chemistry is a notable example; its literature is filled with technical jargon, fixed phrase patterns, and compact notations such as SMILES and SELFIES that represent molecular structures as text. To address this gap, researchers have finetuned existing transformer architectures or trained new ones using chemistry-focused data. The result is a set of models whose representations reflect chemical meaning in ways that generic language models cannot. The rest of this section reviews these specialized models, beginning with those fine-tuned on natural-language chemistry sources, then examining models trained directly on

chemical representations, and finally exploring decoder-only LLMs that combine both text and chemical representations for generation and dialogue.

2.9.1 Models Trained on Natural Language Texts

SciBERT is a BERT-based model trained on 1.14 million scientific papers (over 3.1 billion tokens) from Semantic Scholar [10, 5, 89]. Its custom "scivocab" captures scientific terms, making it well-suited to understand chemistry papers and abstracts. Given any sentence or token in scientific text, SciBERT produces contextual embeddings that can be utilized for tasks such as classification or information extraction in the chemistry literature.

MatSciBERT (Materials Science BERT) is fine-tuned on SciBERT with 2.4 million sentences from materials science publications, including research on alloys, glasses, and concrete [57]. By focusing on materials and chemistry vocabulary, it generates embeddings that better reflect domain context, which helps downstream applications such as named-entity recognition of chemical terms or extracting material properties from text.

ChemicalBERT starts from a SciBERT checkpoint and continues training on over 40,000 chemical industry documents (safety data sheets, product information) plus 13,000 Wikipedia chemistry articles [144]. It produces embeddings finely tuned to chemical nomenclature and regulatory language, making it ideal for tasks like chemical document classification, question answering, and compliance analysis in the chemical sector.

2.9.2 Models Trained on Chemical Language Representations

ChemBERTa An encoder model with a RoBERTa architecture, but is trained entirely on chemical language such as SMILES strings (and in some variants, SELFIES) instead of ordinary English [31]. For instance, one ChemBERTa model was pretrained on 10 million PubChem SMILES, learning the syntax of molecular representations. The input is simply a SMILES string, and the output embedding encodes key structural features that prove helpful for downstream tasks, such as toxicity or bioactivity prediction. An improved version, ChemBERTa-2 [4], added a multi-task training objective by predicting molecular properties along with the MLM objective during pretraining, which further enhances its utility as a chemical foundation model.

MolBERT takes a similar path, uses a BERT-style encoder for SMILES, but supplements the standard masked-language modelling objective with chemistry-specific

tasks: it learns to recognize equivalent SMILES (different strings for the same molecule) and to predict molecular descriptors during pretraining [101, 47]. Trained on roughly 4 million ChEMBL [53] and ZINC [72] molecules, MolBERT's multi-task regime yields richer embeddings that better capture chemical properties.

MoLFormer This family, developed by IBM [150], scales this idea to an extreme: pretraining on over 1 billion SMILES from ZINC and PubChem, it uses enhanced rotary position embeddings [172] and efficient attention mechanisms [83, 189] to handle very long SMILES sequences. MoLFormer-XL embeddings have even outperformed some graph-based models on solubility, bioactivity, and other property benchmarks, showing that large-scale text-only training can rival 3D structural approaches [202, 191].

Other models, **SMILES-BERT** [188], **ChemFormer** [73], and similar models, treat chemical formulas as an actual language. Trained on 19 million ZINC SMILES and 100 million sequences, respectively, these transformers output vector representations that are useful in applications from virtual screening and similarity search to clustering in chemical space. By viewing SMILES and SELFIES as a formalized vocabulary, they unlock the power of modern NLP in cheminformatics.

2.9.3 Decoder-Based Models

ChemLLM A 7-billion-parameter, decoder-only model built specifically for chemistry dialogues [206]. It was trained and instruction-tuned on a custom "ChemData" corpus that converts structured chemical knowledge (e.g., formulas, reactions, safety rules) into conversational examples. You can prompt ChemLLM with a question or request that mixes natural language and chemical notation (names, SMILES, equations), and it responds with detailed, chemistry-aware text. It handles tasks such as converting IUPAC names into SMILES, describing molecular structures in plain English, and predicting reaction products. In benchmarks, ChemLLM matches or beats GPT-3.5 and even approaches GPT-4 on core chemistry problems, making it a practical chat-style expert for molecule Q&A, protocol design, or concept explanation.

MolGPT An early, smaller decoder-only transformer (around 6 million parameters) [7] trained on SMILES strings to generate new molecules. Using a left-to-right language modelling objective, it learns SMILES syntax and can auto-complete or sample novel strings from a given scaffold or start token. Most outputs follow chemical grammar, so they represent valid compounds. In generative benchmarks like MOSES [138] and GuacaMol [19], MolGPT outperforms earlier VAE-based generators, showing that

even a modest GPT-style model can drive molecular design by proposing diverse, property-tuned candidates.

ChemGPT A family of GPT-style models, such as ChemGPT-1.2B [50] built on GPT-Neo's 1.2 billion parameter backbone, aimed at free-form molecule generation. Trained on millions of PubChem SMILES, these models take a text prompt (empty, a partial scaffold, or a property tag) and sample complete SMILES strings, effectively imagining new compounds. They excel in virtual screening and lead optimization: by conditioning on desired features, ChemGPT can propose analogs or novel candidates. The flexibility of a decoder-only design makes it easy to guide generation with simple text hints.

Beyond these dedicated models, researchers have fine-tuned general LLMs like GPT-3 for chemistry (e.g., **GPTChem**, a 175 billion-parameter variant) [75] or adapted LLaMA for scientific content (such as **PMC-LLaMA** on PubMed data) [195]. These models support chemistry Q&A, protocol writing, and even code generation for data analysis. However, they rely heavily on prompt engineering or task-specific fine-tuning to handle chemical notation, and in many cases, chemistry-trained LLMs like ChemLLM outperform them on specialized tasks. Decoder-only chemical LLMs thus serve both generative and assistive roles, turning SMILES, equations, or plain-text queries into actionable insights in the lab.

2.10 Architectural Improvements

The transformer architecture [180] uses multi-head self-attention and feed-forward layers to do effective sequence modelling with parallelization. In the standard transformer, each token is mapped to Query (Q), Key (K), and Value (V) vectors, and self-attention is computed as a scaled dot product:

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

where d_k is the dimensionality of keys. This design allows each token to attend to all others; however, it also has quadratic complexity in sequence length for both computation and memory. For a sequence of length N, the QK^T multiplication outputs an $N \times N$ attention matrix, making self-attention $O(N^2)$ in time and space. Such quadratic scaling severely limits the application of transformers to long texts (e.g., full-length chemical patents or literature). For example, extending at transformer context from 512 to 5000 tokens increases attention memory by nearly a hundred times, making training on long domain-specific corpora impractical. Another limitation of

the original transformer is its reliance on absolute positional encodings. The baseline model adds a fixed positional vector to token embeddings (e.g., sinusoidal waves with period 10,000) to inject word-order information. These absolute position encodings do not generalize well beyond the sequence length seen in training. If we feed a longer text than the model was trained on, the positional signals become out-of-distribution and can cause attention weights to explode or behave unpredictably [29]. In other words, absolute position embeddings lack any built-in sense of relative distance beyond the training window. This is especially problematic in domains where documents often span thousands of tokens: the model either cannot capture long-range dependencies or must be fully retrained on longer inputs.

In summary, while the vanilla transformer is a powerful foundation, it faces two major limitations for embedding domain-specific texts like chemical literature: (1) its self-attention mechanism scales quadratically with sequence length, and (2) its absolute positional encodings do not generalize to longer contexts. The following sections review architectural and training innovations that address these issues.

2.10.1 Efficient Attention: FlashAttention

Beyond the quadratic arithmetic cost, the real performance issue lies in memory traffic. Even on modern, fast GPUs, much of the attention computation time can be dominated by memory read/write overhead rather than arithmetic operations. Each attention layer must load the keys and queries, write the large attention matrix to memory, then reload them to multiply by V. Studies have shown that on modern GPUs, memory access is often the true bottleneck, as compute speed outpaces memory speed [39]. Thus, simply reducing theoretical FLOPs (e.g., approximate attention methods) may not result in real speedups if memory traffic remains high. The real challenge is to redesign attention to be more I/O-aware and minimize unnecessary data movement.

FlashAttention is an exact attention algorithm that directly addresses I/O limitations by redesigning the self-attention mechanism to use **tiling** and **fused GPU kernels** to minimize memory reads/writes. GPUs use a memory hierarchy where **SRAM** is a small, extremely fast memory located directly on the processing chip. In contrast, **HBM** (High-Bandwidth Memory) is the much larger, but slower, main memory pool that resides off-chip. The standard attention computes and stores the full $N \times N$ score matrix in off-chip memory (HBM), then reads it back to multiply by the value vectors. FlashAttention breaks the computation into smaller tiles that fit in fast on-chip memory (SRAM). It fuses all the steps (matrix multiply, masking, softmax, and value weighting) into a single GPU kernel, so each query, key, and

value tensor is read only once, and the final output is also written just once. By avoiding the full $N \times N$ write-read cycle, FlashAttention reduces memory traffic and turns attention from memory-bound into compute-bound. End-to-end training is reported to be improved by 15% for BERT-large (512 tokens) and a 3x speedup for GPT-2 (1000 tokens). Because it only requires extra space proportional to N (for a fixed tile size) instead of N^2, FlashAttention enables the training of transformers on much longer inputs within the same hardware budget. On very long documents (4000-16, 000 tokens), FlashAttention not only runs faster but also enables training on tasks such as the 16, 000-token Path-X challenge, which standard attention simply cannot handle. As a result, it has quickly become a standard building block for long-context transformer models. For example, the Nomic family of text embedding models uses FlashAttention to process up to 8192 tokens efficiently.

2.10.2 Positional Encoding and Long-Context

The original transformer adds a fixed sinusoidal vector to each token embedding based on its position index. While this works well for moderate lengths, it causes two main issues for long texts. First, absolute position embeddings treat position i in one sequence as entirely different from position i in another, lacking a notion of relative distance. This means the model has to learn from scratch how to handle shifts in input order. In tasks like chemical text analysis, where the relevant information might be spread across a document (e.g., multiple mentions of a compound over pages), an absolute scheme offers no built-in translational invariance; the model might struggle to relate a concept at position 100 to one at position 1000 unless it saw similar distances during training. Second, absolute position embeddings do not generalize beyond the maximum length seen in training. If a model is trained on sequences up to 512 tokens, feeding it 1024 tokens forces it to guess positional values outside its familiar range. Although sinusoidal functions can mathematically generate values beyond the training range and for any index, in practice, attention scores for untrained positions often explode or oscillate unpredictably. The resulting softmax can collapse to near one-hot or uniform distributions, which reduces the model's ability to reliably handle longer documents.

Rotary Positional Embedding (RoPE) It replaces fixed positional vectors with a rotation applied to each query-key pair, so positions enter the attention computation as relative offsets rather than absolute indices [172]. In RoPE, each query/key vector is split into $d/2$ two-dimensional pairs, and each 2-D pair is rotated by an angle proportional to its token position. Formally, for position m, and a given frequency band in the embedding, RoPE defines a 2-D rotation per pair: $(q_m^{(2i)}, q_m^{(2i+1)})$ is

transformed to

$$\begin{pmatrix} q'_{m,2i} \\ q'_{m,2i+1} \end{pmatrix} = \begin{pmatrix} \cos\theta_{m,i} & -\sin\theta_{m,i} \\ \sin\theta_{m,i} & \cos\theta_{m,i} \end{pmatrix} \begin{pmatrix} q_{m,2i} \\ q_{m,2i+1} \end{pmatrix} \tag{2.10.1}$$

with a similar rotation for keys. This angle is defined based on the token's position and the dimension's index using a fixed sinusoidal frequency. For a token at position m and dimension pair i, the angle is calculated as:

$$\theta_{m,i} = \frac{m}{b^{2i/d}} \tag{2.10.2}$$

where d is the feature dimension of the vector, $i \in [0, 1, \ldots, d/2 - 1]$ is the index for the dimension pair, and b is a large constant base (typically $10,000$). This formulation creates a spectrum of rotational speeds across the dimensions, from low frequencies that capture long-range relative positions to high frequencies that capture fine-grained local ones. Since these rotations depend only on the difference $(m - n)$, the dot product $Q_m \cdot K_n$ naturally encodes relative distance. Unlike absolute encodings, RoPE can be evaluated at arbitrary positions m at inference time; however, models may degrade beyond their training length$L$. In practice, models using RoPE (e.g., the LLaMA family) have shown better stability for moderate extrapolation and often yield smoother distance-aware attention patterns. This is especially valuable in chemistry texts, where key information (for instance, a compound and its properties) may be scattered throughout a lengthy document. However, pushing far beyond the training length can cause the so-called **Neural Tangent Kernel** (NTK) instability [132]: high-frequency rotations make attention weights numerically unstable on unseen lengths. Empirical studies show that once inputs exceed the training cutoff L, RoPE's attention scores either scatter unpredictably or collapse into overly sharp peaks, causing perplexity to explode and attention to effectively break [183]. To address these failures, researchers have developed several long-context strategies:

1. **Position Interpolation (PI):** It tackles RoPE's extrapolation limits by compressing the position numbers of a longer input to fit the model's original range [29]. For example, if a model trained on $L = 2048$ tokens needs to handle $L' = 4096$ tokens, we map positions 0-4095 onto 0-2048 before applying RoPE which will effectively scale positions by $s = L'/L$. This simple tweak prevents the severe attention failures seen when RoPE is used beyond its training length. Although PI slightly reduces resolution (distant tokens may share similar rotation angles), fine-tuning on a small long-text dataset is often done to adapt the model. In practice, PI is very effective: LLaMA models fine-tuned with PI for a $32K$-token context run stably on long texts and show only minor drops on shorter tasks. With approximately 1,000 fine-tuning steps, PI can extend context windows by eight times or more at a modest computational cost.

2. **NTK-Aware Frequency Rescaling:** Another approach is to modify RoPE's rotation frequencies for longer contexts [46]. In RoPE, high-frequency components correspond to rapid phase changes that become problematic at long range. By observing that neural networks learn low-frequency patterns more easily than high-frequency ones, NTK-aware scaling, we scale each RoPE angle θ by:
$$\theta' = \theta * (\frac{L'}{L})^{\alpha} \tag{2.10.3}$$
By choosing $\alpha < 1$, this stretches out the rotations, and high-frequency dimensions are interpolated less or not at all, while low-frequency ones can still cover the extended range. The result is that attention varies more gently at long distances. Empirically, NTK-aware RoPE (with a tuned α) improves perplexity for longer sequences compared to naive extrapolation. It does involve a trade-off: a fixed α that helps at 16K length might slightly hurt at very short lengths (because we altered the geometry of position space). Still, it can enable zero-shot extension; using the model at double or quadruple length without any fine-tuning, with only minimal loss in accuracy at normal lengths.

3. **Dynamic NTK Scaling:** It adjusts the RoPE scaling factor on the fly based on the current sequence length [46]. Up to the original training length L, it keeps the base angle θ unchanged, but beyond L, it smoothly increases scaling toward the NTK-aware factor $\theta' = \theta(\frac{L'}{L})^{\alpha}$, where L' is the extended context and $\alpha < 1$. We can set
$$\theta' = \theta * ((\alpha * \ell/L) - (\alpha - 1)) \tag{2.10.4}$$
so that $\theta' = \theta$ when $\ell = L$ and $\theta' \to \theta(\frac{L'}{L})^{\alpha}$ as $\ell \to L'$. This method preserves exact RoPE performance on short inputs while applying stronger NTK scaling only at extreme lengths. Peng et al. (2023) incorporated Dynamic NTK into their YaRN framework, doubling the usable context window without any fine-tuning and maintaining stable accuracy from short to very long sequences.

4. **YaRN (Yet another RoPE extensioN):** It is a method that combines several of the above ideas into a unified and effective long-context solution [132]. It improves RoPE's scalability by applying a frequency-aware interpolation scheme: low-frequency components of the position embedding are fully interpolated (like in Position Interpolation), high-frequency components are left unchanged to avoid instability, and mid-range frequencies are partially interpolated using a smooth transition. This interpolation is guided by two thresholds, α and β, which control the extent to which each frequency band is scaled based on the

sequence length. Formally we have:

$$s_i = \begin{cases} s & \text{if } \lambda_i < \alpha \text{ (High frequencies)} \\ \frac{\lambda_i - \alpha}{\beta - \alpha}(1-s) + s & \text{if } \alpha \leq \lambda_i \leq \beta \text{ (Mid-range frequencies)} \\ 1 & \text{if } \lambda_i > \beta \text{ (Low frequencies)} \end{cases} \tag{2.10.5}$$

where $s = L'/L$ (extension ratio). Additionally, YaRN modifies the attention mechanism by introducing a temperature hyperparameter into the softmax function. This rescaling stabilizes attention scores over long inputs, helping to maintain low perplexity even as context length increases significantly. At inference time, YaRN can also be combined with dynamic scaling, allowing models to generalize to longer sequences (e.g., 2 times the training length or more) without retraining. YaRN has proved to be highly efficient in practice. LLaMA models adapted with YaRN achieved context lengths up to 128K tokens while maintaining strong performance, using only 10 less training data and 2.5 times fewer training steps than earlier long-context methods.

Positional encoding enhancements, such as RoPE and its subsequent methods, are essential for long-context embedding models. Fixed absolute encodings would collapse on long documents, but RoPE, combined with position interpolation, smoothly maps any longer sequence back into the model's familiar range. For example, the open-source Nomic embedding family [124] applies both interpolation and frequency rescaling to support an 8192-token context window. This enables it to convert full research articles into useful vectors without requiring retraining for longer inputs. By layering in NTK-aware scaling and techniques such as YaRN, modern transformers treat length extension as a gradual adjustment rather than a hard failure, and can keep attention stable even as sequence lengths are pushed to longer lengths. When paired with efficient attention algorithms, these positional-encoding tricks break through the original transformer's context limits, which is especially valuable in chemistry NLP, where key information may span very long documents or complex reaction chains.

2.10.3 Feed-Forward Block Upgrades

Each transformer layer consists of the multi-head attention followed by a feed-forward network (FFN), which is a simple two-layer MLP that uses a ReLU activation function [180, 3]. Formally we have [161]:

$$FFN(x, W_1, W_2, b_1, b_2) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.10.6}$$

And in some variants such as T5 [141] there is no bias:

$$FFN(x, W_1, W_2) = \max(0, xW_1)W_2 \tag{2.10.7}$$

One limitation of this approach is that ReLU is a unilateral activation function, meaning it outputs zero for negative inputs, which might limit the expressiveness of the FFN [128]. Furthermore, the FFN has to learn an element-wise nonlinearity purely via ReLU, which has no trainable parameters or adaptive behaviour. In recent years, researchers have found that adding a learned, input-dependent gate inside the FFN can significantly enhance model quality and training convergence [161].

Gated Linear Units (GLUs) A GLU layer splits the input into two parts: one is passed through a linear transformation ($xV + c$), and the other through a linear and a sigmoid gate (element-wise filter) [40]. Formally:

$$GLU(x, W, V, b, c) = \sigma(xW + b) \otimes (xV + c) \tag{2.10.8}$$

where xV and xW are two affine transforms of the input, σ is sigmoid and $\otimes$ is element-wise product. Essentially, the network can learn to route information selectively: the gate can attenuate or pass through components of xV based on the input. GLU provides a learnable dynamic activation instead of a fixed function like ReLU.

GeGLU and SwiGLU Building on GLUs, variants like GeGLU (which uses GELU [64] for the gate) and SwiGLU (which uses the Swish function [142]) have been proposed [161]. Formally for the entire FFN we have:

$$FFN_{GEGLU}(x, W, V, W_2) = (\text{GELU}(xW) \otimes xV)W_2 \tag{2.10.9}$$
$$FFN_{Swish}(x, W, V, W_2) = (\text{Swish}_1(xW) \otimes xV)W_2 \tag{2.10.10}$$

where $\text{GELU}(x) = x\Phi(x)$ with Φ the standard normal CDF (an equivalent tanh approximation is often used in practice), $\text{Swish}_\beta(x) = x\sigma(\beta x)$ and the bias is removed. Similarly, SwiGLU uses the Swish activation. All these gated designs still use three weight matrices (two to generate the gated inputs and one for the final projection). Still, they can keep the same total number of parameters as the original ReLU network by slightly reducing the hidden size. The key benefit is a **multiplicative interaction** that ReLU alone cannot provide, allowing the FFN to approximate more complex, data-dependent piecewise functions [40]. As a result, many modern language models now use SwiGLU instead of ReLU (for example, Meta's LLaMA [177, 178, 55] and Google's PaLM [33]). Additionally, the first version of Nomic embedding models [124] adopted SwiGLU with FlashAttention, achieving not only higher accuracy but also

a 25% speedup over its GEGLU setup. In summary, replacing the standard ReLU in a feed-forward network with a gated variant (GLU, GeGLU, or SwiGLU) is a straightforward, zero-cost change that yields richer representations, faster convergence, and improved performance across various NLP tasks.

2.10.4 Memory and Training Efficiency

Training state-of-the-art text embedding models, especially those using contrastive learning, can be both compute and memory-intensive. A key challenge is the need for very large batch sizes. Early work in contrastive learning [126, 30] demonstrates that having many in-batch negatives strengthens the training signal, enabling the model to learn more discriminative embeddings. However, simply scaling up batch size quickly hits GPU memory limits. In this section, we will cover two techniques to overcome memory limitations.

GradCache (Gradient Caching) is a technique introduced to scale contrastive learning to very large batch sizes under limited memory [52]. In contrastive learning, having a large batch with many negative examples significantly improves representations. However, naively increasing batch size N also increases memory usage (storing N embeddings, and backpropagating gradients through N items simultaneously). One approach that comes to mind is gradient accumulation, which is used to simulate a large batch by processing small micro-batches, calculating and accumulating gradients for each, and updating the model only at the end, once all of them have been processed. For this to work, the loss calculation for each micro-batch must be identical for all the other micro-batches, which is not the case in InfoNCE loss. Consider a slightly altered variant of the equation 2.4.3, where both the positive and negative cases are merged into a summation in the denominator, for a sample i we have:

$$L_i = -\log\left(\frac{e^{E_{q_i}\cdot E_{p_i}}}{\sum_{j=1}^{N} e^{E_{q_i}\cdot E_{p_j}}}\right) \tag{2.10.11}$$

On the other hand, we know that in a network with two encoder towers (bi-encoder) with shared parameters θ and batch size N we have:

$$\frac{\partial L}{\partial \theta} = \sum_{i=1}^{N} \frac{\partial L}{\partial E_{q_i}} \frac{\partial E_{q_i}}{\partial \theta}, \tag{2.10.12}$$

$$\frac{\partial L}{\partial \theta} = \sum_{j=1}^{N} \frac{\partial L}{\partial E_{p_j}} \frac{\partial E_{p_j}}{\partial \theta}. \tag{2.10.13}$$

Taking the partial derivative of equation 2.10.11 with respect to query (E_{q_i}) and passage (E_{p_j}) embeddings, we have:

$$\frac{\partial L}{\partial \mathbf{E}_{q_i}} = -\frac{1}{N}\left(\mathbf{E}_{p_i} - \sum_{k=1}^{N} \sigma_{ik}\, \mathbf{E}_{p_k}\right), \tag{2.10.14}$$

$$\frac{\partial L}{\partial \mathbf{E}_{p_j}} = -\frac{1}{N}\left(\boldsymbol{\varepsilon}_j - \sum_{i=1}^{N} \sigma_{ij}\, \mathbf{E}_{q_i}\right), \tag{2.10.15}$$

$$\sigma_{ij} = \frac{e^{\,E_{q_i} \cdot E_{p_j}}}{\sum_{k=1}^{N} e^{\,E_{q_i} \cdot E_{p_k}}} \tag{2.10.16}$$

Where σ_{ij} denote the softmax probability of passage p_j among the N candidates for query q_i The ε_j term in equation 2.10.15 ensures that the strong *pull* force is only applied by the true positive pair. It is formally defined as:

$$\boldsymbol{\varepsilon}_j = \begin{cases} \mathbf{E}_{q_j}, & \text{if passage } p_j \text{ is the positive for query } q_j, \\ \mathbf{0}, & \text{otherwise.} \end{cases} \tag{2.10.17}$$

The key idea lies in how the gradients are used to update model parameters. The gradient $\frac{\partial L}{\partial \theta}$ splits into two parts based on chain rule:

1. The **Representation Gradient**, such as $\frac{\partial L}{\partial \mathbf{E}_{q_i}}$ and $\frac{\partial L}{\partial \mathbf{E}_{p_j}}$. These gradients depend only on the final embedding vectors for the entire batch, and can be computed without any knowledge of the encoder's internal graph or activations.

2. The **Encoder Gradient**, such as $\frac{\partial E_{q_i}}{\partial \theta}$ and $\frac{\partial E_{p_j}}{\partial \theta}$. This is the typical back-propagation through the encoder for a single example, which requires the example's computation graph and stored activations.

Because the representation gradient for any one sample relies on all other samples in the batch, we cannot simply **accumulate gradients** in the usual way. GradCache overcomes this by separating the two factors based on the following steps:

1. **Graph-less forward pass:** Run the encoder over the whole batch to get every embedding E_{q_i} and E_{p_j}, but do not build or store any computation graphs. This will result in a set of embeddings required to calculate the representation gradient.

2. **Representation gradient caching:** Using the embeddings from Step 1 to

compute $\frac{\partial L}{\partial \mathbf{E}_{q_i}}$ and $\frac{\partial L}{\partial \mathbf{E}_{p_j}}$ for each sample, and store these gradient vectors in a cache.

3. **Sub-batch backpropagation:** Split the batch into small chunks. For each chunk, perform a normal forward pass, creating the compute graph, then perform the back-propagation [151] using the cached gradients from step 2 for those samples. The resulting parameter gradients ($\frac{\partial L}{\partial \theta}$) are accumulated.

4. **Optimizer Step:** After processing all chunks, the accumulated gradient is used to perform a single optimizer step.

Importantly, this procedure is exact, not an approximation. It results in the same $\frac{\partial L}{\partial \theta}$ as a single forward/backward pass over the whole batch. By never holding the full batch's activations in memory at once, GradCache reduces peak memory use to the size of one micro-batch, enabling effectively unlimited batch sizes (bounded only by compute time and the space needed to store cached embeddings and gradients). GradCache makes it feasible to train state-of-the-art contrastive embedding models with massive batches, without requiring prohibitive GPU memory.

Mixed-Precision Training Using 16-bit formats (Floating-point 16 or Brain Floating-point 16) has become the standard practice for scaling transformer training. By halving the bit-width of most weights and activations relative to FP32, we roughly halve memory use and inter-GPU communication, while taking advantage of specialized hardware units (Tensor Cores on NVIDIA GPUs or matrix units on TPUs [125]) that run 16-bit operations faster than 32-bit. However, standard FP16 suffers from a narrow exponent range (1-bit sign / 5-bit exponent / 10-bit mantissa; normal values $\approx 6.1 \times 10^{-5}$ to 6.5×10^{4}), which can cause gradients to underflow or overflow during back-propagation. BFloat16 (BF16) addresses this by keeping the same 8-bit exponent as FP32 but reducing the mantissa to 7 bits (BF16 layout 1-bit sign/ 8-bit exponent /7-bit mantissa; while FP32 is 1-bit sign/ 8-bit exponent / 23-bit mantissa). This preserves the ability to represent very large and very small values without normal-range saturation, at the cost of modestly lower precision. In practice, we keep critical tensors such as the copy of the model weights and specific accumulation buffers in full 32-bit precision, while switching all other computations to BF16 [113]. Because BF16 does not require manual loss-scaling tricks to avoid numerical issues, it delivers stable training almost out of the box. Empirical studies [81] show that, with some enhancements like stochastic rounding or compensated summation, BF16 training can match FP32 accuracy across a range of tasks. In terms of hardware efficiency, BF16 units can offer better power efficiency and lower latency compared to FP32 units. Modern frameworks automate this via **Automatic Mixed Precision (AMP)**, which runs most operations in BF16 while preserving FP32 where necessary. The result is about

a 1.5-2× reduction in memory footprint overall (since master weights and optimizer state often remain FP32), enabling larger batch sizes or bigger models on the same hardware. Nearly all recent open-source LLMs (such as BLOOM [95]) rely on mixed precision for this reason.

2.11 Tokenizer Domain-Adaptation

Modern transformer models, such as BERT, rely on subword tokenizers to split text into smaller, manageable units. BERT specifically uses a tokenizer called WordPiece [158, 198]. The process of building a WordPiece vocabulary begins by collecting all unique characters in the training dataset. Then, it gradually combines pairs of these subwords into larger tokens. The pairs chosen for merging are those that most significantly improve the likelihood of representing the training data, calculated by:

$$\text{score} = \frac{\text{freq(pair)}}{\text{freq(A)} \times \text{freq(B)}} \tag{2.11.1}$$

This method ensures that tokens represent meaningful and frequent combinations. When applying the tokenizer to new text, WordPiece selects the longest matching subword from its vocabulary, segmenting words piece by piece. This effectively handles rare or unknown words by breaking them down into known, smaller parts, thereby addressing the out-of-vocabulary issue. Other common tokenization strategies include **Byte Pair Encoding (BPE)** [159] and SentencePiece [92]. BPE builds vocabulary by repeatedly merging the most frequently occurring pairs of characters or subwords. SentencePiece uses a probabilistic method (Unigram model) to choose tokens that best represent the data. Both techniques aim to keep the tokens semantically meaningful and maintain manageable overall sequence lengths. Researchers have developed various methods to adapt tokenizers, especially BERT tokenizer, to specific fields or tasks. Below, we provide a brief summary of some approaches from the literature.

SciBERT retrains BERT specifically on a large scientific dataset (about 1.14 million journal articles) to create a specialized vocabulary called **SciVocab**[10]. This was done because BERT's general vocabulary often fragments or misses specialized scientific terms. SciBERT has the same vocabulary size as the original BERT (approximately 30,000 tokens), but shares only about 42% of the tokens. This domain-specific vocabulary slightly improved performance in scientific NLP tasks, confirming that domain-specific vocabularies facilitate a better understanding of scientific language.

exBERT is a modular vocabulary extension with an adapter that enhances a pretrained BERT model by adding an extension module to handle additional vocabulary

[174]. Instead of changing BERT's original weights, exBERT freezes the base model and learns a small set of new parameters: specifically, new embeddings for domain-specific tokens and an adapter module to integrate these new embeddings. The motivation is to adapt efficiently to new domains when computational resources or data are limited. Experiments in the biomedical domain have shown that exBERT can outperform complete retraining approaches in limited-resource situations while maintaining compatibility with BERT's original structure. **AVocaDo** (**A**dapt the **V**ocabulary to Downstream **D**omain) is a fine-tuning method that expands and adjusts the tokenizer specifically for a downstream task domain [68]. In this approach, the model identifies important domain-specific words directly from the task dataset itself (without a separate large pretraining corpus) and adds them to the vocabulary. During fine-tuning, new token embeddings are normalized using an auxiliary mechanism, an extra module that maps input tokens with the expanded vocabulary back to the original tokenizer's embedding space. This essentially allows the model to use new tokens during fine-tuning without significantly altering the pretrained model. AVocaDo showed substantial improvements in domain-specific classification benchmarks (e.g., +8-13 f1 points in scientific and review domains). However, it modestly increases complexity, adding an extra forward step and roughly doubling memory usage during fine-tuning due to handling both tokenizations, but it avoids retraining the base transformer.

CancerBERT extends BERT's existing vocabulary by utilizing reserved [UNUSED] token slots for new domain-specific terms [210]. Designed explicitly for oncology, it incorporated cancer-related terms (such as medical terminology and drug names) into these slots. It continued pretraining on about 3 million clinical notes and pathology reports of cancer patients. By reusing these reserved slots, CancerBERT maintained compatibility with BERT's architecture while integrating new domain knowledge. This targeted vocabulary extension improved accuracy in extracting cancer-specific information from clinical records.

Chapter 3

Chemical Text Embedding Benchmark

Recent developments in language models have opened a new chapter in high-performing information retrieval and content creation, with embedding models playing a central role in improving how data is represented and handled. Standardized benchmarks, such as the Massive Text Embedding Benchmark (MTEB), have made it easier to evaluate embedding models across general domains; however, there remains a lack of tailored tools for specialized fields like chemistry, which pose their own unique challenges. This chapter presents the Chemical Text Embedding Benchmark (ChemTEB), a new benchmark built specifically for the chemical sciences. ChemTEB is designed to address the particular linguistic and semantic challenges of chemical texts and data, providing an extensive set of tasks based on chemistry-specific material. By testing 34 different open-source and proprietary models with ChemTEB, we highlight where current approaches succeed and where they fall short in processing chemical information. Our goal is to provide researchers with a unified, domain-specific evaluation tool that supports the development of more accurate and effective NLP models for chemistry applications. Additionally, ChemTEB provides insight into how general-purpose models perform when applied to a specialized domain. ChemTEB comes with open-source code [1] and data [2], contributing further to its accessibility and utility.

[1] https://doi.org/10.5281/zenodo.16896163
[2] https://huggingface.co/BASF-AI

Authorship and Contributions

Citation. A. Shiraee Kasmaee, M. Khodadad, M. Arshi Saloot, N. Sherck, S. Dokas, H. Mahyar, and S. Samiee. ChemTEB: Chemical text embedding benchmark, an overview of embedding models performance & efficiency on a specific domain. In Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop, volume 262 of Proceedings of Machine Learning Research, pages 512-531. PMLR, 14 Dec 2024

Author (Ali Shiraee Kasmaee)

- Led the project and coordinated collaboration; contributed to task design ideation.
- Developing scripts to gather chemical textual data from different sources.
- Built datasets for classification, pair classification, bitext mining, and retrieval.
- Designed and implemented MTEB-compatible task definitions and dataset loaders, and standardized the datasets to the required format.
- Developed evaluation code; ran and managed all model evaluations, and generated figures and tables from the results.
- Co-wrote the manuscript.

Collaborators

- **Mohammad Khodadad:** Task design ideation; built classification and clustering datasets; generated figures and tables from results; co-wrote the manuscript.
- **Mohammad Arshi Saloot:** Task design ideation; gathered SDS data; built SDS classification datasets.
- **Nick Sherck:** Task design ideation; chemical validation of tasks.
- **Stephen Dokas:** Task design ideation; chemical validation of tasks; idea and validation for ontology-based automated labels in Wikipedia classification tasks.
- **Hamidreza Mahyar and Soheila Samiee:** Supervision, guidance, writing and revision.

Computational Resources

We acknowledge the use of Nvidia V100 instances from the Cedar cluster at Compute Canada.

3.1 Introduction

Recent advances in deep learning and natural language processing have greatly improved the field, making it clear that effective text representation is essential for understanding semantic similarity. This capability plays a significant role in text mining, search, retrieval, and related tasks. Over the past decade, a variety of promising models have been created to address this need. Early examples, such as GloVe [134] and Word2vec [114], introduced word embeddings but did not account for context. Newer models have shifted to using transformer architectures, which make it possible to include context in token embeddings [180]. Each new model brings its own set of architectural features, parameter sizes, maximum context lengths, and pretraining strategies. One of the first to use this approach was BERT, which combined transformer layers and self-supervised learning for embeddings [180, 42]. After BERT, several variations were developed to further boost performance, such as ROBERTA [106], and others focused on adapting to specific scientific fields, like SciBERT [10]. Initially, pooling methods such as averaging the output layer or utilizing the first special token (e.g., BERT's [CLS])[43] were commonly employed to obtain a single embedding for a text; however, these approaches proved less effective in capturing deeper semantic meaning. Sentence-BERT [145] improved on this by using a Siamese bi-encoder architecture and triplet loss [157], resulting in strong performance for semantic representation and making it well-suited for embedding tasks. More recently, models such as E5 [185] and Nomic embed [124] have introduced contrastive learning into the pretraining stage, enabling models to distinguish between similar and dissimilar samples and making them more efficient in practice. The BGE family [200] is notable for its use of a pretraining method inspired by MAE [199], along with contrastive learning, which leverages large batch sizes to further improve embedding quality. M3-embedding [119] emphasizes versatility, supporting multiple functions, granularities, and languages to enhance performance. Additionally, companies such as OpenAI, Cohere, and Amazon have launched their own proprietary embedding models, providing users with even more options. The growth in NLP has also had an impact on various scientific disciplines, including biology, medicine, and physics, allowing researchers to analyze and interpret large volumes of text more accurately and efficiently than before. Embedding models are now essential for handling complex problems across these areas. They work by converting high-dimensional data into compact vector spaces that preserve semantic meaning, which is particularly important for tasks such as mining chemical literature or predicting molecular properties. Retrieval-Augmented Generation (RAG) [100] has emerged as a powerful approach by combining language models with external retrieval systems, providing applications with access to domain-specific knowledge in real-time and making them more effective for tasks that require both deep learning

and up-to-date information. As these techniques become more widely used, there is a growing need for highly efficient embedding models in industry. Although general NLP benchmarks, such as the Massive Text Embedding Benchmark (MTEB) [118], have helped standardize model evaluation across different tasks, they are not well-suited for chemistry-related problems. The subtle language and complex meanings found in chemical literature are often overlooked by models trained only on general data. This highlights the importance of a specialized benchmark for chemistry, where both accuracy and proper context are crucial. Enhanced NLP models could transform various areas of chemistry, including automated literature surveys, synthesis planning, patent analysis, and the advancement of Autonomous Agents in Chemistry [17, 143]. To address this need, we introduce the Chemical Text Embedding Benchmark (ChemTEB), a new benchmark created for the chemical sciences. ChemTEB encompasses a diverse range of tasks, including classifying chemical texts and mining parallel texts between natural language and SMILES representations. Our benchmark is designed to be a strong, domain-specific evaluation tool that supports the development of more accurate and efficient NLP models for chemistry. With its open-source code and datasets, ChemTEB enables easy testing of different models and the addition of new tasks and datasets as needed.

3.2 ChemTEB

In this chapter, we utilize a wide range of datasets to evaluate general text embedding models on various chemistry-related tasks, including classification, Pair Classification, Clustering, Retrieval, and Bitext Mining. The main data sources include PubChem [88], English Wikipedia, BeIR [175], CoconutDB [171], and Safety Data Sheets [136]. Each source brings different and unique types of information, which are essential for a thorough evaluation of NLP models in the chemistry domain. All datasets and tasks were reviewed by domain experts to ensure their relevance and accuracy for chemical applications.

3.2.1 Data Sources

PubChem [87] is an open and freely available database that contains detailed information on chemical molecules, including names, descriptions, chemical formulas, properties, SMILES notations, and 3D molecular structures. In our work, we used PubChem to construct datasets for pair classification and bitext mining tasks. For example, we paired SMILES strings (both isomeric and canonical) with their associated titles and descriptions, and also matched descriptions or paraphrases of entities from different sources to build pair classification tasks.

Wikipedia offers a broad collection of articles covering both general knowledge and scientific topics, including chemistry. For this project, we built the corpus by traversing Wikipedia's category graph rooted at Category:Chemistry, following subcategories breadth-first to depth five, and collecting all main-namespace pages belonging to any visited category (pages were included if any ancestor category descends from **Category:Chemistry**). From this corpus, we constructed datasets for classification and clustering tasks. These datasets vary in difficulty and the number of classes, with all labels assigned by chemistry domain experts.

BeIR (Benchmarking Information Retrieval) [175] is a benchmark suite that includes datasets such as **HotpotQA** [204] and **Natural Questions (NQ)** [94], which are designed for complex question-answering scenarios. In our research, we filtered these datasets for chemistry-related content and used them to build retrieval tasks, where the objective is to find relevant documents or text passages in response to specific queries.

CoconutDB [171] is a specialized database for natural products, providing extensive data on molecular structures and properties. This resource is especially useful for analyzing natural compounds. In this study, we relied on CoconutDB for bitext mining and pair classification, specifically pairing compound formulas with their corresponding SMILES representations.

Safety Data Sheets were obtained from Kaggle [135], which gathered more than $200,000$ SDS documents through web scraping. After collecting the data, we performed rigorous cleaning and annotation to improve its quality and relevance. We created two specific label categories from this data: *Gloves_Required* and *Eyes_Protection_Required*. Since most SDS records specify whether gloves or eye protection are needed, we developed a method that combines large language models (LLMs) and regular expressions to extract this information efficiently. This allowed us to convert the unstructured text into a structured, Boolean format, indicating whether each type of protection is required.

3.2.2 Tasks

We present a variety of benchmarks designed to evaluate different aspects of natural language and chemical data processing. Each benchmark focuses on a specific task. In this section, we provide an overview of the task, the data sources used for its collection, and the evaluation process. These benchmarks offer a comprehensive set of evaluation tasks utilizing diverse datasets tailored to various modelling approaches. Table 3.1 offers a summary of datasets and statistics associated with them.

Classification In this task, each dataset contains a text field paired with labels. We apply a logistic regression classifier on top of the embeddings generated by the model, first training on the training set and then evaluating results on the test set using the F1 score. The primary sources for these datasets are chemistry-focused English **Wikipedia** articles, which are labeled according to different chemistry subfields, and **Safety Data Sheets** (SDS) [136], which are comprehensive documents detailing the properties and hazards of chemicals for user safety and regulatory compliance.

Clustering tasks require the grouping of related text samples into clusters that reflect meaningful relationships in the data, based on their embeddings. As with classification, the clustering datasets come from chemistry-related English **Wikipedia** articles, where sections are grouped into chemistry subfields. We use a mini-batch k-means algorithm with a batch size of 32 for training on these texts. Performance is measured using the V-measure metric [148].

Pair classification involves deciding if two pieces of text are related, assigning a binary label to each pair. In the context of chemistry, this could mean verifying whether both texts refer to the same chemical entity, or whether compound names or descriptions accurately match their SMILES representations. Here, each text pair is embedded using the model, distances are calculated (using metrics such as cosine similarity, Euclidean, Manhattan, or dot product), and the optimal threshold is selected for each. The f1 score is computed for all metrics, and the highest value is used as the main measure of performance. Datasets for pair classification are built from resources such as **PubChem** [88] and **COCONUT** [171].

Bitext Mining aims to find pairs of texts that are translations or semantically equivalent, using semantic similarity searches between query embeddings and corpus embeddings. We draw on data from sources like **PubChem** [88] and **COCONUT** [171], matching SMILES strings for chemical entities with their corresponding titles, descriptions, and formulas. Model performance in this task is measured by the **F1 score**.

Retrieval where the focus is on how well the model can find relevant documents given a query. Datasets in this category consist of queries and a set of documents, along with information mapping which documents are relevant to which queries. All texts are embedded using an embedding model, and documents are retrieved based on cosine similarity between embeddings. For evaluation, we use chemistry-specific subsets of the **Natural Questions** [94] and **HotpotQA** [204] datasets, with **nDCG@10** as the main metric for assessing performance.

Table 3.1: Summary of datasets. This table outlines the datasets used for different tasks, including their Hugging Face dataset names, the original data sources, and the sample size distribution. The distribution is reported using the 5th percentile, median, and 95th percentile values for the number of tokens

Task		HuggingFace Name	Data Source	#Samples	Sequence Lengths (tokens [3])		
					5th Percentile	Median	95th Percentile
Classification	1	WikipediaEasy10Classification	Wikipedia	2105	42	178	612.4
	2	WikipediaEasy5Classification	Wikipedia	1164	43	171.5	547.85
	3	WikipediaMedium5Classification	Wikipedia	617	39	137	563.6
	4	WikipediaMedium2CrystallographyVsChromatographyTitrationpHClassification	Wikipedia	1451	41.5	175	658.5
	5	WikipediaMedium2BioluminescenceVsNeurochemistryClassification	Wikipedia	486	42	158	574.25
	6	WikipediaEZ2Classification	Wikipedia	58921	41	164	590
	7	WikipediaHard2BioluminescenceVsLuminescenceClassification	Wikipedia	410	41	148.5	579.3
	8	WikipediaEasy2GeneExpressionVsMetallurgyClassification	Wikipedia	5741	42	175	630
	9	WikipediaEasy2GreenhouseVsEnantiopureClassification	Wikipedia	1136	34	139.5	513
	10	WikipediaEZ10Classification	Wikipedia	43146	41	165	582
	11	WikipediaHard2SaltsVsSemiconductorMaterialsClassification	Wikipedia	491	38.5	141	447.5
	12	WikipediaEasy2SolidStateVsColloidalClassification	Wikipedia	2216	42	151	532
	13	WikipediaMedium2ComputationalVsSpectroscopistsClassification	Wikipedia	1101	38	155	639
	14	WikipediaHard2IsotopesVsFissionProductsNuclearFissionClassification	Wikipedia	417	43.8	209	706.4
	15	WikipediaEasy2SpecialClassification	Wikipedia	1312	35.55	133	465
	16	SDSGlovesClassification	Safety Data Sheets	8000	498	1071	1871
	17	SDSEyeProtectionClassification	Safety Data Sheets	8000	492	1060	1876
BitextMining	18	CoconutSMILES2FormulaBM	CoconutDB	8000	6	11	150
	19	PubChemSMILESIsoTitleBM	PubChem	14140	4	22	93
	20	PubChemSMILESISoDescBM	PubChem	14140	12	45	134
	21	PubChemSMILESCanonTitleBM	PubChem	30914	3	12	43
	22	PubChemSMILESCanonDescBM	PubChem	30914	8	24	109
Retrieval	23	ChemHotpotQARetrieval	HotpotQA	10275	19	71	183
	24	ChemNQRetrieval	Natural Questions	22960	13	81	231
Clustering	25	WikipediaMedium5Clustering	Wikipedia	617	39	137	563.6
	26	WikipediaEasy10Clustering	Wikipedia	2105	42	178	612.4
PairClassification	27	WikipediaAIParagraphsParaphrasePC	Wikipedia	5408	28	104	354
	28	CoconutSMILES2FormulaPC	CoconutDB	8000	6	11	108
	29	PubChemAISentenceParaphrasePC	PubChem	4096	9	20	59
	30	PubChemSMILESCanonTitlePC	PubChem	4096	4	16	30
	31	PubChemSynonymPC	PubChem	4096	3	8	38
	32	PubChemSMILESCanonDescPC	PubChem	4096	12	23	105
	33	PubChemSMILESIsoDescPC	PubChem	4096	12	48	125
	34	PubChemSMILESIsoTitlePC	PubChem	4096	4	35	70
	35	PubChemWikiParagraphsPC	PubChem	4096	8	66	235

3.2.3 Embedding Models

We evaluate a total of 34 different embedding models using the ChemTEB benchmark, comprising 27 open-source models and 7 proprietary ones. Many of these models have already been discussed in detail in Section 2.4.3. Table 3.2 provides a comparative overview, highlighting various characteristics such as model size, parameter count, maximum context length, and embedding dimensionality.

3.2.4 Ranking Process for Model Performance

Model rankings are determined by their performance on datasets from each task category. First, we calculate the arithmetic mean of the evaluation metrics for each task, which summarizes the model's effectiveness across the individual benchmarks. Next, an overall score for each model is obtained using the Reciprocal Rank Fusion

(RRF) method [36], calculated as:

$$\text{RRF}_{\text{score}}(m) = \sum_{d \in \text{Datasets}} \frac{1}{k + r_d(m)} \tag{3.2.1}$$

where $r_d(m)$ is the position of model m in the ranking for dataset d. The constant k controls how fast the weight drops as the rank gets larger and limits how much any single dataset can add: at most $1/(k{+}1)$ for rank 1. Using $1/(k{+}r)$ instead of $1/r$ reduces the effect of a single top rank and keeps lower ranks non-zero, so the final score rewards models that do well on many datasets. We set $k{=}10$ because each list is short (34 models). A smaller k gives a bit more separation among the top ranks while still favouring consistent results across datasets (rank 1: $1/11 \approx 0.091$; rank 10: $1/20 = 0.050$). For very long lists, many IR setups use $k{\approx}60$; the original RRF study fixed $k{=}60$ and noted the exact value was not critical, so we choose 10 for our shorter lists [36].

3.3 Results

3.3.1 Model Performance

Table 3.3 presents the average results of each embedding model across all task categories, along with their overall $\text{RRF}_{\text{score}}$ (for details on this ranking method, refer to Section 3.2.4). When comparing the **models**, it is clear that no single model dominates in every category. However, general proprietary models generally achieve higher scores than open-source alternatives. The *OpenAI-text embedding 3-large* model delivered the best performance in three out of five task categories, placing it at the top of the overall rankings. Among the open-source models, *Nomic Embedding v1.5* and *Nomic Embedding v1* achieved the highest aggregate scores, coming in just behind the leading proprietary model. For a comprehensive breakdown, Table 3.4 provides the detailed rankings for all evaluated models within each specific task category. These rankings are calculated based on the average performance for every defined task in each category, providing a clear view of model strengths and weaknesses in various scenarios. In addition to accuracy and ranking, processing efficiency is also a key consideration. Table 3.5 reports the processing times for all tasks, measured on an Nvidia V100 GPU instance. This comparison allows for a practical assessment of both performance and computational requirements across models. Together, these tables offer a complete overview of the evaluated models, including their average and task-specific performance, rankings across various benchmarks, and inference speed under consistent hardware conditions.

Table 3.2: Embedding model summary. This table lists each model along with its HuggingFace or proprietary identifier, disk size, parameter count, maximum supported context length, and default embedding dimension. Models are organized into open-source and proprietary categories for straightforward comparison

	Model Name	HuggingFace Model / Model ID (Proprietary)	Model Size	# Parameters	Context length	Embedding size
	Open-Source Models					
1	BERT	google-bert/bert-base-uncased	440 MB	109.4 M	512	768
2	SciBERT	allenai/scibert_scivocab_uncased	442 MB	109.9 M	512	768
3	MatSciBERT	m3rg-iitd/matscibert	440 MB	109.9 M	512	768
4	Chemical BERT	recobo/chemical-bert-uncased	440 MB	109.9 M	512	768
5	Nomic BERT	nomic-ai/nomic-bert-2048	549 MB	136.7 M	2048	768
6	Nomic Embedding v1	nomic-ai/nomic-embed-text-v1	547 MB	136.7 M	8192	768
7	Nomic Embedding v1.5	nomic-ai/nomic-embed-text-v1.5	547 MB	136.7 M	8192	768
8	SBERT - all Mini LM L6.v2	sentence-transformers/all-MiniLM-L6-v2	90.9 MB	22.7 M	512	384
9	SBERT - all Mini LM L12.v2	sentence-transformers/all-MiniLM-L12-v2	133 MB	33.3 M	512	384
10	SBERT - all MPNET-base.v2	sentence-transformers/all-mpnet-base-v2	438 MB	109.4 M	514	768
11	SBERT - multi-qa-mpnet-base.v1	sentence-transformers/multi-qa-mpnet-base-dot-v1	438 MB	109.4 M	512	768
12	E5 - small	intfloat/e5-small	133 MB	33.3 M	512	384
13	E5 - base	intfloat/e5-base	438 MB	109.4 M	512	768
14	E5 - large	intfloat/e5-large	1.34 GB	335.1 M	512	1024
15	E5 - small v2	intfloat/e5-small-v2	133 MB	33.6 M	512	384
16	E5 - base v2	intfloat/e5-base-v2	438 MB	109.4 M	512	768
17	E5 - large v2	intfloat/e5-large-v2	1.34 GB	335.1 M	512	1024
18	E5 - Multilingual small	intfloat/multilingual-e5-small	471 MB	117.6 M	512	384
19	E5 - Multilingual base	intfloat/multilingual-e5-base	1.11 GB	278 M	514	768
20	E5 - Multilingual large	intfloat/multilingual-e5-large	2.24 GB	559.8 M	514	1024
21	BGE - small en	BAAI/bge-small-en	133 MB	33.3 M	512	384
22	BGE - base en	BAAI/bge-base-en	438 MB	109.4 M	512	768
23	BGE - large en	BAAI/bge-large-en	1.34 GB	335.1 M	512	1024
24	BGE - small en v1.5	BAAI/bge-small-en-v1.5	133 MB	33.3 M	512	384
25	BGE - base en v1.5	BAAI/bge-base-en-v1.5	438 MB	109.4 M	512	768
26	BGE - large en v1.5	BAAI/bge-large-en-v1.5	1.34 GB	335.1 M	512	1024
27	BGE - Multilingual - M3	BAAI/bge-m3	2.27 GB	576.7 M	8192	1024
	Proprietary Models					
28	OpenAI - Text embedding 3 - small	text-embedding-3-small	N/A	N/A	8191	1536
29	OpenAI - Text embedding 3 - large	text-embedding-3-large	N/A	N/A	8191	3072
30	OpenAI - Text embedding - Ada - 02	text-embedding-ada-002	N/A	N/A	8191	1536
31	Amazon - Titan Text Embedding v2	amazon.titan-embed-text-v2:0	N/A	N/A	8191	1536
32	Amazon - Titan Embedding G1 Text	amazon.titan-embed-text-v1	N/A	N/A	8191	1536
33	Cohere - Embed English V3	cohere.embed-english-v3	N/A	N/A	512	1024
34	Cohere - Embed Multilingual V3	cohere.embed-multilingual-v3	N/A	N/A	512	1024

From the perspective of different **task categories**, models tended to achieve the highest scores on classification tasks. At the same time, bitext mining presented the greatest challenge, resulting in the lowest overall performance, approaching near-zero. As discussed earlier in Section 3.2.2, bitext mining tasks focus on translating between SMILES representations of chemical compounds and their corresponding titles or descriptions, which introduces additional complexity. The low scores observed in this task are primarily because general-purpose models have not been trained on data types such as SMILES code. Consequently, they are unable to fully understand the semantic

Table 3.3: Overview of model performance. This table offers a detailed comparison of all models across key evaluation tasks, including text classification (reported as macro F1-score), bitext mining (F1-score), text retrieval (nDCG@10), clustering (F1-score), pair classification (maximum F1-score), and the final aggregate (Reciprocal Rank Fusion). For clarity, models are categorized into two groups: open-source and proprietary. The top-performing model in each group is underlined, and the highest-scoring model overall is shown in bold

	Classification (Macro F1)	Bitext Mining (F1)	Retrieval (nDCG@10)	Clustering (V-measure)	Pair Classification (Max F1)	Final Score (RRF)
BERT	0.72 ± 0.04	0.0 ± 0.0	0.28 ± 0.02	0.2 ± 0.03	0.41 ± 0.05	0.122
SciBERT	0.71 ± 0.04	0.0002 ± 0.0	0.2 ± 0.03	0.18 ± 0.02	0.43 ± 0.05	0.122
MatSciBERT	0.7 ± 0.04	0.0003 ± 0.0001	0.11 ± 0.02	0.21 ± 0.03	0.41 ± 0.05	0.122
Chemical BERT	0.68 ± 0.04	0.0003 ± 0.0	0.17 ± 0.01	0.13 ± 0.02	0.42 ± 0.05	0.120
Nomic BERT	0.67 ± 0.04	0.0001 ± 0.0	0.05 ± 0.0	0.22 ± 0.03	0.38 ± 0.04	0.118
Nomic Embedding v1	0.77 ± 0.04	0.0023 ± 0.0002	0.72 ± 0.02	0.46 ± 0.03	**0.55 ± 0.06**	0.285
Nomic Embedding v1.5	0.78 ± 0.04	0.0026 ± 0.0002	0.75 ± 0.02	0.5 ± 0.04	**0.55 ± 0.06**	0.339
SBERT - all Mini LM L6.v2	0.78 ± 0.03	0.0015 ± 0.0002	0.61 ± 0.01	0.36 ± 0.02	0.54 ± 0.06	0.232
SBERT - all Mini LM L12.v2	0.77 ± 0.04	0.0013 ± 0.0001	0.58 ± 0.0	0.34 ± 0.01	0.54 ± 0.06	0.201
SBERT - all MPNET-base.v2	0.78 ± 0.04	0.001 ± 0.0001	0.56 ± 0.0	0.5 ± 0.03	0.54 ± 0.06	0.239
SBERT - multi-qa-mpnet-base.v1	0.74 ± 0.04	0.0009 ± 0.0001	0.56 ± 0.01	0.42 ± 0.04	0.54 ± 0.06	0.185
E5 - small	0.75 ± 0.03	0.0015 ± 0.0001	0.69 ± 0.02	0.12 ± 0.02	0.48 ± 0.05	0.166
E5 - base	0.76 ± 0.04	0.0019 ± 0.0001	0.68 ± 0.01	0.34 ± 0.05	0.49 ± 0.05	0.192
E5 - large	0.77 ± 0.04	0.0029 ± 0.0002	0.7 ± 0.01	0.51 ± 0.04	0.5 ± 0.05	0.290
E5 - small v2	0.76 ± 0.03	0.0012 ± 0.0001	0.69 ± 0.01	0.19 ± 0.03	0.46 ± 0.05	0.165
E5 - base v2	0.76 ± 0.04	0.0016 ± 0.0001	0.68 ± 0.01	0.38 ± 0.05	0.47 ± 0.05	0.178
E5 - large v2	0.76 ± 0.04	0.0022 ± 0.0002	0.73 ± 0.01	0.33 ± 0.05	0.48 ± 0.05	0.214
E5 - Multilingual small	0.74 ± 0.04	0.0018 ± 0.0001	**0.76 ± 0.01**	0.17 ± 0.01	0.47 ± 0.05	0.207
E5 - Multilingual base	0.75 ± 0.04	0.0022 ± 0.0001	0.68 ± 0.0	0.48 ± 0.03	0.47 ± 0.05	0.196
E5 - Multilingual large	0.74 ± 0.04	0.0026 ± 0.0002	0.67 ± 0.0	0.3 ± 0.05	0.48 ± 0.05	0.187
BGE - small en	0.78 ± 0.04	0.0012 ± 0.0001	0.52 ± 0.04	0.27 ± 0.03	0.48 ± 0.05	0.160
BGE - base en	0.77 ± 0.04	0.0019 ± 0.0001	0.59 ± 0.03	0.44 ± 0.05	0.48 ± 0.05	0.186
BGE - large en	0.78 ± 0.04	0.0016 ± 0.0001	0.44 ± 0.06	0.45 ± 0.05	0.49 ± 0.05	0.191
BGE - small en v1.5	0.78 ± 0.03	0.0013 ± 0.0001	0.63 ± 0.03	0.25 ± 0.04	0.48 ± 0.05	0.180
BGE - base en v1.5	0.77 ± 0.04	0.0018 ± 0.0001	0.69 ± 0.02	0.47 ± 0.05	0.49 ± 0.05	0.219
BGE - large en v1.5	0.78 ± 0.04	0.0019 ± 0.0001	0.67 ± 0.02	0.39 ± 0.06	0.5 ± 0.05	0.224
BGE - Multilingual - M3	0.76 ± 0.03	0.0012 ± 0.0002	0.68 ± 0.02	0.45 ± 0.05	0.47 ± 0.06	0.176
OpenAI - Text embedding 3 - small	0.78 ± 0.04	0.0027 ± 0.0003	0.65 ± 0.01	0.49 ± 0.05	0.5 ± 0.05	0.273
OpenAI - Text embedding 3 - large	0.8 ± 0.04	**0.0062 ± 0.0006**	0.71 ± 0.01	**0.6 ± 0.03**	0.53 ± 0.05	**0.384**
OpenAI - Text embedding - Ada - 02	0.78 ± 0.04	0.0035 ± 0.0002	0.66 ± 0.02	0.52 ± 0.04	0.49 ± 0.05	0.279
Amazon - Titan Text Embedding v2	0.77 ± 0.03	0.0024 ± 0.0002	0.62 ± 0.0	0.49 ± 0.04	0.49 ± 0.05	0.224
Amazon - Titan Embedding G1 Text	**0.81 ± 0.03**	0.0032 ± 0.0003	0.6 ± 0.02	0.45 ± 0.06	0.49 ± 0.05	0.285
Cohere - Embed English V3	**0.81 ± 0.03**	0.0012 ± 0.0	0.49 ± 0.04	0.55 ± 0.02	0.53 ± 0.06	0.278
Cohere - Embed Multilingual V3	0.8 ± 0.03	0.0024 ± 0.0001	0.49 ± 0.04	0.53 ± 0.03	0.53 ± 0.06	0.281

Table 3.4: Summary of model ranks

	Classification	Bitext Mining	Retrieval	Clustering	Pair Classification	RRF_Score(k=10)
Nomic BERT	34	33	34	27	34	0.118
Chemical BERT	33	30	32	33	31	0.120
MatSciBERT	32	31	33	28	32	0.122
BERT	30	34	30	29	33	0.122
SciBERT	31	32	31	31	30	0.122
BGE - small en	12	27	26	25	22	0.160
E5 - small v2	23	25	8	30	29	0.165
E5 - small	25	21	9	34	24	0.166
BGE - Multilingual - M3	21	26	10	15	28	0.176
E5 - base v2	22	18	11	19	25	0.178
BGE - small en v1.5	9	23	18	26	20	0.180
SBERT - multi-qa-mpnet-base.v1	28	29	24	17	5	0.185
BGE - base en	16	13	22	16	19	0.186
E5 - Multilingual large	27	7	14	24	23	0.187
BGE - large en	10	19	29	13	17	0.191
E5 - base	20	14	12	22	15	0.192
E5 - Multilingual base	26	11	13	10	27	0.196
SBERT - all Mini LM L12.v2	18	22	23	21	4	0.201
E5 - Multilingual small	29	16	1	32	26	0.207
E5 - large v2	24	12	3	23	21	0.214
BGE - base en v1.5	15	17	7	11	18	0.219
BGE - large en v1.5	6	15	15	18	12	0.224
Amazon - Titan Text Embedding v2	17	8	19	8	14	0.224
SBERT - all Mini LM L6.v2	8	20	20	20	3	0.232
SBERT - all MPNET-base.v2	7	28	25	6	6	0.239
OpenAI - Text embedding 3 - small	5	5	17	9	10	0.273
Cohere - Embed English V3	2	24	28	2	8	0.278
OpenAI - Text embedding - Ada - 02	11	2	16	4	16	0.279
Cohere - Embed Multilingual V3	4	9	27	3	9	0.281
Nomic Embedding v1	19	10	4	12	2	0.285
Amazon - Titan Embedding G1 Text	1	3	21	14	13	0.285
E5 - large	14	4	6	5	11	0.290
Nomic Embedding v1.5	13	6	2	7	1	0.339
OpenAI - Text embedding 3 - large	3	1	5	1	7	0.384

connections between various SMILES strings, which results in weaker performance for this particular task. In comparison, retrieval, clustering, and pair classification tasks were somewhat less challenging, ranking second, third, and fourth in terms of both difficulty and overall model performance.

The models analyzed in this benchmark often share architectural and training similarities. To better understand how these shared characteristics influence performance in each task category, we organized the models into eight families: (i) BERT Family, (ii) Nomic embedding family, (iii) SBERT family, (iv) E5 family, (v) BGE family, (vi) OpenAI family, (vii) Amazon family, and (viii) Cohere family. To visualize and compare the performance of each model family across all datasets and task types, we used Kernel Density Estimation (KDE), as shown in Figure 3.1. KDE is a non-parametric technique for estimating the probability density function of a variable, without assuming an underlying distribution for the data. It works by smoothing individual data points using kernels, often Gaussian, placed at each point.

Table 3.5: Average time (seconds) to run a benchmark for each task in each category on an Nvidia V100 32GB GPU instance.

	Classification	Bitext Mining	Retrieval	Clustering	Pair Classification
BERT	13.78 ± 3.47	23.24 ± 1.92	56.6 ± 0.69	5.92 ± 0.57	6.94 ± 0.84
SciBERT	11.37 ± 2.57	22.42 ± 1.8	54.19 ± 0.63	5.83 ± 0.54	4.76 ± 0.39
MatSciBERT	11.05 ± 2.56	22.29 ± 1.74	54.37 ± 0.61	5.81 ± 0.54	4.92 ± 0.42
Chemical BERT	11.47 ± 2.59	22.25 ± 1.75	54.73 ± 0.57	5.85 ± 0.54	4.92 ± 0.42
Nomic BERT	15.14 ± 3.62	29.26 ± 2.24	76.75 ± 0.91	8.16 ± 0.78	6.97 ± 0.64
Nomic Embedding v1	23.13 ± 5.03	31.05 ± 2.4	79.45 ± 0.88	12.01 ± 1.24	5.91 ± 0.41
Nomic Embedding v1.5	22.82 ± 4.93	28.39 ± 2.12	79.66 ± 0.91	12.09 ± 1.24	5.23 ± 0.35
SBERT - all Mini LM L6.v2	2.36 ± 0.52	9.01 ± 0.89	12.83 ± 0.3	1.13 ± 0.11	1.73 ± 0.14
SBERT - all Mini LM L12.v2	2.82 ± 0.57	11.73 ± 1.08	16.26 ± 0.39	1.17 ± 0.1	1.99 ± 0.09
SBERT - all MPNET-base.v2	11.36 ± 2.73	24.49 ± 1.87	61.51 ± 0.76	6.26 ± 0.6	4.43 ± 0.29
SBERT - multi-qa-mpnet-base.v1	13.29 ± 3.09	24.06 ± 1.92	62.06 ± 0.67	7.13 ± 0.69	4.42 ± 0.3
E5 - small	4.98 ± 1.06	12.55 ± 1.11	21.79 ± 0.29	2.36 ± 0.22	2.54 ± 0.2
E5 - base	11.24 ± 2.67	23.93 ± 1.97	58.84 ± 0.75	6.28 ± 0.6	5.28 ± 0.46
E5 - large	37.37 ± 9.5	62.4 ± 4.78	191.41 ± 2.06	20.46 ± 1.99	14.83 ± 1.43
E5 - small v2	5.34 ± 1.08	12.63 ± 1.12	22.15 ± 0.25	2.45 ± 0.23	2.39 ± 0.21
E5 - base v2	11.21 ± 2.66	24.27 ± 2.02	59.45 ± 0.73	6.34 ± 0.6	4.83 ± 0.49
E5 - large v2	36.87 ± 9.28	64.27 ± 4.96	193.9 ± 2.14	20.54 ± 1.97	14.49 ± 1.47
E5 - Multilingual small	5.2 ± 1.11	11.96 ± 1.06	21.68 ± 0.28	2.37 ± 0.23	2.29 ± 0.2
E5 - Multilingual base	12.51 ± 2.99	23.96 ± 1.97	62.13 ± 0.65	6.82 ± 0.67	4.74 ± 0.48
E5 - Multilingual large	40.26 ± 10.31	60.97 ± 4.51	209.69 ± 0.52	22.01 ± 2.18	13.8 ± 1.43
BGE - small en	5.23 ± 1.05	12.46 ± 1.1	21.64 ± 0.29	2.32 ± 0.22	2.82 ± 0.19
BGE - base en	11.14 ± 2.64	23.99 ± 1.98	58.64 ± 0.72	6.29 ± 0.6	5.32 ± 0.48
BGE - large en	37.04 ± 9.33	62.27 ± 4.79	191.56 ± 2.06	20.44 ± 1.97	14.89 ± 1.44
BGE - small en v1.5	5.28 ± 1.05	12.37 ± 1.07	21.83 ± 0.25	2.39 ± 0.23	2.68 ± 0.19
BGE - base en v1.5	11.14 ± 2.63	23.82 ± 1.99	59.08 ± 0.8	6.27 ± 0.59	5.27 ± 0.46
BGE - large en v1.5	36.57 ± 9.12	62.24 ± 4.8	191.63 ± 2.14	20.41 ± 1.97	14.85 ± 1.43
BGE - Multilingual - M3	1139.9 ± 251.82	707.86 ± 48.87	3031.81 ± 22.43	640.67 ± 75.54	31.82 ± 8.61
OpenAI - Text embedding 3 - small	37.17 ± 6.89	372.97 ± 36.14	518.72 ± 12.57	27.74 ± 2.46	63.49 ± 2.91
OpenAI - Text embedding 3 - large	62.18 ± 11.8	730.16 ± 70.34	1006.01 ± 27.68	49.39 ± 4.65	123.33 ± 5.89
OpenAI - Text embedding - Ada - 02	35.57 ± 6.77	372.55 ± 36.18	518.73 ± 12.83	30.77 ± 1.88	64.41 ± 2.94
Amazon - Titan Text Embedding v2	128.01 ± 35.05	1178.06 ± 99.02	1595.24 ± 34.49	84.65 ± 7.8	244.12 ± 3.41
Amazon - Titan Embedding G1 Text	142.23 ± 37.78	1174.83 ± 97.29	1627.31 ± 40.39	89.03 ± 8.38	243.45 ± 3.53
Cohere - Embed English V3	21.21 ± 5.64	83.08 ± 5.98	134.29 ± 2.48	13.27 ± 1.25	16.65 ± 0.89
Cohere - Embed Multilingual V3	22.32 ± 6.07	80.27 ± 5.86	138.74 ± 2.51	14.08 ± 1.3	18.07 ± 1.29

By summing these kernels, KDE provides a continuous estimate of the underlying distribution. The smoothness of the KDE curve is controlled by the **bandwidth parameter**: a smaller bandwidth reveals finer details, while a larger bandwidth produces a smoother, more general distribution. This approach is particularly useful for visualizing the spread and concentration of model performances in different families, making it easier to interpret and compare their effectiveness across various tasks.

3.3.2 Model Efficiency

The models assessed in this benchmark differ in terms of architecture, training data volume, model size, computational speed, and overall effectiveness, among other key attributes. Depending on the application, a particular model may be preferable; however, a comprehensive comparison across several characteristics is necessary to

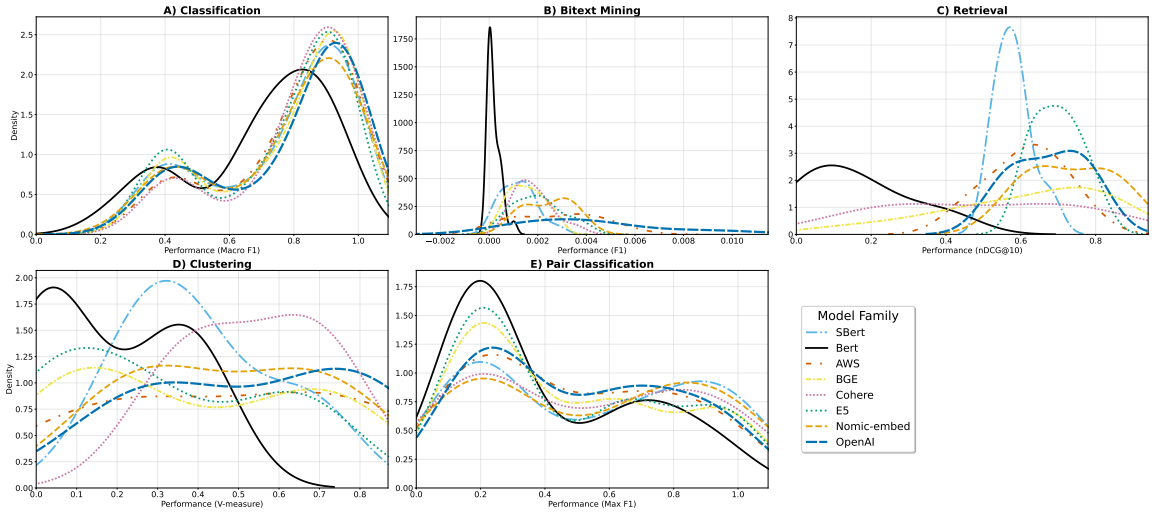

Figure 3.1: Distribution plots are shown for all five task categories. In these KDE visualizations, the x-axis displays the range of predicted values, reflecting the performance spread for each task category and model family, while the y-axis shows the estimated probability density. Each model family is represented by a distinct coloured line, allowing for straightforward comparison of how their performance values are distributed.

make an informed selection. To support this, Figure 3.2 presents a visual comparison of each model's speed (measured on the pair classification task), model size, embedding dimension, and RRF score. Note that for proprietary models, information about model size was not provided. This visualization illustrates the diverse range of model performances, highlighting each model's strengths and limitations. A clear pattern emerges: models with slower inference speeds are typically larger, produce higher-dimensional embeddings, and tend to achieve better overall performance. For instance, *OpenAI - Text Embedding 3 - Large* scored the highest on the RRF metric but was among the slowest models. In contrast, *SBERT - All Mini LM L6.v2* was notable for being both the smallest and fastest, though its performance lagged. BERT-based models stood out for their lower RRF scores and relatively slower inference times, marking a distinct separation from other model types. Among open-source models, *Nomic Embedding v1.5* offered an effective compromise between speed and strong results.

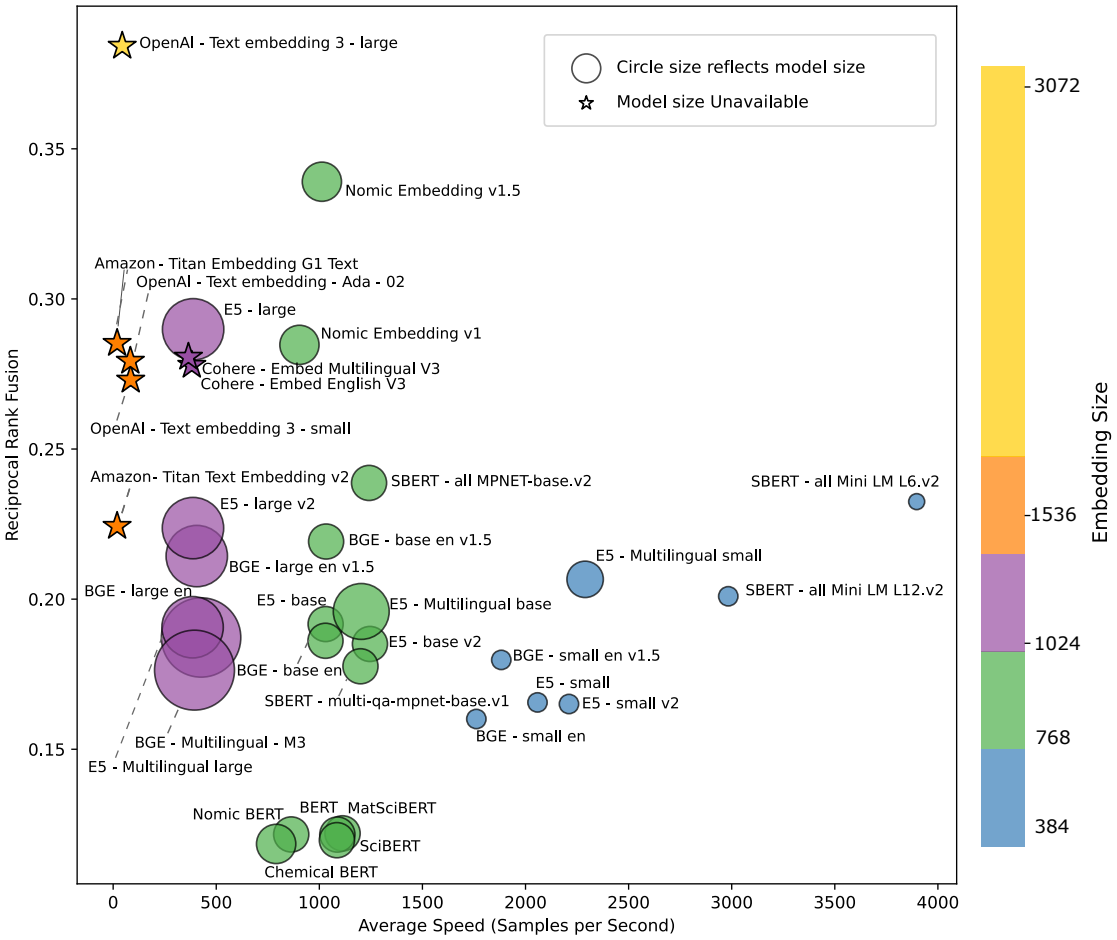

Figure 3.2: Overview of model efficiency. The efficiency of all evaluated models is depicted using (i) circles for open-source models, where the circle's size reflects the number of parameters, and (ii) stars for proprietary models. Model color indicates the embedding size. The x-axis represents average inference speed (in embedded samples per second), computed across seven pair classification tasks (tasks 29 to 35 in Table 3.1) using a V100 GPU.

3.3.3 Domain Adaptation

To our knowledge, the only models that have been explicitly tailored for the chemical domain are MatSciBERT [57] and ChemicalBERT (from Recobo[4]). SciBERT [10], while broadly trained on scientific literature, is also more suitable for chemistry-related tasks than general-purpose models. Within the BERT model family, these

[4] recobo/chemical-bert-uncased

domain-specialized variants surpassed BERT-base on **bitext mining**, which is the most complex task involving a certain level of SMILES code understanding. However, apart from SciBERT's strong results in pair classification, no consistent and substantial gains were observed for the other tasks.

On the other hand, when looking beyond the BERT family, these domain-adapted models tended to underperform in most task categories. This is further supported by their consistently low RRF scores in the overall ranking (see Table 3.4 for the complete task-by-task comparison). One possible reason for this is that these models are based on the original BERT architecture, which relies solely on Masked Language Modeling (MLM) for pretraining. The evaluation suggests that improvements, such as contrastive learning objectives and post-BERT architectural changes, play a greater role in enhancing performance for specific domains than simply adapting earlier architectures to new domains. These results suggest that rather than depending on older, domain-adapted models, future progress in the field will come from designing domain-specific models using more recent and efficient architectures and training paradigms.

Figure 3.3 compares how different models perform across ChemTEB and MTEB benchmarks, grouped by task type. To facilitate direct comparison, the same metrics used on the MTEB leaderboard were employed. In pair classification tasks, which are specifically designed for chemical language, there is a noticeable drop in the average performance for ChemTEB tasks relative to MTEB. This indicates that the models evaluated generally lack domain-specific expertise. When it comes to clustering, the ChemTEB benchmark stands out for offering datasets that better distinguish model performance, as seen in the wider spread of scores. For retrieval, results also show greater variability in ChemTEB, likely due to the more specialized chemical context, though many models still perform better on ChemTEB retrieval tasks than on those in MTEB. Classification tasks in ChemTEB exhibit higher average model performance with less variability in results, suggesting that these may be more straightforward than their MTEB counterparts. This may be explained by the use of general Wikipedia articles in ChemTEB's classification datasets. Overall, these findings demonstrate the significant impact of domain adaptation on model effectiveness and reinforce the need for tailored chemistry-focused evaluation strategies.

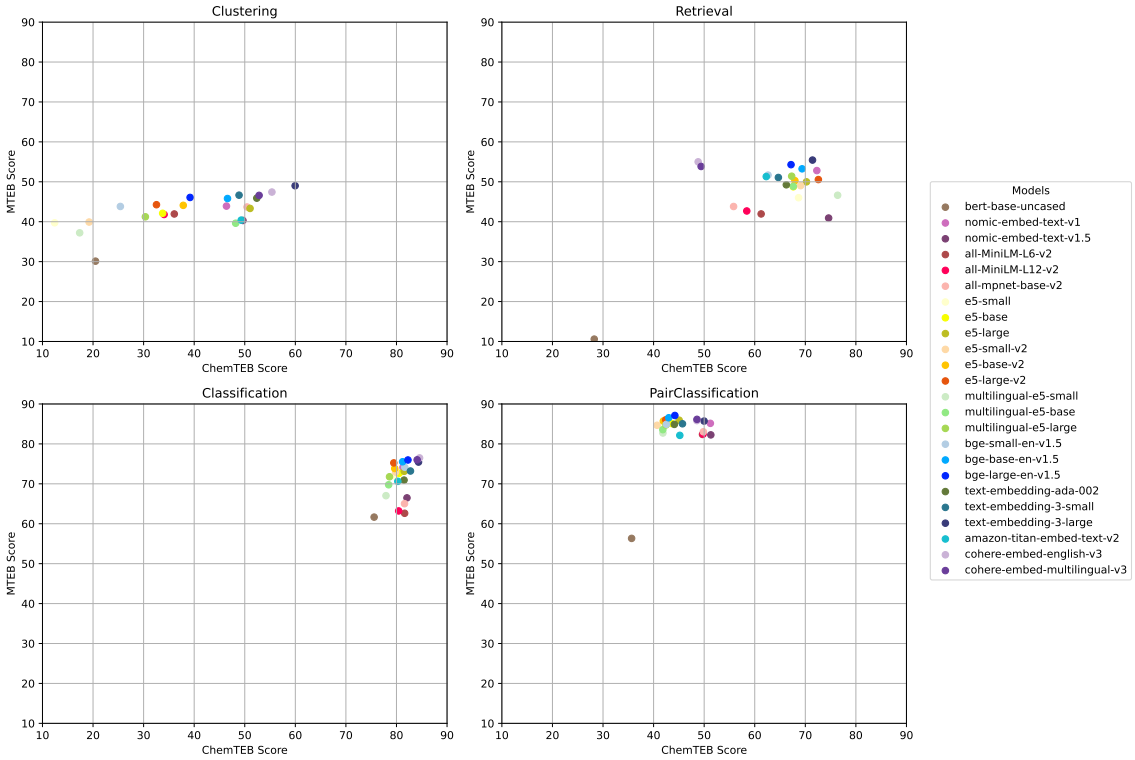

Figure 3.3: Performance comparison of models on ChemTEB and MTEB benchmarks across a range of tasks. Each marker represents a model included both in this benchmark and on the MTEB leaderboard at the time of evaluation. The plot illustrates differences in task difficulty and highlights how domain relevance influences results.

3.3.4 Correlation between models' performances and tasks

Figures 3.4 and 3.5 show correlation matrices for both datasets and models, where color intensity indicates the degree of correlation. In Figure 3.4, we see that datasets within classification, bitext mining, and retrieval tasks generally display positive correlations, except for the SDS datasets in the classification category, which do not follow this trend. For clustering and pair classification tasks, the pattern is less clear, and in pair classification, some datasets are even negatively correlated.

3.4 Conclusion

This chapter addresses an important gap in the evaluation of text embedding models by introducing ChemTEB, an open-source, purpose-built benchmark specifically tailored

to the demands of chemical texts and data. ChemTEB provides a standardized and extensible framework for evaluating general open and proprietary embedding models in chemistry, allowing researchers to compare model performance across a diverse set of tasks systematically. While ChemTEB offers a robust starting point for benchmarking, it is also designed to be adaptable and expandable as new, more specialized tasks and datasets emerge. For specialized or emerging use cases, additional task development and dataset curation may be necessary to ensure that evaluation remains aligned with the unique demands of each application. Its model-agnostic framework means that it can easily be extended to assess any model or include new data, making it a valuable resource in the open-source benchmarking ecosystem. The findings of this study make clear the urgent need for robust, domain-adapted models to better represent and retrieve chemistry-specific information. In addition to providing a practical tool for the research community, this work highlights the need for ongoing progress in designing and optimizing models for specialized scientific domains.

In contrast, Figure 3.5 highlights a clear separation of models into two main groups: the first group includes all BERT-based models, and the second consists of all other evaluated models. The main distinction between these groups lies in the use of contrastive learning after pretraining; a step absent in the BERT-based models, which also tend to perform the worst across nearly all task categories. This finding underscores the impact of contrastive learning on performance. Within the second group, some model families are more closely related in terms of their results, likely due to similar architectures or training strategies. For example, the Nomic embedding family is most closely correlated with the SBERT family, while Cohere models show the strongest correlation with SBERT, followed by the Amazon family.

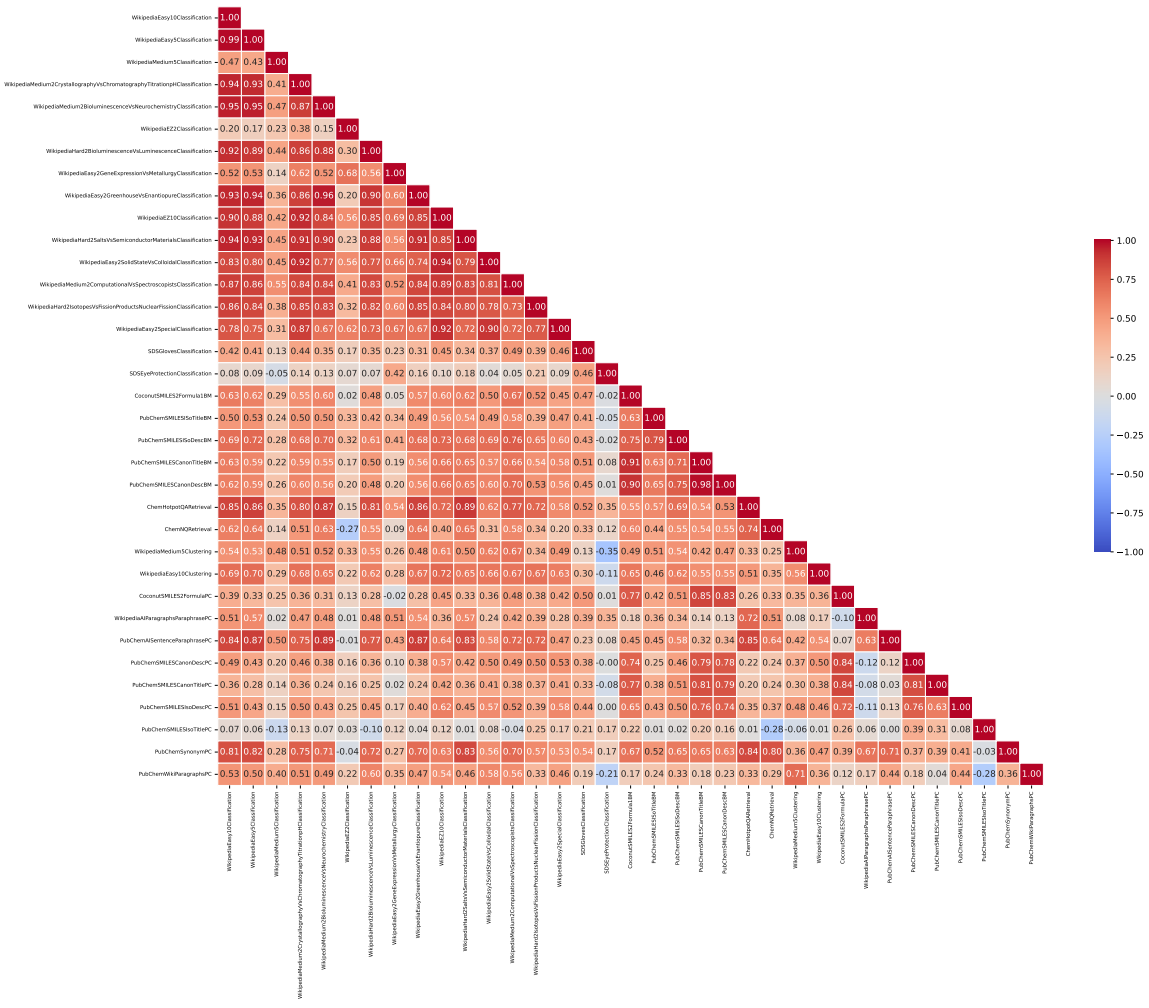

Figure 3.4: Correlation matrix between datasets. Each row and column represents a different dataset evaluated in the ChemTEB benchmark, with the cell values and color indicating how strongly model performance aligns between each dataset pair

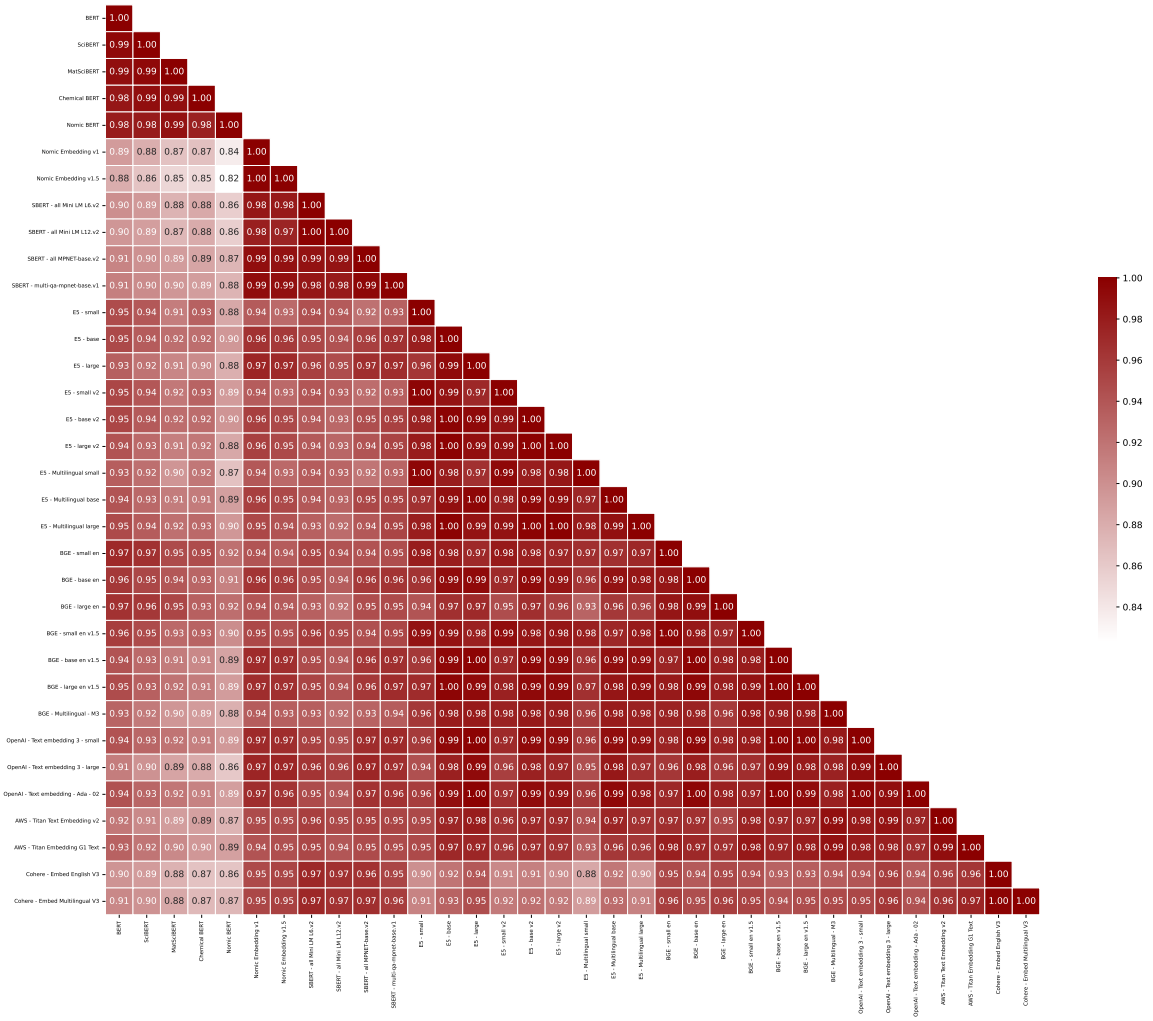

Figure 3.5: Correlation matrix between models. Each row and column corresponds to a specific model tested in ChemTEB. The cell values and colors represent the correlation of performance across all datasets for every pair of models

Chapter 4

Chemical Text Embedding Models

Retrieval-Augmented Generation (RAG) systems in chemistry require highly accurate and relevant retrieval of chemical literature. However, general-purpose text embedding models often struggle to accurately represent complex chemical terms, resulting in lower retrieval performance. To date, no embedding models have been specifically designed for chemical literature retrieval, resulting in a significant performance gap in this area. To address this, this chapter introduces **ChEmbed**, a family of domain-adapted text embedding models that are fine-tuned on chemistry-focused text collected from **PubChem**, **Semantic Scholar**, and **ChemRxiv**. For model training, we generate about 1.7 million high-quality query-passage pairs using large language models to create synthetic queries. To better handle chemical terminology, we also extend the tokenizer by adding 900 new chemistry-specific tokens to available unused slots, which helps reduce the fragmentation of chemical entities, such as IUPAC names. ChEmbed also supports a 2048-token context length, allowing for the efficient retrieval of longer passages without splitting them into smaller parts. In evaluations using the newly developed ChemRxiv Retrieval benchmark, ChEmbed outperforms leading general embedding models, improving nDCG@10 from 0.82 to 0.91. Overall, ChEmbed provides a practical, lightweight, and reproducible embedding solution that enhances retrieval in chemical RAG pipelines. All code and training scripts are openly available [1].

[1] https://doi.org/10.5281/zenodo.16897929

Authorship and Contributions

Citation. A. Shiraee Kasmaee, M. Khodadad, M. Astaraki, M. A. Saloot, N. Sherck, H. Mahyar, and S. Samiee. Chembed: Enhancing chemical literature search through domain-specific text embeddings. arXiv preprint arXiv:2508.01643, 2025.

Author (Ali Shiraee Kasmaee)

- Led the project and collaborations; contributed to ideation and design.
- Developed scripts to gather chemical data from multiple sources.
- Designed and developed a synthetic data generation pipeline with LLMs.
- Designed and ran training experiments; developed the evaluation pipeline, carried out model evaluations, and generated figures and tables.
- Maintained the repository and integrated the tokenizer adaptation module.
- Co-authored the manuscript.

Collaborators

- **Mohammad Khodadad:** Ideation and consultation; trained the tokenizer; collaborated on integrating the tokenizer adaptation module.
- **Mahdi Astaraki:** Ideation and consultation; prompt design and quality control for synthetic query generation; literature review and writing.
- **Mohammad Arshi Saloot:** Ideation on tokenizer adaptation; writing and revision.
- **Nick Sherck:** Performed chemical validation across the pipeline; conducted quality control on the token set; provided consultation on tokenizer training and adaptation; reviewed the manuscript.
- **Hamidreza Mahyar & Soheila Samiee:** Supervision, guidance, and revision.

Computational Resources

Training was conducted on the Narval cluster (Compute Canada) with 4 NVIDIA A100 GPUs. Evaluation used an AWS instance with an A10 GPU provided by BASF Canada.

4.1 Introduction

Retrieval-Augmented Generation (RAG) enables large language models to use external knowledge sources during inference, which improves factual accuracy, particularly when the subject matter differs substantially from what the model encountered during pretraining. Chemistry is a perfect example of this difference: its specialized language, formulas, and reaction contexts are rarely included in general-purpose datasets, leading standard LLMs to hallucinate information or misunderstand essential concepts [131, 1, 56, 206]. Since the quality of a RAG pipeline depends mainly on the effectiveness of its retriever (search component), having strong, domain-specific text embeddings is crucial [124]. Despite rapid progress in universal embedding models, a consistent performance gap remains in chemistry. Current scientific models have trouble with detailed chemical vocabulary, and two main challenges slow progress: (i) **data scarcity**, as contrastive learning requires curated query-passage pairs that are difficult and expensive to collect in specialized areas (although recent studies suggest synthetic data generation as a possible solution [2, 14, 77]); and (ii) **benchmark mismatch**, since most public retrieval benchmarks focus on encyclopedic text and do not reflect the complexity of primary chemical literature [164]. In addition, generic sub-word tokenizers often split chemical names and molecular descriptors in ways that harm the semantic integrity of the text before it even reaches the encoder [13]. This chapter addresses these issues by introducing **ChEmbed**, a text embedding model specialized for chemistry, fine-tuned on 1.7 million synthetic query-passage pairs created from chemical articles and resources. ChEmbed also uses a tokenizer tailored for chemical terms and is evaluated using a new retrieval benchmark constructed from actual chemical literature. The main contributions of this chapter are:

1. **Synthetic data for domain adaptation:** We show that generating large amounts of synthetic query-passage data with LLMs is an effective way to train retrieval models in chemistry.

2. **Domain-adaptive tokenizer:** We apply a lightweight method for adding chemistry-specific vocabulary to an existing WordPiece tokenizer [158, 198] without retraining the full tokenizer or model.

3. **Literature-driven benchmark:** We release a retrieval evaluation task that is built from chemical literature, rather than relying on general encyclopedic sources.

4. **ChEmbed models:** Our best model, which is publicly available, achieves a 9% absolute improvement in nDCG@10 over the general-purpose base model when tested on the new chemistry-specific benchmark.

Table 4.1: Summary of chemistry-related datasets used for training and evaluation

Idx	Dataset	Description	# Samples	Usage	LLM used for query synthesis
1	PubChem compounds	Title, IUPAC, SMILES, synonyms	2,087,164	Tokenizer	N/A
2	PubChem descriptions	Compound descriptions	393,321	Training	`gpt-4o-mini`
3	S2ORC chemistry	Filtered chemistry paragraphs	1,187,726	Training	`gpt-4.1-nano`
4	ChemRxiv paragraphs	Extracted ChemRxiv paragraphs (CC-BY)	139,057	Training	`o3-mini`
5	ChemRxiv paragraphs	Extracted ChemRxiv paragraphs (CC-BY-NC)	69,457	Evaluation	`Claude Sonnet 3.7`
6	ChemRxiv metadata	Title and abstract ChemRxiv preprints	30,378	Training	N/A

These developments help reduce the domain gap for RAG systems in chemistry, supporting the creation of more dependable AI tools to advance chemical research and discovery.

4.2 Dataset Construction

Training bi-encoder text embedding models effectively usually depends on having structured data in the form of (query, passage) pairs or triplets (such as query, positive passage, and negatives), especially when using contrastive learning objectives [102, 200, 185, 124]. However, in specialized fields like chemistry, such structured datasets are often hard to find and are not readily available. To train an embedding model tailored to chemistry, we begin by collecting paragraphs from scientific literature that focus on chemistry. After assembling these domain-specific passages, we prompt a Large Language Model (LLM) to generate related queries for each paragraph in a way that mimics a user's query in a real information retrieval scenario. This approach allows us to build the necessary paired data for contrastive training in the chemistry domain. The datasets prepared and used in this work are listed in Table 4.1.

4.2.1 Data Sources and Preprocessing

We collected chemistry-related paragraphs from the following main sources:

1. **PubChem** is a large chemical database containing details for over 100 million compounds, including names, SMILES, IUPAC names, synonyms, molecular formulas, and descriptions. Using its programmatic interface (PUG-View) [86], we retrieved information for around 2 million compounds, many of which have missing descriptions. After preprocessing, which removed entries that were not real descriptions (for example, cross-references like "see also ...") and texts shorter than five words, we obtained about 393,321 usable descriptions. To increase the dataset size, we used paraphrasing. For descriptions with at least two sentences, an LLM generated two short questions about different aspects of the description and produced one paraphrase; we paired one question with

the original text and the other with the paraphrase, which raised the number of training samples to 480,227.

2. **S2ORC** is a massive dataset comprising over 81 million academic papers, with 8.1 million containing structured full text [107, 170]. This corpus encompasses a wide range of disciplines and features rich metadata, detailed citation information, and comprehensive full-text annotations. For our work, we selected the subset of papers available under the public domain or Creative Commons BY licenses, totalling 6.2 million papers. We filtered these for the chemistry subject, split the relevant 118,000 documents into paragraphs, and then removed conclusion sections, table or figure captions, and segments that were too short or lacked information. After preprocessing, we were left with approximately 1.18 million high-quality paragraphs.

3. **ChemRxiv** is an open-access preprint server dedicated to chemistry and related areas, allowing researchers to share their early findings. We used its public API to gather metadata and PDFs from approximately 30,000 chemistry manuscripts. Texts were processed and split into paragraphs using GROBID, a tool for extracting structured content from scientific papers. To ensure quality, we employed a similar approach to that used in the S2ORC dataset [107, 170], removing paragraphs with an average unigram log probability below -20 or with fewer than 50 words. After completing these steps, we obtained approximately 209,000 high-quality paragraphs.

4.2.2 Synthetic Query Generation via LLMs

To produce high-quality (query, passage) pairs for contrastive training, we used Large Language Models (LLMs) to generate synthetic queries based on chemistry paragraphs. We aimed to closely simulate real-world information retrieval scenarios, such as when a user submits a specific chemistry-related query to find a relevant passage containing the answer. For this purpose, we crafted LLM prompts that directed the model to create a single, clear, and meaningful chemistry question that could be answered using the provided paragraph. We iteratively designed and tested the prompts, trying zero-shot and few-shot variants on small samples. A short few-shot template (3-4 chemistry examples) worked best; we pared the examples down to the most essential ones to maximize answerability and specificity. The data-gathering scripts and the exact prompt templates are available in our Zenodo record.[2] The instructions specifically prohibited simple yes/no questions and references to the paragraph itself (such as "according to this paragraph"). For generating synthetic queries, we used a

[2] https://doi.org/10.5281/zenodo.16895962

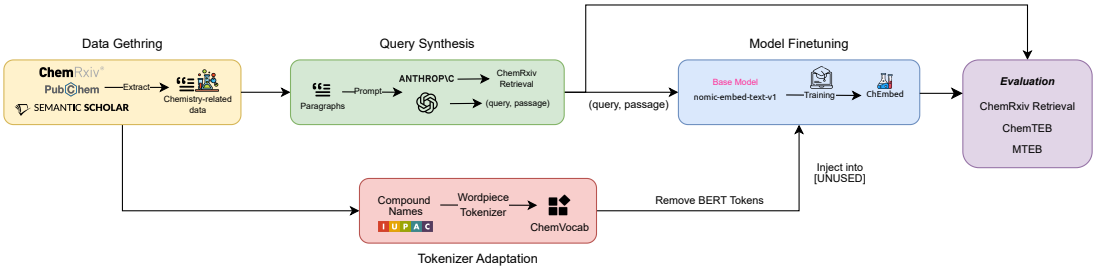

Figure 4.1: Overview of the ChEmbed workflow. Chemistry-related paragraphs are collected from multiple sources and matched with synthetic queries generated by prompting large language models (LLMs). Around 69,000 paragraphs are set aside as an evaluation set, and separate LLMs are used for generating queries for training and evaluation. Meanwhile, a ChemVocab tokenizer is built from 2 million IUPAC names, with 900 unique tokens added into the unused slots of the `bert-base-uncased` tokenizer. The base model (nomic-embed-text-v1) is then fine-tuned using these query-passage pairs, resulting in the final ChEmbed model series.

set of LLMs from the OpenAI platform (`o3-mini`, `gpt-4.1-nano`, and `gpt-4o-mini`), selecting models that strike a balance between scale, cost, and data complexity. For the retrieval evaluation set, we relied on a separate model and provider, `Antrhopic Claude Sonnet 3.7 Thinking`, which was applied to a held-out set of ChemRxiv paragraphs that were not included in training. Following this approach, we retained only those query-passage pairs that fully met our criteria. The LLMs declined to generate queries for around 29,000 paragraphs, thus filtering them out of the dataset. After manually reviewing a sample of these rejected cases, we found that the LLM consistently skipped paragraphs that were either not relevant, too short, or lacked meaningful scientific content, such as funding acknowledgements, broad conclusions, or short, uninformative sections.

4.3 Model Architecture and Domain Adaptation

A primary challenge in developing domain-specific embedding models is selecting a base architecture that offers strong performance and facilitates easy adaptation. In this work, we chose the Nomic embedding family [124], which is among the most effective and lightweight open-source text embedding models, supporting long contexts of up to 8192 tokens. Unlike many closed or partially open alternatives, Nomic models provide complete transparency with access to intermediate weights and pretraining data, which helps ensure reproducibility and enables in-depth analysis. In terms of architecture, Nomic is based on BERT but features essential upgrades such as rotary

positional embeddings, FlashAttention for efficient handling of longer sequences, and SwiGLU activations [161], all of which contribute to better accuracy and scalability. Notably, in the previous chapter, we saw that Nomic was the highest-performing open-source model on chemistry-specific tasks in ChemTEB [164]. Because of these factors, Nomic models are a strong choice for adapting to chemistry-focused tasks. An overview of the data and model pipeline is shown in Figure 4.1.

4.3.1 Domain Adaptation Strategies

To adapt the Nomic models for chemistry retrieval tasks, we investigated multiple fine-tuning approaches using both supervised and unsupervised objectives. We worked with two key variants, both initially trained by the Nomic team: `nomic-embed-text-v1-unsupervised`, which was trained on 235 million unsupervised text pairs with a maximum learning rate of 2×10^{-4}, and `nomic-embed-text-v1`, which received further fine-tuning on 1.6 million supervised hard-negative triplets using a lower learning rate of 2×10^{-5}. For model training, we tried two data formats: (1) using query-passage pairs directly, and (2) constructing triplets consisting of a query, a positive document, and negative samples with hard-negative mining. Both methods can use the same contrastive InfoNCE loss [126]:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{\exp\left(s(q_i, d_i^+)/\tau\right)}{\exp\left(s(q_i, d_i^+)/\tau\right) + \sum_{d^- \in \mathcal{N}(q_i)} \exp\left(s(q_i, d^-)/\tau\right)} \tag{4.3.1}$$

where, $s(q, d)$ is the cosine similarity scaled by a temperature parameter τ, d_i^+ is the correct passage for query q_i, and $\mathcal{N}(q_i)$ refers to the negative samples for that query. In the "pairs" setup, all other passages in the batch are used as in-batch negatives; in the "triplets" setup, we use a fixed set of H pre-selected negatives per query. We followed the Nomic protocol and paired seven negatives per query, since Nussbaum et al. (2024) [124] reported that adding more did not improve performance. We evaluated three schemes (7 hard, 7 random, and a mix of 3 hard + 4 random negatives). Our results showed that fine-tuning with in-batch negatives alone performed better than the triplet-based approach on our synthetic chemistry dataset, likely because our "hard" negatives were generated by the model and not labeled by humans, so they did not have much supervision signal compared to pairs. Most hyperparameters were kept consistent with the original Nomic configuration, with the main changes being a different learning rate of 2×10^{-5}, a linear warmup over 5% of total steps, and a maximum context length of 2048 tokens for training. All models were trained using 4× NVIDIA A100 40GB GPUs with GradCache [52] and mixed-precision training [113], which enabled us to use a large total batch size of $16,384$ for efficient contrastive

Term	**ChemVocab (ours)**	`bert-base-uncased`
`chlorofluorocarbon`	`chloro/fluoro/carbon`	`ch/lor/of/lu/oro/carbon`
`acetaminophen`	`acet/amino/phen`	`ace/tam/ino/phen`
`secbutylamine`	`sec/butyl/amine`	`secbutylamine`
`nitrophenylphosphate`	`nitrophenyl/phosphate`	`ni/tro/ph/en/yl/ph/os/phate`

Table 4.2: Example tokenizations with our ChemVocab vs. the general `bert-base-uncased` tokenizer. Tokens are WordPiece units; continuation markers (e.g., "##") are omitted for readability.

learning. A summary of the full data and model pipeline is illustrated in Figure 4.1

4.3.2 Tokenizer Adaptation

One ongoing challenge when adapting language models to chemistry is the inefficient tokenization of complex terms, such as IUPAC names. Similar to other open-source text embedding models based on BERT [43], the initial version of the Nomic embedding models uses the default `bert-base-uncased` tokenizer, which contains $30,522$ tokens in its vocabulary. Importantly, this tokenizer has exactly 994 `[UNUSED]` tokens, which are placeholders not mapped to any actual word or subword in the original vocabulary. If fewer than 994 new tokens need to be added, these unused slots can be reassigned for domain-specific vocabulary, offering a straightforward way to expand the tokenizer. This method maintains the original tokenizer's structure and compatibility while directly enhancing its ability to handle specialized chemical language. To build a tokenizer better suited for chemistry, we trained a WordPiece tokenizer [198] on 2,083,502 unique IUPAC names from PubChem compounds. Any tokens that were already part of the `bert-base-uncased` vocabulary were excluded, and we selected the top 900 remaining chemistry-specific tokens to add into the available `[UNUSED]` slots, resulting in our adapted tokenizer, **ChemVocab**. The embeddings for these new tokens were initialized from a normal distribution with a mean of 0 and a standard deviation of 0.2. This approach allowed the model to better encode detailed chemical terminology without significant changes to the overall architecture. As a concrete illustration of the domain gap, Table **??** provides four representative examples showing how standard tokenization fragments chemical terms. At the same time, the adapted tokenizer (ChemVocab) results in more compact and meaningful representations.

4.4 Experiments & Results

Experimental Setup We fine-tuned `nomic-embed-text-v1` specifically for the chemical retrieval task. This fine-tuning process was done both with and without adding new tokens to the tokenizer. For the cases where new tokens were added, we explored three different methods to optimize token addition, which are described in more detail in Section 4.4.2. We then evaluated effectiveness using retrieval tasks from three benchmark suites: ChemRxiv Retrieval, MTEB (English v2), and ChemTEB. The **ChemRxiv Retrieval** benchmark is a new task developed for this work, having a collection of 69,457 paragraphs from ChemRxiv, paired with 5,000 synthetic queries generated by an LLM (Anthropic's Claude 3.7 Sonnet) other than those used for training to reduce any potential bias from data generation. **MTEB (English v2)** [118] is a well-known retrieval benchmark that covers 41 English tasks across seven areas, including classification, clustering, and retrieval. However, it is a general-purpose benchmark and does not focus specifically on chemistry-related tasks. **ChemTEB** [164] is a chemistry-specific benchmark built on top of MTEB, containing two retrieval tasks based mainly on encyclopedic sources, and is used to assess how well embedding models generalize to chemical content.

4.4.1 Domain-Specific Retrieval Performance

Quantitative Performance Comparison Table 4.3 shows the results of various models evaluated on the ChemRxiv Retrieval task. All ChEmbed variants consistently outperform not only the original `nomic-embed-text-v1` baseline but also every other notable open-source and proprietary embedding model tested so far. Interestingly, even the basic **ChEmbed$_{\text{vanilla}}$** model, which relies on the unmodified BERT tokenizer without any domain-specific tokenizer enhancements, achieves an *nDCG@10* of **0.884**. The best result is achieved by the progressive tokenizer adaptation strategy (**ChEmbed$_{\text{prog}}$**), which reaches an *nDCG@10* of **0.911**, representing a total gain of 9.0 percentage points over the base model. Every ChEmbed variant, including the weakest, also surpasses the top-performing open-source alternative model, `Qwen3-Embedding-8B`, which achieves an *nDCG@10* of only 0.865, despite having about 55× as many parameters.

Speed-Performance Trade-off Analysis Figure 4.2 illustrates how different embedding models balance speed and performance on the ChemRxiv retrieval task. Models are positioned by their retrieval speed (samples processed per second) along the horizontal axis and their *nDCG@10* scores on the vertical axis. The size of each marker represents the number of model parameters, while colour indicates the model's maximum embedding dimension. When tested on an NVIDIA A10

Table 4.3: Performance of embedding models on the **ChemRxiv Retrieval** benchmark. "N/A" indicates that the model's publisher did not provide the parameter count. The highest score for each metric is highlighted in bold.

Model Name	Emb. size	#Params (M)	MAP@10	MRR@10	NDCG@10
Open-Source Models					
chemical-bert-uncased	768	109.9	0.096	0.096	0.110
matscibert	768	109.9	0.117	0.117	0.137
nomic-bert-2048	768	136.7	0.019	0.019	0.025
ModernBERT-base	768	149.0	0.048	0.048	0.056
ModernBERT-large	1024	394.8	0.049	0.049	0.058
scibert_scivocab_uncased	768	109.9	0.101	0.101	0.119
bert-base-uncased	768	109.5	0.099	0.099	0.117
all-MiniLM-L12-v2	384	33.4	0.556	0.556	0.603
all-MiniLM-L6-v2	384	22.7	0.626	0.626	0.674
all-mpnet-base-v2	768	109.5	0.618	0.618	0.670
multi-qa-mpnet-base-dot-v1	768	109.5	0.697	0.697	0.741
e5-small	384	33.0	0.682	0.682	0.726
e5-base	768	109.0	0.728	0.728	0.770
e5-large	1024	335.0	0.765	0.765	0.806
e5-small-v2	384	33.0	0.715	0.715	0.756
e5-base-v2	768	109.0	0.717	0.718	0.761
e5-large-v2	1024	335.0	0.781	0.781	0.821
multilingual-e5-small	384	118.0	0.736	0.736	0.778
multilingual-e5-base	768	278.0	0.758	0.757	0.798
multilingual-e5-large	1024	560.0	0.753	0.753	0.794
gte-small	384	33.4	0.687	0.687	0.735
gte-base	768	109.5	0.700	0.700	0.748
gte-large	1024	335.1	0.722	0.722	0.768
gte-multilingual-base	1024	305.0	0.712	0.712	0.761
bge-small-en	384	33.4	0.589	0.589	0.638
bge-base-en	768	109.5	0.604	0.604	0.655
bge-large-en	1024	335.1	0.584	0.584	0.635
bge-small-en-v1.5	384	33.4	0.672	0.672	0.719
bge-base-en-v1.5	768	109.5	0.698	0.698	0.744
bge-large-en-v1.5	1024	335.1	0.717	0.717	0.763
bge-m3	4096	568.0	0.758	0.758	0.798
nomic-embed-text-v1-unsupervised	768	136.7	0.773	0.774	0.814
nomic-embed-text-v1	768	136.7	0.782	0.782	0.821
nomic-embed-text-v1.5	768	137.0	0.739	0.739	0.783
nomic-embed-text-v2-moe	768	475.3	0.781	0.781	0.820
modernbert-embed-base	768	149.0	0.772	0.772	0.813
stella_en_1.5B_v5	8960	1540.0	0.760	0.760	0.802
jina-embeddings-v3	1024	572.0	0.715	0.715	0.760
Qwen3-Embedding-0.6B†	1024	596.0	0.779	0.779	0.819
Qwen3-Embedding-4B†	2560	4020.0	0.826	0.826	0.861
Qwen3-Embedding-8B†	4096	7570.0	0.831	0.831	0.865
$\text{ChEmbed}_{\text{vanilla}}$	768	136.7	0.878	0.878	0.902
$\text{ChEmbed}_{\text{progressive}}$	768	136.7	**0.889**	**0.889**	**0.911**
Proprietary Models					
text-embedding-ada-002	1536	N/A	0.725	0.726	0.770
text-embedding-3-small	1536	N/A	0.721	0.721	0.767
text-embedding-3-large	3072	N/A	0.728	0.729	0.775
amazon-titan-embed-text-v1	1536	N/A	0.611	0.611	0.665
amazon-titan-embed-text-v2	1024	N/A	0.763	0.763	0.805
cohere-embed-english-v3	1024	N/A	0.737	0.737	0.781
cohere-embed-multilingual-v3	1024	N/A	0.747	0.747	0.789

† Loaded with 8-bit quantisation to fit into GPU VRAM; no major performance drop observed.

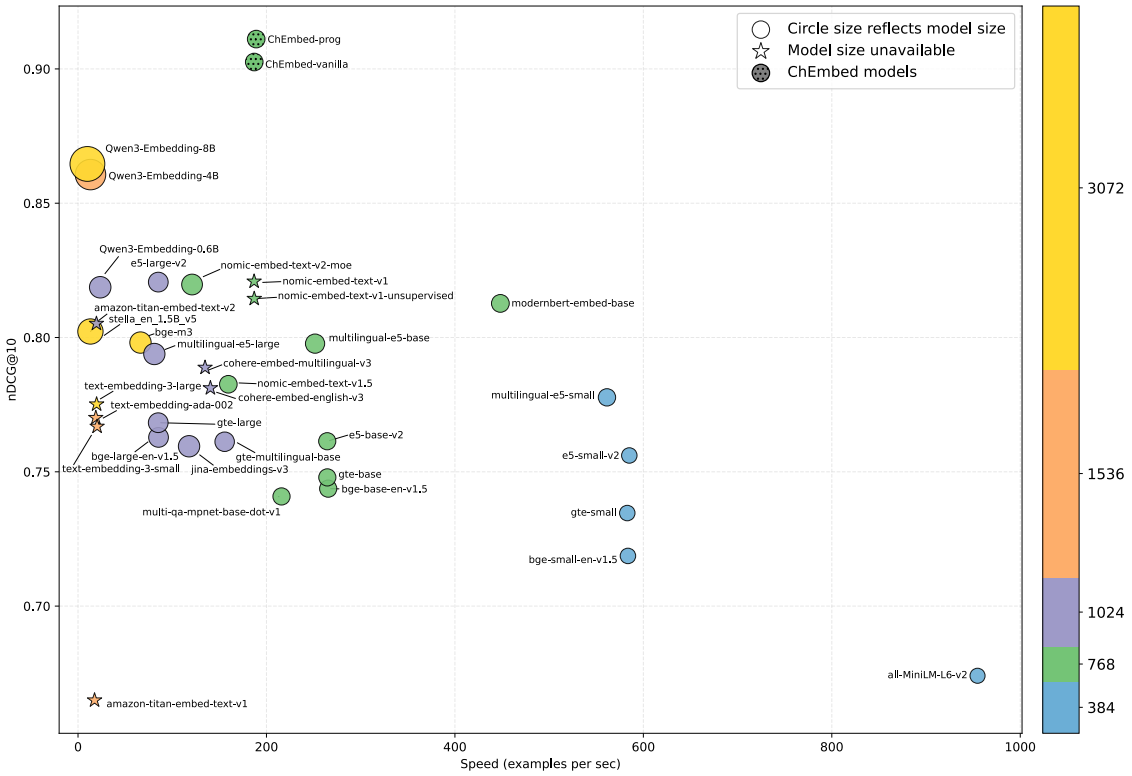

Figure 4.2: Efficiency of models on ChemRxiv Retrieval: horizontal axis shows speed, vertical axis shows $nDCG@10$, and the size of each circle is proportional to the number of parameters

24GB GPU, the ChEmbed models achieve an average throughput of 189 samples per second. In contrast, the top open-source competitors, `Qwen3-Embedding-4B` and `Qwen3-Embedding-8B`, manage only 13 and 10 samples per second, respectively. Despite being over 30 and 55 times larger than ChEmbed, the Qwen3 models not only process data more slowly but also perform worse in retrieval accuracy, where their best variant (`Qwen3-Embedding-8B`) reaches just 0.865 *nDCG@10*, whereas ChEmbed attains 0.911. This highlights that ChEmbed delivers state-of-the-art performance for chemical literature retrieval while being significantly faster and more resource-efficient than the larger alternatives.

4.4.2 Tokenizer Adaptation Analysis

We fine-tuned `nomic-embed-text-v1` under four different tokenizer setups, keeping the remainder of the training process unchanged:

1. **ChEmbed$_{\text{vanilla}}$** uses the standard `bert-base-uncased` tokenizer with no additional vocabulary.

2. **ChEmbed$_{\text{full}}$** uses the ChemVocab tokenizer, allowing all embedding parameters to be updated during training.

3. **ChEmbed$_{\text{plug}}$** also employs the ChemVocab tokenizer, but only the embeddings for the new chemistry-specific tokens are updated in the word embedding layer, while the original BERT token embeddings remain frozen; all other parts of the model are trainable.

4. **ChEmbed$_{\text{prog}}$** follows a progressive schedule, beginning with a phase where only the new token embeddings are trained (with all other weights frozen), followed by unfreezing and training the entire network for additional epochs.

Table 4.4 summarizes the improvements in retrieval quality resulting from each tokenizer configuration compared to the baseline. The results clearly demonstrate that each step of vocabulary adaptation yields noticeable benefits for chemistry-specific retrieval. The progressive adaptation strategy, where training begins by updating only the new token embeddings before unfreezing the entire network, yields the highest overall performance. This suggests that a carefully staged integration of domain-specific vocabulary can maximize the benefits of adaptation while preserving robustness to general-domain language.

Table 4.4: Effect of various tokenizer adaptation methods on retrieval performance for ChemRxiv

Variant	$nDCG@10$	Δ vs. baseline
`nomic-embed-text-v1` (baseline)	0.821	–
ChEmbed$_{\text{vanilla}}$	0.902	+8.1 %
ChEmbed$_{\text{full}}$	0.895	+7.4 %
ChEmbed$_{\text{plug}}$	0.903	+8.2 %
ChEmbed$_{\text{progressive}}$	**0.911**	+9.0 %

4.4.3 Evaluation on General Benchmarks

The primary training objective of our embedding models was fine-tuning on retrieval-oriented query-passage pairs, which comprised approximately 99% of the training set. Additionally, the fine-tuning data consisted almost entirely of specialized chemical literature. In contrast, publicly available benchmarks such as **MTEB**

Table 4.5: Performance comparison on ChemTEB and MTEB across shared non-retrieval task types. Metrics reported are accuracy for Classification, V-measure for Clustering, and average precision (AP) for Pair Classification. **Mean (Task)** represents the average score across all tasks, while **Mean (Task Type)** is calculated by averaging within each task type, followed by the mean of these category averages

	ChemTEB					MTEB				
Model	Cls	Clust	Pair	Mean (T)	Mean (T-type)	Cls	Clust	Pair	Mean (T)	Mean (T-type)
nomic-embed-text-v1-unsupervised	0.824	0.567	**0.635**	0.763	**0.675**	0.754	0.444	0.836	0.637	0.678
nomic-embed-text-v1	**0.837**	0.570	0.594	**0.764**	0.667	**0.774**	**0.466**	**0.853**	**0.657**	**0.698**
$\texttt{ChEmbed}_{\texttt{vanilla}}$	0.795	0.526	0.594	0.731	0.638	0.766	0.427	0.843	0.635	0.678
$\texttt{ChEmbed}_{\texttt{full}}$	0.813	0.546	0.547	0.735	0.635	0.773	0.436	0.849	0.643	0.686
$\texttt{ChEmbed}_{\texttt{plug}}$	0.796	**0.583**	0.564	0.730	0.648	0.767	0.425	0.842	0.635	0.678
$\texttt{ChEmbed}_{\texttt{prog}}$	0.801	0.490	0.566	0.726	0.619	0.769	0.426	0.845	0.637	0.680

English v2 and **ChemTEB** were created to assess text embedding models on a broader range of tasks and across more general domains. As a result, these broader benchmarks may not capture real improvements in retrieval performance for specialized fields and might even give an inaccurate picture of the model's true capabilities. This makes it essential to carefully consider two main sources of mismatch between the training and evaluation benchmarks: **task mismatch** and **domain mismatch**.

Task Adaptation To investigate task mismatch, we assessed how our models perform on non-retrieval tasks that are present in both ChemTEB and MTEB. Specifically, we included Classification, Clustering, and Pair Classification, reporting both the average score across all tasks and the average within each task category (and then across categories). As anticipated, the general-purpose embedding model, `nomic-embed-text-v1`, consistently outperformed our retrieval-specialized models, achieving mean scores of **0.764** on ChemTEB and **0.657** on MTEB across all tasks. Among the ChEmbed models, only $\texttt{ChEmbed}_{\texttt{plug}}$ distinguishes itself, obtaining the highest Clustering score on ChemTEB (0.583) across all evaluated models. For the remaining non-retrieval tasks, either the baseline model or its unsupervised version outperforms the ChEmbed variants. These results underscore that optimizing a model solely for retrieval does not automatically enhance its performance on other task types; therefore, evaluations should focus on the intended real-world use case rather than relying exclusively on general benchmarks.

Domain Adaptation To better understand the impact of domain mismatch, we compared how the models performed on retrieval tasks across ChemRxiv Retrieval, ChemTEB Retrieval, and MTEB Retrieval. For a fair comparison, we summarized key dataset properties for each benchmark, including the average number of queries and corpus size. The ChemRxiv Retrieval benchmark is highly specialized and

Table 4.6: Retrieval results (nDCG@10, higher is better) with corresponding dataset statistics. Benchmarks are presented from most domain-specific to most general

			Dataset statistics			nDCG@10		
Dataset	Domain-Specific	Encyclopedic	#Tasks	Avg Queries	Avg Corpus	`nomic-embed-text-v1`	$\texttt{ChEmbed}_{\texttt{vanilla}}$	$\texttt{ChEmbed}_{\texttt{prog}}$
ChemRxiv Retrieval	✓	✗	1	5000	69,457	0.821	0.902	**0.911**
ChemTEB Retrieval	✗	✓	2	116	16,501	0.763	0.706	0.718
MTEB Retrieval	✗	✓	10	1482	109,645	0.544	0.458	0.462

closely matches our training domain, featuring 5,000 queries and a collection of 69,000 documents. In contrast, ChemTEB Retrieval, although focused on chemistry, draws its tasks from general encyclopedic sources such as HotpotQA [204] and Natural Questions [94]. Specifically, ChemHotpotQARetrieval contains only 206 queries with 10,069 documents, whereas ChemNQRetrieval includes just 27 queries over 22,933 documents, with approximately 117 queries and 16,501 documents per retrieval task on average. Similarly, MTEB Retrieval covers a much broader, general domain, with an average of 1,482 queries and a corpus size of 109,645 documents per task. These significant differences in the number of queries and corpus size highlight ChemTEB's limited relevance as an evaluation benchmark for specialized chemical retrieval scenarios. This limitation is further illustrated by our analysis of model checkpoints in Figure 4.3. Here, we track the retrieval performance of $\texttt{ChEmbed}_{\texttt{vanilla}}$ model snapshots saved after each epoch of fine-tuning across training epochs. The results show that as training progresses, performance on ChemRxiv Retrieval steadily improves, with several peaks and fluctuations, while performance on ChemTEB Retrieval declines over time. This clear divergence suggests that the queries and passages in ChemTEB differ significantly from those found in the specialized chemical literature used for training, resulting in a domain gap and lower evaluation reliability on ChemTEB as the model becomes more specialized.

Taken together, our analysis highlights significant shortcomings in existing benchmarks for accurately evaluating embedding models fine-tuned for retrieval tasks in specific scientific areas. Results on non-retrieval tasks indicate that evaluation should closely align with the training objective, as focusing solely on retrieval can compromise effectiveness on tasks unrelated to retrieval. Likewise, the trends observed in retrieval performance and training indicate that strong domain alignment between the training set and the evaluation benchmark is essential. Based on these findings, we believe that effective evaluation of chemistry retrieval embedding models requires alignment in both the retrieval task and the domain of literature, underscoring the need for a new, dedicated retrieval benchmark.

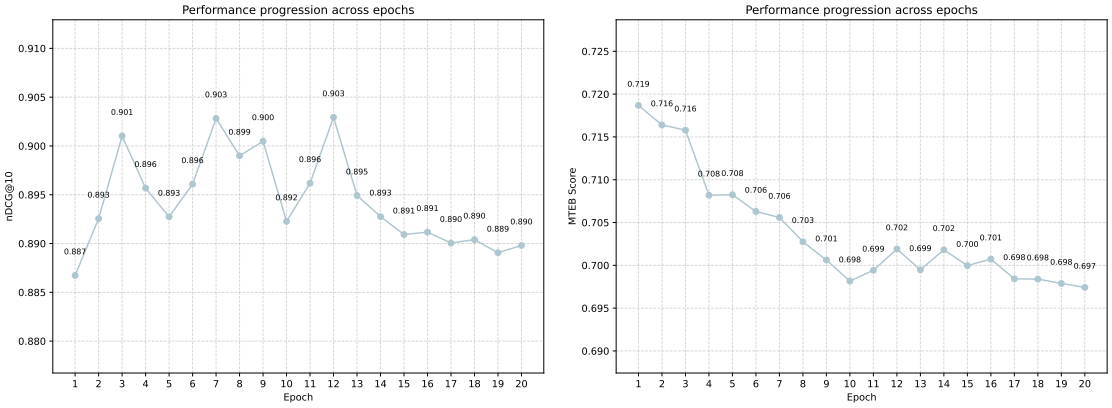

Figure 4.3: Evaluation of model checkpoints from each epoch on ChemRxiv retrieval (left) and ChemTEB retrieval (right) sets

Conclusions

In this chapter, we presented **ChEmbed**, an embedding model specifically adapted for retrieving chemical literature. Through a progressive approach to tokenizer augmentation and by training on large batches of synthetically generated query-passage pairs, **ChEmbed** achieved marked improvements in retrieval accuracy. On the ChemRxiv benchmark, our model achieved a substantial 9% gain in nDCG@10 over a general-purpose embedding model, outperforming even much larger proprietary models, while remaining efficient, fast, and supporting context windows of up to 2048 tokens. Our findings highlight several key aspects of domain adaptation. Most notably, using synthetic contrastive training helps overcome the frequent problem of limited data in specialized fields, such as chemistry. Furthermore, our efficient vocabulary augmentation method provided nearly as much improvement as fully retraining a tokenizer, but with significantly less complexity and resource utilization. We also demonstrated the importance of evaluating models on tasks closely aligned with their target domain; for example, ChemRxiv data proved more suitable than general-purpose benchmarks such as ChemTEB or MTEB. Although our study focused on chemistry, the techniques we developed, especially tokenizer augmentation and synthetic query-based contrastive training, are also applicable to other technical or scientific domains. Ultimately, **ChEmbed** offers clear benefits for chemical information retrieval, making literature searches and discovery processes more accurate, efficient, and practical for researchers.

Chapter 5

Conclusion

In this chapter, we will review the main problem addressed in this thesis, explain the solutions and techniques developed, discuss the lessons learned and the limitations, and suggest some directions for future research.

5.1 Problem Restatement

One of the main challenges in retrieval-augmented generation (RAG) systems is the quality of the retriever. In recent systems, this is usually a text embedding model, which converts text (or even other types of data such as images) into vectors, numerical representations that make it possible to search quickly and accurately through a knowledge base. This approach allows large language models (LLMs) to use external knowledge without changing their own parametric memory through extra training or fine-tuning. Most of the existing text embedding models are trained on large, general-purpose web-scale datasets. While these models work well for many tasks, they often underperform when used in specialized domains, like chemistry, which have their own technical language and vocabulary. This is a problem since the lack of domain adaptation can hurt the performance of retrieval systems, making it harder to find the most relevant documents in a RAG pipeline.

5.2 Contributions

The main goal of this thesis is to address the existing gap in domain-specific adaptation of text embedding models for chemistry information retrieval. The approach to this problem was developed in two main stages. First, the standard Massive Text Embedding Benchmark (MTEB), which is widely recognized for evaluating text embedding models, was extended by introducing a dedicated chemistry-focused

evaluation suite. This new benchmark enabled a thorough assessment of both open-source and proprietary embedding models on a range of chemistry-specific retrieval tasks, allowing for direct comparison and the identification of the most promising open-source model as a foundation for further development. Building upon this evaluation, the second stage focused on adapting the selected model for the chemistry domain. Key contributions of this thesis are as follows:

1. **ChemTEB, comprehensive Chemical Text Embedding Benchmark:** Developed a 34-task benchmark that enables researchers to evaluate retrieval, classification, clustering, and bitext-mining in realistic chemical language. This benchmark addresses the shortcomings of general NLP evaluations and is fully open-source and is integrated into the MTEB Python package[1], encouraging broad community adoption.

2. **ChemRxiv Retrieval task for real-world literature search:** Curated a paragraph-level retrieval benchmark based on academic articles, allowing models to be tested on the kind of dense and technical prose commonly encountered by chemists in practice.

3. **Scalable LLM-based pipeline for synthetic query generation:** Designed and validated an automated workflow that generated 1.7 million queries for chemistry-related paragraphs, demonstrating that high-quality contrastive pretraining is achievable even when manual labelling is limited or absent.

4. **ChEmbed; a domain-adapted embedding family that sets new state-of-the-art:** Fine-tuned a bi-encoder pretrained model on the synthetic dataset, achieving a notable improvement on the ChemRxiv benchmark compared to the base model, while maintaining efficiency.

5. **Light-weight tokenizer adaptation with chemical tokens:** Trained a tokenizer on a collection of IUPAC names and augmented the unused token slots of a general-domain tokenizer with the resulting chemistry-specific tokens, effectively improving retrieval accuracy without requiring a full retraining of the tokenizer.

6. **Open resources for the community:** Released all training and evaluation data, synthetic pair generation scripts, trained model checkpoints, and evaluation code under permissive licenses, facilitating follow-up research and lowering the barrier to entry for other scientific fields.

[1] https://github.com/embeddings-benchmark/mteb

5.3 Key Findings

The **ChemTEB** benchmark showed very clearly that no single embedding model wins across all task categories, because each model's pretraining and data shape where it shines. For example, the proprietary OpenAI `text-embedding-3-large` model leads overall, outperforming in three of the five task categories, but it still loses to other models on specific tasks. Among the open-source models, `nomic-embed-text-v1.5` and its predecessor `nomic-embed-text-v1` showed best performance respectively; with the latter being the only one whose code, weights, and training data are fully public, which makes it uniquely transparent and reproducible. Tasks that mostly use general-purpose text, like classification which is mostly Wikipedia-derived, are the easiest ones, because most models have already seen similar content during pretraining. In contrast, the bitext mining tasks, where models must align a sentence in natural language with another in a specialized chemical notation score almost zero for most models, since they have not learned to bridge natural language and chemical representations like SMIELS and SELFIES. We also found that BERT-based models simply fine-tuned with masked language modelling on chemical text lag behind newer, contrastively trained architectures: they may pick up a little chemistry knowledge, but they do not learn powerful semantic representations, showing that modern training methods are far more effective than a plain domain fine-tune with MLM. In developing **ChEmbed**, we started with the `nomic-embed-text-v1` backbone as the base model and showed that with the right domain adaptation, a compact model can outperform much larger ones. To get the appropriate data to train the model, we generated millions of synthetic queries for paragraphs from PubChem, Semantic Scholar articles, and ChemRxiv preprints, which let us build both training and evaluation datasets at scale. We tried adding mined hard negatives and random negatives to our contrastive pipeline, but they did not improve performance much over the default in-batch negatives, probably because those *hard* examples do not actually carry stronger supervision than our positive pairs. For tokenizer adaptation, we added 900 unique chemistry-related tokens via training a WordPiece tokenizer on chemical compound IUPAC names and tested several adaptation strategies; we found that a two-stage method; first updating only the new tokens, then fine-tuning the whole model, worked best, improving retrieval quality more than other approaches. Using this method, ChEmbed outperformed the `nomic-embed-text-v1` base model by approximately 9% on a ChemRxiv literature retrieval task and also surpassed all the open-source and well-known proprietary models we tested. Overall, this approach, combining synthetic query generation, targeted tokenizer adaptation, and staged fine-tuning, provides a practical and reproducible recipe for building specialized embeddings for any scientific or technical field, not just chemistry.

5.4 Limitations

ChemTEB While ChemTEB is specifically designed with a chemistry focus, it nevertheless retains an inherent bias towards general-purpose data, particularly evident in tasks involving classification, clustering, and retrieval. This bias arises primarily due to the use of datasets partially sourced from general-domain repositories, which may not entirely reflect the nuanced complexity and specialized language of chemistry literature. It was the main reason we extended it with an additional ChemRxiv retrieval task that was more representative of our training data. Additionally, ChemTEB currently lacks tasks explicitly designed for multi-hop reasoning or detailed reranking evaluation, which limits its capacity to assess models' abilities to handle complex, chained retrieval scenarios common in real-world chemical research. Lastly, the benchmark exclusively focuses on English-language resources, thus overlooking the substantial volume of chemical knowledge available in other languages, which restricts its applicability in global contexts.

ChemTEB While ChemTEB is specifically designed with a chemistry focus, it nevertheless retains an inherent bias towards general-purpose data (about 20/35 tasks use general-domain sources, vs. 15/35 from chemistry-native sources), particularly evident in tasks involving classification, clustering, and retrieval. This bias arises primarily due to the use of datasets partially sourced from general-domain repositories, which may not entirely reflect the nuanced complexity and specialized language of chemistry literature. This was a key reason we added, in Chapter 4, a separate ChemRxiv retrieval task built from primary literature to complement ChemTEB's encyclopedic sets. Additionally, ChemTEB currently lacks tasks explicitly designed for multi-hop reasoning or detailed reranking evaluation, which limits its capacity to assess models' abilities to handle complex, chained retrieval scenarios common in real-world chemical research. Lastly, the benchmark exclusively focuses on English-language resources, thus overlooking the large body of chemical literature in other languages, which limits applicability outside English-language contexts.

ChEmbed A primary limitation of ChEmbed is its monolingual nature; being trained exclusively on English-language chemical data significantly restricts its practical applicability, especially in multilingual environments or for patent databases containing extensive non-English chemical literature. Exploring the use of multilingual base models, such as the recently introduced `nomic-embed-text-v2-moe` [123], could also potentially address this gap. Moreover, ChEmbed was intentionally fine-tuned for retrieval tasks to enhance performance specifically in retrieval-augmented generation pipelines for chemistry. Consequently, when deployed for other tasks such as classification or clustering, the model may not perform optimally. Addressing this

issue would require further efforts to create specialized training datasets tailored for these additional tasks. Lastly, while tokenization improvements were noteworthy, the automated process for selecting tokens from a large corpus of IUPAC names could be further refined. Collaborating with domain experts in chemistry and linguistics might enable a more selective and impactful tokenization strategy.

5.5 Future Research Ideas

Based on the key findings and limitations of the ChEmbed, we propose the following paths towards building better domain-specific embedding models:

1. **Chemical Cross-Encoder:** Train a chemistry reranker and use it in a two-stage retrieval pipeline. A fast bi-encoder first fetches the top-k passages; a cross-encoder then reads each query-passage pair together, scores relevance, and reorders the list. During training, the re-ranker can also supply hard negatives and target scores to improve the retriever.

2. **Adapt Decoder-Based Language Models for Embeddings:** Most early embedding models use pretrained encoder-based architectures as backbones, but employing decoder-style large language models (LLMs) is a promising direction. Pretrained generative models possess strong reasoning abilities that could be distilled into effective embeddings. Recent work has shown that fine-tuning large decoder models for embedding tasks can yield state-of-the-art results [209]. Future research could focus on domain-adapting these decoder-based models for chemical texts, while addressing efficiency and scalability challenges due to their size.

3. **Multimodal Embedding Models:** Many applications in chemistry involve not just chemistry-related natural language, but also molecular representations and even images such as spectra or molecular structures. An important extension would be to train joint embedding models across multiple modalities, which can map text, SMILES, molecular graphs, and images into a joint semantic space.

4. **Multilingual Embedding Models:** Chemical knowledge is global, yet most domain embedding models are limited to English. A key research direction is to develop multilingual embeddings for cross-lingual retrieval, enabling queries in one language to find documents in another. This could be achieved by synthesizing data in different languages and selecting multilingual embedding models as the base model for further processing.

5. **Joint Retriever-Generator Optimization:** Embedding models are often used in retrieval-augmented generation. Future work could involve co-training

the retriever alongside a generative LLM designed for chemistry tasks such as question answering or summarization.

6. **Diverse Synthetic Queries:** Use multiple LLM families and different prompt styles (patent, safety, clinician, student) to reduce generator bias in training and evaluation.

7. **Expert-Guided Chemistry Vocabulary:** Work with chemistry experts to pick and verify candidate tokens, rather than relying only on algorithms (WordPiece/BPE). Add a token only when experts confirm it is meaningful and useful. This aims to cut over-segmentation and capture full chemical spans beyond IUPAC names.

Bibliography

[1] A. Acharya, S. Sharma, R. Cosbey, M. Subramanian, S. Howland, and M. Glenski. Exploring the benefits of domain-pretraining of generative large language models for chemistry. *arXiv preprint arXiv:2411.03542*, 2024.

[2] B. Adler, N. Agarwal, A. Aithal, D. H. Anh, P. Bhattacharya, A. Brundyn, J. Casper, B. Catanzaro, S. Clay, J. Cohen, et al. Nemotron-4 340b technical report. *arXiv preprint arXiv:2406.11704*, 2024.

[3] A. F. Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[4] W. Ahmad, E. Simon, S. Chithrananda, G. Grand, and B. Ramsundar. Chemberta-2: Towards chemical foundation models. *arXiv preprint arXiv:2209.01712*, 2022.

[5] W. Ammar, D. Groeneveld, C. Bhagavatula, I. Beltagy, M. Crawford, D. Downey, J. Dunkelberger, A. Elgohary, S. Feldman, V. Ha, et al. Construction of the literature graph in semantic scholar. *arXiv preprint arXiv:1805.02262*, 2018.

[6] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[7] V. Bagal, R. Aggarwal, P. Vinod, and U. D. Priyakumar. Molgpt: molecular generation using a transformer-decoder model. *Journal of Chemical Information and Modeling*, 62(9):2064–2076, 2021.

[8] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] P. BehnamGhader, V. Adlakha, M. Mosbach, D. Bahdanau, N. Chapados, and S. Reddy. Llm2vec: Large language models are secretly powerful text encoders. *arXiv preprint arXiv:2404.05961*, 2024.

[10] I. Beltagy, K. Lo, and A. Cohan. Scibert: A pretrained language model for scientific text. 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)*, pages 3615–3620, 2019.

[11] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[12] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[13] M. J. Bommarito, D. M. Katz, and J. Bommarito. Kl3m tokenizers: A family of domain-specific and character-level tokenizers for legal, financial, and preprocessing applications. *arXiv preprint arXiv:2503.17247*, 2025.

[14] L. Bonifacio, H. Abonizio, M. Fadaee, and R. Nogueira. Inpars: Data augmentation for information retrieval using large language models. *arXiv preprint arXiv:2202.05144*, 2022.

[15] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022.

[16] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[17] A. M. Bran, S. Cox, O. Schilter, C. Baldassari, A. D. White, and P. Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

[18] J. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Advances in neural information processing systems*, 2, 1989.

[19] N. Brown, M. Fiscato, M. H. Segler, and A. C. Vaucher. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59(3):1096–1108, 2019.

[20] P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[21] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

[22] C. Burges, R. Ragno, and Q. Le. Learning to rank with nonsmooth cost functions. *Advances in neural information processing systems*, 19, 2006.

[23] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[24] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007.

[25] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

[26] D. Chen, A. Fisch, J. Weston, and A. Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

[27] 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. *arXiv preprint arXiv:2402.03216*, 2024.

[28] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[29] S. Chen, S. Wong, L. Chen, and Y. Tian. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023.

[30] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PmLR, 2020.

[31] S. Chithrananda, G. Grand, and B. Ramsundar. Chemberta: large-scale self-supervised pretraining for molecular property prediction. *arXiv preprint arXiv:2010.09885*, 2020.

[32] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[33] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24 (240):1–113, 2023.

[34] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. 2011.

[35] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, 2019.

[36] G. V. Cormack, C. L. Clarke, and S. Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 758–759, 2009.

[37] E. Cortes, V. Woloszyn, A. Binder, T. Himmelsbach, D. Barone, and S. Möller. An empirical comparison of question classification methods for question answering systems. In *Proceedings of the twelfth language resources and evaluation conference*, pages 5408–5416, 2020.

[38] Z. Dai, V. Y. Zhao, J. Ma, Y. Luan, J. Ni, J. Lu, A. Bakalov, K. Guu, K. B. Hall, and M.-W. Chang. Promptagator: Few-shot dense retrieval from 8 examples. *arXiv preprint arXiv:2209.11755*, 2022.

[39] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

[40] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

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

[43] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[44] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015.

[45] J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[46] emozilla. Dynamically scaled rope further increases performance of long context llama with zero fine-tuning, 2023. URL https://www.reddit.com/r/LocalLLaMA/comments/14mrgpr/dynamically_scaled_rope_further_increases/.

[47] B. Fabian, T. Edlich, H. Gaspar, M. Segler, J. Meyers, M. Fiscato, and M. Ahmed. Molecular representation learning with language models and domain-relevant auxiliary tasks. *arXiv preprint arXiv:2011.13230*, 2020.

[48] T. Formal, C. Lassance, B. Piwowarski, and S. Clinchant. Splade v2: Sparse lexical and expansion model for information retrieval. *arXiv preprint arXiv:2109.10086*, 2021.

[49] T. Formal, B. Piwowarski, and S. Clinchant. Splade: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2288–2292, 2021.

[50] N. C. Frey, R. Soklaski, S. Axelrod, S. Samsi, R. Gomez-Bombarelli, C. W. Coley, and V. Gadepally. Neural scaling of deep chemical models. *Nature Machine Intelligence*, 5(11):1297–1305, 2023.

[51] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[52] L. Gao, Y. Zhang, J. Han, and J. Callan. Scaling deep contrastive learning batch size under memory limited setup. *arXiv preprint arXiv:2101.06983*, 2021.

[53] A. Gaulton, L. J. Bellis, A. P. Bento, J. Chambers, M. Davies, A. Hersey, Y. Light, S. McGlinchey, D. Michalovich, B. Al-Lazikani, et al. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40 (D1):D1100–D1107, 2012.

[54] P. Goyal, D. Mahajan, A. Gupta, and I. Misra. Scaling and benchmarking self-supervised visual representation learning. In *Proceedings of the ieee/cvf International Conference on computer vision*, pages 6391–6400, 2019.

[55] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[56] Y. Gu, R. Tinn, H. Cheng, M. Lucas, N. Usuyama, X. Liu, T. Naumann, J. Gao, and H. Poon. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH)*, 3(1):1–23, 2021.

[57] T. Gupta, M. Zaki, N. A. Krishnan, and Mausam. Matscibert: A materials domain language model for text mining and information extraction. *npj Computational Materials*, 8(1):102, 2022.

[58] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR, 2020.

[59] Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[60] J. He, D. Q. Nguyen, S. A. Akhondi, C. Druckenbrodt, C. Thorne, R. Hoessel, Z. Afzal, Z. Zhai, B. Fang, H. Yoshikawa, et al. Chemu 2020: Natural language processing methods are effective for information extraction from chemical patents. *Frontiers in Research Metrics and Analytics*, 6:654438, 2021.

[61] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[62] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[63] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[64] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[65] L. Hirschman, A. Yeh, C. Blaschke, and A. Valencia. Overview of biocreative: critical assessment of information extraction for biology, 2005.

[66] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[67] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[68] J. Hong, T. Kim, H. Lim, and J. Choo. Avocado: Strategy for adapting vocabulary to downstream domain. *arXiv preprint arXiv:2110.13434*, 2021.

[69] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. *Advances in neural information processing systems*, 27, 2014.

[70] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025.

[71] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338, 2013.

[72] J. J. Irwin and B. K. Shoichet. Zinc- a free database of commercially available compounds for virtual screening. *Journal of chemical information and modeling*, 45(1):177–182, 2005.

[73] R. Irwin, S. Dimitriadis, J. He, and E. J. Bjerrum. Chemformer: a pre-trained transformer for computational chemistry. *Machine Learning: Science and Technology*, 3(1):015022, 2022.

[74] G. Izacard, M. Caron, L. Hosseini, S. Riedel, P. Bojanowski, A. Joulin, and E. Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

[75] K. M. Jablonka, P. Schwaller, A. Ortega-Guerrero, and B. Smit. Leveraging large language models for predictive chemistry. *Nature Machine Intelligence*, 6 (2):161–169, 2024.

[76] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[77] V. Jeronymo, L. Bonifacio, H. Abonizio, M. Fadaee, R. Lotufo, J. Zavrel, and R. Nogueira. Inpars-v2: Large language models as efficient dataset generators for information retrieval. *arXiv preprint arXiv:2301.01820*, 2023.

[78] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, 2002.

[79] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems (TOIS)*, 25(2):7–es, 2007.

[80] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[81] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

[82] V. Karpukhin, B. Oguz, S. Min, P. S. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781, 2020.

[83] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[84] O. Khattab and M. Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48, 2020.

[85] J.-D. Kim and S. Pyysalo. *BioNLP Shared Task*, pages 138–141. Springer New York, New York, NY, 2013. ISBN 978-1-4419-9863-7. doi: 10.1007/978-1-4419-9863-7_138. URL https://doi.org/10.1007/978-1-4419-9863-7_138.

[86] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, et al. Pubchem 2019 update: improved access to chemical data. *Nucleic acids research*, 47(D1):D1102–D1109, 2019.

[87] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, L. Zaslavsky, J. Zhang, and E. E. Bolton. PubChem 2023 update. *Nucleic Acids Research*, 51(D1):D1373–D1380, 10 2022. ISSN 0305-1048. doi: 10.1093/nar/gkac956. URL https://doi.org/10.1093/nar/gkac956.

[88] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, et al. Pubchem 2023 update. *Nucleic acids research*, 51 (D1):D1373–D1380, 2023.

[89] R. Kinney, C. Anastasiades, R. Authur, I. Beltagy, J. Bragg, A. Buraczynski, I. Cachola, S. Candra, Y. Chandrasekhar, A. Cohan, et al. The semantic scholar open data platform. *arXiv preprint arXiv:2301.10140*, 2023.

[90] M. Krallinger, O. Rabal, F. Leitner, M. Vazquez, D. Salgado, Z. Lu, R. Leaman, Y. Lu, D. Ji, D. M. Lowe, et al. The chemdner corpus of chemicals and drugs and its annotation principles. *Journal of cheminformatics*, 7:1–17, 2015.

[91] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[92] T. Kudo and J. Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[93] A. Kusupati, G. Bhatt, A. Rege, M. Wallingford, A. Sinha, V. Ramanujan, W. Howard-Snyder, K. Chen, S. Kakade, P. Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

[94] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[95] T. Le Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. 2023.

[96] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[97] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[98] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

[99] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

[100] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[101] J. Li and X. Jiang. Mol-bert: An effective molecular representation with bert for molecular property prediction. *Wireless Communications and Mobile Computing*, 2021(1):7181815, 2021.

[102] Z. Li, X. Zhang, Y. Zhang, D. Long, P. Xie, and M. Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023.

[103] Y. Liang, N. Duan, Y. Gong, N. Wu, F. Guo, W. Qi, M. Gong, L. Shou, D. Jiang, G. Cao, X. Fan, R. Zhang, R. Agrawal, E. Cui, S. Wei, T. Bharti, Y. Qiao, J.-H. Chen, W. Wu, S. Liu, F. Yang, D. Campos, R. Majumder, and M. Zhou. Xglue: A new benchmark dataset for cross-lingual pre-training, understanding and generation. *arXiv*, abs/2004.01401, 2020.

[104] J. Lin, X. Ma, S.-C. Lin, J.-H. Yang, R. Pradeep, and R. Nogueira. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 2356–2362, 2021.

[105] T.-Y. Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[106] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[107] K. Lo, L. L. Wang, M. Neumann, R. Kinney, and D. S. Weld. S2orc: The semantic scholar open research corpus. *arXiv preprint arXiv:1911.02782*, 2019.

[108] A. M. Bran, S. Cox, O. Schilter, C. Baldassari, A. D. White, and P. Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535, 2024.

[109] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[110] C. Manning and H. Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[111] C. D. Manning. *An introduction to information retrieval*. 2009.

[112] G. Marcus. The next decade in ai: four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177*, 2020.

[113] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

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

[115] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

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

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

[118] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022.

[119] M.-L. M.-F. Multi-Granularity. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation.

[120] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–15, 2021.

[121] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng. Ms marco: A human-generated machine reading comprehension dataset. 2016.

[122] J. Ni, C. Qu, J. Lu, Z. Dai, G. H. Ábrego, J. Ma, V. Y. Zhao, Y. Luan, K. B. Hall, M.-W. Chang, et al. Large dual encoders are generalizable retrievers. *arXiv preprint arXiv:2112.07899*, 2021.

[123] Z. Nussbaum and B. Duderstadt. Training sparse mixture of experts text embedding models. *arXiv preprint arXiv:2502.07972*, 2025.

[124] Z. Nussbaum, J. X. Morris, B. Duderstadt, and A. Mulyar. Nomic embed: Training a reproducible long context text embedder. *arXiv preprint arXiv:2402.01613*, 2024.

[125] NVIDIA. NVIDIA A100 Tensor Core GPU Architecture. Technical report, NVIDIA Corporation, May 2020. White Paper.

[126] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[127] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[128] X. Pan and V. Srikumar. Expressiveness of rectifier networks. In *International conference on machine learning*, pages 2427–2435. PMLR, 2016.

[129] L. Pang, Y. Lan, J. Guo, J. Xu, S. Wan, and X. Cheng. Text matching as image recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[130] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

[131] B. Peng, E. Chersoni, Y.-Y. Hsu, C.-R. Huang, et al. Is domain adaptation worth your investment? comparing bert and finbert on financial tasks. Association for Computational Linguistics (ACL), 2021.

[132] B. Peng, J. Quesnelle, H. Fan, and E. Shippole. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

[133] Y. Peng, S. Yan, and Z. Lu. Transfer learning in biomedical natural language processing: an evaluation of bert and elmo on ten benchmarking datasets. *arXiv preprint arXiv:1906.05474*, 2019.

[134] 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)*, pages 1532–1543, 2014.

[135] E. Pereira. Msds-opp: Operator procedures prediction in material safety data sheets. 02 2020. doi: 10.24840/978-972-752-264-4.

[136] E. Pereira. Msds-opp: Operator procedures prediction in material safety data sheets. In *15th Doctoral Symposium*, page 42, 2020.

[137] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*, 2019.

[138] D. Polykovskiy, A. Zhebrak, B. Sanchez-Lengeling, S. Golovanov, O. Tatanov, S. Belyaev, R. Kurbanov, A. Artamonov, V. Aladinskiy, M. Veselov, et al. Molecular sets (moses): a benchmarking platform for molecular generation models. *Frontiers in pharmacology*, 11:565644, 2020.

[139] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[140] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[141] 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. *Journal of machine learning research*, 21(140):1–67, 2020.

[142] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions, 2017. URL https://arxiv.org/abs/1710.05941.

[143] M. C. Ramos, C. J. Collison, and A. D. White. A review of large language models and autonomous agents in chemistry. *arXiv preprint arXiv:2407.01603*, 2024.

[144] recobo. BERT for Chemical Industry (chemical-bert-uncased). https://huggingface.co/recobo/chemical-bert-uncased, Aug. 2021. Accessed: 2025-06-08.

[145] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[146] A. Roberts, C. Raffel, and N. Shazeer. How much knowledge can you pack into the parameters of a language model? *arXiv preprint arXiv:2002.08910*, 2020.

[147] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, M. Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

[148] A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pages 410–420, 2007.

[149] F. Rosenblatt. *Perceptions and the theory of brain mechanisms*. Spartan books, 1962.

[150] J. Ross, B. Belgodere, V. Chenthamarakshan, I. Padhi, Y. Mroueh, and P. Das. Large-scale chemical language representations capture molecular structure and properties. *Nature Machine Intelligence*, 4(12):1256–1264, 2022.

[151] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[152] J. Saad-Falcon, D. Y. Fu, S. Arora, N. Guha, and C. Ré. Benchmarking and building long-context retrieval models with loco and m2-bert. *arXiv preprint arXiv:2402.07440*, 2024.

[153] G. Salton, A. Wong, and C.-S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[154] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[155] K. Santhanam, O. Khattab, J. Saad-Falcon, C. Potts, and M. Zaharia. Colbertv2: Effective and efficient retrieval via lightweight late interaction. *arXiv preprint arXiv:2112.01488*, 2021.

[156] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

[157] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[158] M. Schuster and K. Nakajima. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[159] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[160] C. E. Shannon. Prediction and entropy of printed english. *Bell system technical journal*, 30(1):50–64, 1951.

[161] N. Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[162] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[163] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd international conference on world wide web*, pages 373–374, 2014.

[164] A. Shiraee Kasmaee, M. Khodadad, M. Arshi Saloot, N. Sherck, S. Dokas, H. Mahyar, and S. Samiee. ChemTEB: Chemical text embedding benchmark, an overview of embedding models performance & efficiency on a specific domain. In *Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop*, volume 262 of *Proceedings of Machine Learning Research*, pages 512–531. PMLR, 14 Dec 2024. URL https://proceedings.mlr.press/v262/shiraee-kasmaee24a.html.

[165] A. Shiraee Kasmaee, M. Khodadad, M. Astaraki, M. A. Saloot, N. Sherck, H. Mahyar, and S. Samiee. Chembed: Enhancing chemical literature search through domain-specific text embeddings. *arXiv preprint arXiv:2508.01643*, 2025.

[166] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston. Retrieval augmentation reduces hallucination in conversation. *arXiv preprint arXiv:2104.07567*, 2021.

[167] A. Siddhant, J. Hu, M. Johnson, O. Firat, and S. Ruder. Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalization. In *Proceedings of the International Conference on Machine Learning 2020*, pages 4411–4421, 2020.

[168] K. Singhal, S. Azizi, T. Tu, S. S. Mahdavi, J. Wei, H. W. Chung, N. Scales, A. Tanwani, H. Cole-Lewis, S. Pfohl, et al. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180, 2023.

[169] K. Singhal, T. Tu, J. Gottweis, R. Sayres, E. Wulczyn, M. Amin, L. Hou, K. Clark, S. R. Pfohl, H. Cole-Lewis, et al. Toward expert-level medical question answering with large language models. *Nature Medicine*, pages 1–8, 2025.

[170] L. Soldaini and K. Lo. peS2o (Pretraining Efficiently on S2ORC) Dataset. Technical report, Allen Institute for AI, 2023. ODC-By, https://github.com/allenai/pes2o.

[171] M. Sorokina, P. Merseburger, K. Rajan, M. A. Yirik, and C. Steinbeck. Coconut online: collection of open natural products database. *Journal of Cheminformatics*, 13(1):2, 2021.

[172] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[173] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[174] W. Tai, H. Kung, X. L. Dong, M. Comiter, and C.-F. Kuo. exbert: Extending pre-trained models with domain-specific vocabulary under constrained training resources. In *Findings of the association for computational linguistics: EMNLP 2020*, pages 1433–1439, 2020.

[175] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663*, 2021.

[176] N. Thakur, J. Ni, G. H. Ábrego, J. Wieting, J. Lin, and D. Cer. Leveraging llms for synthesizing training data across many languages in multilingual dense retrieval. *arXiv preprint arXiv:2311.05800*, 2023.

[177] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[178] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[179] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, et al. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 12:1, 2016.

[180] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[181] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[182] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

[183] J. Wang, T. Ji, Y. Wu, H. Yan, T. Gui, Q. Zhang, X.-J. Huang, and X. Wang. Length generalization of causal transformers without position encoding. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 14024–14040, 2024.

[184] L. Wang, N. Yang, X. Huang, B. Jiao, L. Yang, D. Jiang, R. Majumder, and F. Wei. Simlm: Pre-training with representation bottleneck for dense passage retrieval. *arXiv preprint arXiv:2207.02578*, 2022.

[185] L. Wang, N. Yang, X. Huang, B. Jiao, L. Yang, D. Jiang, R. Majumder, and F. Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

[186] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei. Improving text embeddings with large language models. *arXiv preprint arXiv:2401.00368*, 2023.

[187] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei. Multilingual e5 text embeddings: A technical report. *arXiv preprint arXiv:2402.05672*, 2024.

[188] S. Wang, Y. Guo, Y. Wang, H. Sun, and J. Huang. Smiles-bert: large scale unsupervised pre-training for molecular property prediction. In *Proceedings of the 10th ACM international conference on bioinformatics, computational biology and health informatics*, pages 429–436, 2019.

[189] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[190] Y. Wang, L. Wang, Y. Li, D. He, and T.-Y. Liu. A theoretical analysis of ndcg type ranking measures. In *Conference on learning theory*, pages 25–54. PMLR, 2013.

[191] Y. Wang, J. Wang, Z. Cao, and A. Barati Farimani. Molecular contrastive learning of representations via graph neural networks. *Nature Machine Intelligence*, 4(3):279–287, 2022.

[192] J. Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1): 36–45, 1966.

[193] A. Williams, N. Nangia, and S. R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[194] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[195] C. Wu, W. Lin, X. Zhang, Y. Zhang, W. Xie, and Y. Wang. Pmc-llama: toward building open-source language models for medicine. *Journal of the American Medical Informatics Association*, 31(9):1833–1843, 2024.

[196] S. Wu and S. McClean. Evaluation of system measures for incomplete relevance judgment in ir. In *International Conference on Flexible Query Answering Systems*, pages 245–256. Springer, 2006.

[197] S. Wu and S. McClean. Information retrieval evaluation with partial relevance judgment. In *British National Conference on Databases*, pages 86–93. Springer, 2006.

[198] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[199] S. Xiao, Z. Liu, Y. Shao, and Z. Cao. Retromae: Pre-training retrieval-oriented language models via masked auto-encoder. *arXiv preprint arXiv:2205.12035*, 2022.

[200] S. Xiao, Z. Liu, P. Zhang, N. Muennighoff, D. Lian, and J.-Y. Nie. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pages 641–649, 2024.

[201] C. Xiong, Z. Dai, J. Callan, Z. Liu, and R. Power. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR conference on research and development in information retrieval*, pages 55–64, 2017.

[202] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[203] L. Xu, H. Hu, X. Zhang, L. Li, C. Cao, Y. Li, Y. Xu, K. Sun, D. Yu, C. Yu, et al. Clue: A chinese language understanding evaluation benchmark. *arXiv preprint arXiv:2004.05986*, 2020.

[204] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

[205] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[206] D. Zhang, W. Liu, Q. Tan, J. Chen, H. Yan, Y. Yan, J. Li, W. Huang, X. Yue, W. Ouyang, et al. Chemllm: A chemical large language model. *arXiv preprint arXiv:2402.06852*, 2024.

[207] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*, pages 649–666. Springer, 2016.

[208] X. Zhang, Y. Zhang, D. Long, W. Xie, Z. Dai, J. Tang, H. Lin, B. Yang, P. Xie, F. Huang, et al. mgte: Generalized long-context text representation and reranking models for multilingual text retrieval. *arXiv preprint arXiv:2407.19669*, 2024.

[209] Y. Zhang, M. Li, D. Long, X. Zhang, H. Lin, B. Yang, P. Xie, A. Yang, D. Liu, J. Lin, F. Huang, and J. Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

[210] S. Zhou, N. Wang, L. Wang, H. Liu, and R. Zhang. Cancerbert: a cancer domain-specific language model for extracting breast cancer phenotypes from electronic health records. *Journal of the American Medical Informatics Association*, 29(7):1208–1216, 2022.