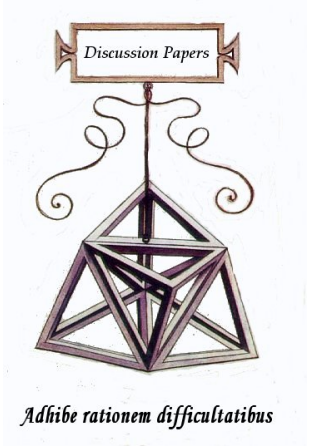
Discussion papers
E-papers of the Department of Economics and Management – University di Pisa



Maria Saveria Mavillonio

Natural Language Processing Techniques for Long Financial Document

Discussion paper n. 317

2024

Discussion paper n. 317, presented: **November 2024**

Authors' address/Indirizzo degli autori:

Maria Saveria Mavillonio — University of Pisa - Department of Economics and Management, Via Cosimo Ridolfi 10, 56124 Pisa – Italy. E-mail: mariasaveria.mavillonio@phd.unipi.it

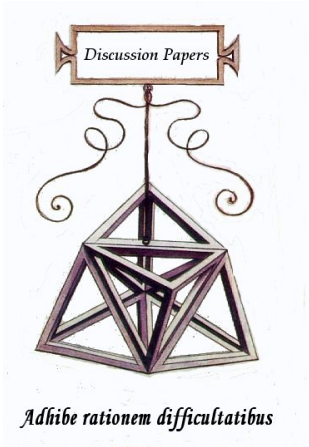
Please cite as:/Si prega di citare come:

Maria Saveria Mavillonio (2024), "Natural Language Processing Techniques for Long Financial Document", Discussion Papers, Department of Economics and Management – University of Pisa, n. 317 (http://www.ec.unipi.it/ricerca/discussion-papers).

Discussion Papers Series contact: pietro.battiston@unipi.it

Discussion paper

n. 317



Maria Saveria Mavillonio

Natural Language Processing Techniques for Long Financial Document

Abstract

In finance, Natural Language Processing (NLP) has become both a powerful and challenging tool, as extensive unstructured documents—such as business plans, financial reports, and regulatory filings—hold essential insights for strategic decision-making. This paper reviews the progression of NLP text representation methods, from foundational models to advanced Transformer architectures that greatly enhance semantic and contextual analysis. Yet, these models encounter limitations when applied to long financial documents, where computational efficiency and contextual coherence are critical. Recent innovations, including sparse attention mechanisms and domain-specific model adaptations, have improved the processing of lengthy texts, allowing for more accurate analysis of financial documents by capturing field-specific semantics. This paper also highlights the transformative role of NLP in financial analysis, especially where structured data is limited. Selecting the most suitable model for specific tasks is essential for maximizing NLP's impact in finance. Organized to provide a thorough overview, the paper covers text representation techniques, strategies for handling long texts, and applications in finance, establishing a foundation for advancing NLP-driven data analysis in this field.

Keywords: Long Text; Financial Document Representation; Natural Language Processing; Transformers

JEL CLassification: C45; G2; G23; L26

Natural Language Processing Techniques for Long Financial Document

Maria Saveria Mavillonio*

Abstract

In finance, Natural Language Processing (NLP) has become both a powerful and challenging tool, as extensive unstructured documents—such as business plans, financial reports, and regulatory filings—hold essential insights for strategic decision-making. This paper reviews the progression of NLP text representation methods, from foundational models to advanced Transformer architectures that greatly enhance semantic and contextual analysis. Yet, these models encounter limitations when applied to long financial documents, where computational efficiency and contextual coherence are critical. Recent innovations, including sparse attention mechanisms and domain-specific model adaptations, have improved the processing of lengthy texts, allowing for more accurate analysis of financial documents by capturing field-specific semantics. This paper also highlights the transformative role of NLP in financial analysis, especially where structured data is limited. Selecting the most suitable model for specific tasks is essential for maximizing NLP's impact in finance. Organized to provide a thorough overview, the paper covers text representation techniques, strategies for handling long texts, and applications in finance, establishing a foundation for advancing NLP-driven data analysis in this field.

Keywords: Long Text, Financial Document Representation, Natural Language Processing, Transformers

JEL Classification: C45, C53, G2, G23, L26

*Department of Economics and Management, University of Pisa, Italy.
Corresponding author: Email: mariasaveria.mavillonio@phd.unipi.it
This research was supported by funding from the PRIN grant no. 20177FX2A7, provided by the Italian Ministry of University and Research.

1 Introduction

In recent years, Natural Language Processing (NLP) has become an essential tool for extracting and interpreting information from large volumes of unstructured textual data across a wide range of domains. As Chowdhary defines it, NLP is the study and development of computational systems that can process human language to derive useful insights [30]. A core challenge in NLP involves representing long-form text in a way that captures its contextual depth without sacrificing computational efficiency. This task is especially relevant in domains like finance, where extended documents—such as business plans, financial reports, and regulatory filings—contain complex and nuanced language critical to decision-making. Accordingly, advancements in NLP techniques for text representation have enabled more sophisticated analysis of these types of documents, opening new avenues for data-driven insights where traditional data sources are limited.

This paper examines the evolution of NLP methods for text representation, tracing the progression from early, foundational models to advanced deep learning techniques that leverage attention mechanisms for more nuanced understanding. We begin by reviewing the development of NLP representation methods, from basic models like Bag-of-Words (BoW) to distributed embeddings such as Word2Vec and GloVe, culminating in the advent of Transformer-based architectures like BERT. These advancements significantly improved NLP's capacity to capture context and semantics, though they also introduced challenges, such as high computational demands when processing extensive datasets.

We then address the specific challenges posed by long texts, where preserving coherence and contextual relevance across lengthy documents becomes complex. Recent innovations, including adaptations to Transformer architectures, have provided strategies to manage long-form texts efficiently, enabling NLP to handle extended documents more effectively.

Lastly, we explore applications of NLP in the financial sector, where complex documents require specialized processing. The literature highlights the use of NLP in tasks like sentiment analysis, risk assessment, and forecasting, illustrating how these techniques have transformed financial document analysis by offering insights in settings where structured data may be scarce.

This paper is structured as follows: in the section 2 we first provide a historical and technical overview of text representation techniques, then discuss methods specifically for long-text representation in the section 3, and finally in the section 4 examine NLP applications in financial document analysis. This organization offers both a foundational understanding of NLP advancements in text representation and a focused review of their impact in finance, setting the stage for exploring NLP's broader potential in alternative data analysis.

2 Natural Language Processing

Natural Language Processing (NLP) involves enabling computers to understand and process human languages, such as English or Italian. While natural languages are intuitive for humans, they pose significant challenges for computers due to ambiguities, context dependencies, and the lack of rigid formal structures. NLP encompasses a wide range of applications. For instance, machine translation requires a program to read a sentence in one language (e.g., English) and generate an equivalent sentence in another language (e.g., Italian) that preserves the original meaning [54, 68].

Another significant application of NLP is automated summarization, which condenses lengthy texts into brief summaries, preserving essential information [1]. This application is widely used in journalism and academia to distill lengthy articles or research papers into concise summaries, allowing readers to grasp main points quickly. In finance and macroeconomics, NLP algorithms analyze news feeds, economic reports, and earnings statements to gauge market and economic sentiment, aiding in investment decisions and policy evaluations. By extracting insights from these large datasets, NLP helps analysts understand market trends and economic indicators, providing valuable input for both financial forecasting and macroeconomic assessments [9, 19, 4, 5].

Information retrieval systems play a crucial role in extracting relevant information from vast datasets, as seen in search engines, recommendation systems, and digital libraries. NLP techniques such as query understanding, semantic search, and relevance ranking allow IR systems to interpret user queries accurately and rank results by relevance [84].

Another innovative task is Text-to-Speech (TTS) that converts written text into spoken language, widely used in accessibility tools, virtual assistants, and customer service. TTS leverages NLP to capture natural prosody and intonation, producing lifelike speech output. Advanced models, including Tacotron [91] and WaveNet [88], are often used to ensure the speech sounds authentic and adaptable to various tones and expressions [96]. In social sciences, NLP is applied to analyze large datasets of social media posts, helping researchers study social trends, behaviors, and public health issues [87].

In any NLP task, a critical first step is determining the type of text to use as input, as document types can vary widely based on task requirements. For instance, in macroeconomic forecasting, financial news articles might be aggregated at a chosen time frequency (daily, quarterly, etc.), whereas more detailed analyses might focus on reports such as government economic reviews or industry-specific briefings. Similarly, earnings reports, corporate filings, and policy documents each offer unique insights that are relevant for specific downstream tasks, such as sentiment analysis, economic indicator prediction, or trend analysis.

After defining the input text type, it typically cannot be used directly in an NLP model. Instead, a sequence of preparatory steps is required to make the text suitable for machine processing—a process known as the NLP pipeline. This pipeline encompasses everything from initial text preparation to the generation of final predictions, ensuring that input data is appropriately structured and preprocessed to optimize model performance.

In the next sections, we first details the whole NLP pipeline by introducing the steps that are usually involved. Then, in last two sections, we focus on a crucial step of the pipeline: how to obtain a vectorial representation of the words and the text.

2.1 The NLP Pipeline

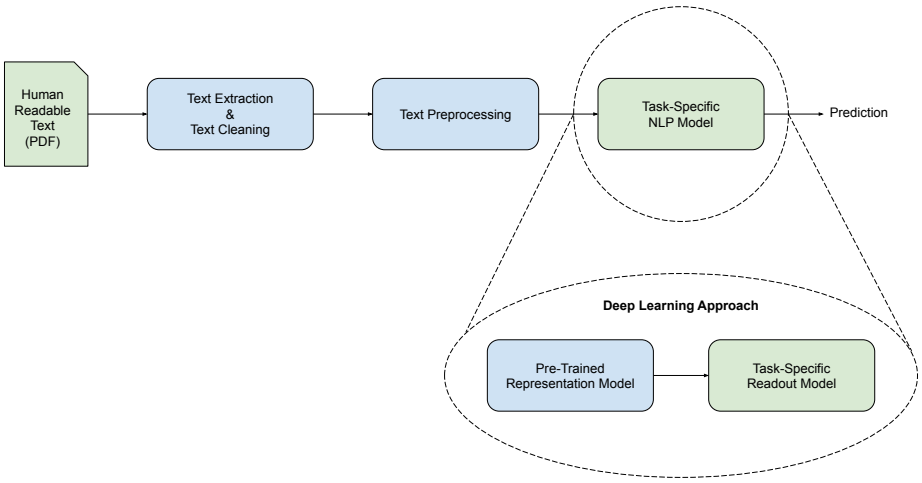


Figure 1: A sketch of the NLP pipeline. In the dashed circle, we represent the current approach based on DL.

The NLP pipeline (see Figure 1) begins with extraction and cleaning, where text document from diverse formats (such as PDFs, HTML, or Word documents) is converted into a raw text, with unnecessary elements like HTML tags and special characters removed. Following this, pre-processing aim to prepare the text for the subsequent NLP model. The pre-processing can include different steps such as remove punctuation and extra spaces, normalize it by converting to lowercase, and tokenize it by breaking the text into individual words or phrases. During the pre-processing we an also filter out non-informative words like "the" or "is" (these are called *stop words*), or non-informative grammatical roles, i.e. parts-of-speech (POS). Further, stemming and lemmatization can be applied to reduce words to

their root forms, enhancing consistency across variations like "running" and "run". It is worth highlighting that the choice of the pre-processing steps that should be applied depdends on the NLP model we aim to apply.

The final step, NLP modeling, is tailored to the specific task and, in the deep learning era, is generally divided into two phases. First, a pre-trained representation model—such as BERT or GPT—is applied to obtain rich, task-independent semantic embeddings of the text. These embeddings are then fed into a classifier or regressor based on the task, whether it's sentiment classification, named entity recognition, or prediction. This approach of using a general representation model followed by a task-specific model enables a versatile and effective pipeline, allowing NLP systems to handle a wide range of applications with enhanced accuracy and semantic understanding.

2.1.1 Text Extraction & Text Cleaning

Often, the documents of interest are not machine-readable. One of the most used format for document is the PDF [1]. The goal of the PDF format is to present documents—including text, formatting, and images—in a consistent manner, independent of software, hardware, or operating system. Built on the PostScript language, each PDF file contains a complete description of a fixed-layout document, including text, fonts, vector graphics, raster images, and all necessary information for accurate display. While PDFs can also include additional elements beyond basic text and graphics (such as logical structure, interactive annotations, form fields, layers, and multimedia), these aspects fall outside the scope of this discussion.

There are two types of PDF textual document:

- *native PDFs* are the ones that were "born digital", i.e. the PDFs were created from an electronic version of a document (for example, by using a word processor program);
- *scanned PDFs* are obtained from printed documents:the pages of the document have beeen scanned and stored in PDF file.

Scanned documents are the more challenging to extract since the PDF file contains only images. To extract the text, it is necessary to employ an Optical Character Reader (OCR), i.e. a computer program which is able to convert images of typed, handwritten or printed text into machine-encoded text. The result of the conversion are highly affected from the quality of the input images.

Nevertheless, also the text extraction from native PDFs is not straightforward. Since the PDF format focuses on the appearance of the documents, it usually lacks

[1] Portable Document Format (PDF) is standardized as ISO 32000 and developed by Adobe in 1992.

of explicit structural metadata, such as headings, paragraphs, or the semantics of tables. This absence complicates the identification of different content types and the understanding of their hierarchy and interrelationships. Moreover, each PDF file can have its own layout, that can differ from page to page; for example, some pages can be formatted into a multiple-columns layout, while other can contain tables or an image.

The output of the text extraction is the so-called *raw text*, where all the information about the appearance (e.g. the font information, the position in the page, etc..) are discarded. The only information left in the raw text is the sequence of characters, stored according an encoding schema such as ASCII or UTF-8.

2.1.2 Text Preprocessing

Before applying any algorithm, raw text must be transformed into sequences of linguistic features, commonly referred to as tokens in NLP. Let $\mathcal{D}$ be a document, we represent it as:

$$\mathcal{D} = (w_1, w_2, \ldots, w_N)\,,$$

where w_i is the i-token of the document, and N is the number of tokens in $\mathcal{D}$. The transformation of a raw text in a sequence of tokens is called *pre-processing*.

The steps required in the pre-processing depend on the application domain and the NLP model involved. For example, in the field of economics, the standard pre-processing typically involves *tokenization* and *stemming*. This standard approach has been extensively covered in prior works (e.g., [63, 48, 45, 6].

Nowadays, with the advent of the Transformers era, the usual pre-processing comprises only the tokenization. However, it is worth highlighting that the final performances are affected from the quality of the raw text. Thus, it always recommended to provide the raw text as clean as possible (e.g. removing urls).

Below, we outline several key techniques used to clean preprocess text data, thereby eliminating noise and enhancing the extraction of meaningful features.

Tokenization is a foundational preprocessing technique that segments a text stream into smaller units, such as words, phrases, or symbols, known as tokens. The primary objective is to break down a sentence into its constituent words for further analysis. Tokenization is a critical first step in both text classification and text mining, providing the parser with a structured input for downstream tasks. For example, *Don't you hate that? What? Uncomfortable silences!* can be transformed in `Do", n't", you", hate", that" ?", What ", ?", Uncomfortable", silences"."` by leveraging the tokenize function of the Python-based Natural Language Toolkit (NLTK) [16].

Stopwords Removal. In many classification tasks, common words such as articles, conjunctions, and prepositions (e.g. "a", "an", "the", "and", "but", "or") provide little value for the learning algorithms. Stopword removal is a standard preprocessing technique used to filter out these words, thereby reducing dimensionality and focusing the model on more informative tokens.

Case-folding, often achieved by converting all characters to lowercase, is a standard approach in text normalization, helping match terms like "bank" with "Bank" or "apple" with "Apple". This strategy enhances consistency across large corpora and simplifies search queries when users input lowercase, like typing "amazon" for "Amazon" (the company). However, full case-folding can blur essential distinctions, especially for proper nouns and acronyms that rely on capitalization to convey unique meanings, such as "Fed" (Federal Reserve) vs. "fed" (past tense of feed) or "US" (United States) vs. "us" (pronoun).

Selective case-folding offers a more flexible approach by converting only certain tokens, such as sentence-initial words or fully capitalized titles, to lowercase, leaving mid-sentence capitalization intact to maintain key distinctions. Machine learning models, known as truecasing, use contextual data to improve these decisions, identifying where capitalization carries specific meaning. Exception handling, including slang and abbreviation tools, can further manage ambiguities in context. Nonetheless, given users' tendency to type in lowercase, full lowercasing remains an efficient solution in many scenarios [63].

Slang and Abbreviation often are included in the informal text, particulary on social media, that can hinder accurate interpretation. For example, abbreviations like ASAP (for as soon as possible) or slang phrases like spill the tea (meaning "reveal gossip") add complexity to natural language processing. During preprocessing, these informal expressions are typically converted into their formal counterparts to improve model comprehension and consistency.

This conversion step is crucial because models trained on standardized language often struggle to interpret less formal or context-specific expressions accurately. By translating slang and abbreviations into more universally understood terms, preprocessing enhances the model's performance, aligning the text data with a consistent and structured feature space, thereby improving interpretation and analysis outcomes.

Parts-of-Speech (PoS) tagging is a fundamental task that assigns a grammatical category, such as noun, verb, adjective, or adverb, to each word in a text. By capturing the structure and semantics of phrases, PoS tagging enhances machine understanding of human language, allowing for more accurate language

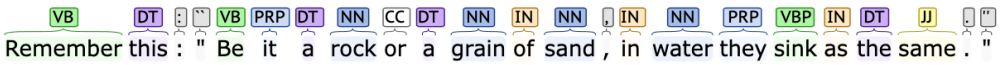


Figure 2: Example of POS tagging using CoreNLP.

analysis. This process acts as a bridge between raw language data and machine comprehension, aiding in the correct interpretation of the phrase's meaning and structure.

For example, consider the phrase: *Remember this: "Be it a rock or a grain of sand, in water they sink as the same."* Applying PoS tagging yields the output in the Figure 2, where each word is assigned a grammatical category, such as VB (verb), DT (determiner), NN (noun), JJ (adjective), PRP (pronoun), CC (conjunction), and IN (preposition).

Noise Entities Removal in text data, such as extra punctuation marks, special characters (e.g., , #, $), and other irrelevant symbols, can hinder the performance of classification algorithms. For instance, hashtags (#BlackLivesMatter) or excess punctuation (!!!) may add unnecessary complexity without contributing to the core meaning. Removing these elements during preprocessing streamlines the feature set, enabling the model to focus on meaningful linguistic patterns rather than extraneous details. This step helps improve classification accuracy by ensuring that the model prioritizes essential information within the text.

Spelling Correction is essential for improving NLP accuracy, especially in user-generated content where misspellings are common. Techniques for error correction often involve checking each word against a dictionary, flagging any words not found as potential errors, such as "recieve", which can be corrected to "receive". Edit distance algorithms like Levenshtein and Damerau-Levenshtein measure [39] the minimum edits needed to align a misspelled word with known terms, suggesting corrections with minimal differences. Context-aware approaches further enhance accuracy by using surrounding words to refine choices, as with their vs. there. Additionally, Trie structures allow efficient lookup by grouping words based on shared prefixes, enabling quick correction. Together, these techniques improve data quality and ensure consistency, enhancing model performance across NLP tasks.

Lemmatization is the process of identifying and reducing different word forms to a common root, or lemma, regardless of surface-level variations. For example, "am", "are", and "is" are all lemmatized to the root "be", while "walking", "walks", and "walked" are reduced to "walk". This approach facilitates the consistent recognition

of a concept across its various morphological forms [2].

In practice, lemmatization simplifies complex sentences to their essential components while preserving their core meaning. For instance, the sentence "The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom" would be transformed to "The sad aspect of life right now be that science gather knowledge fast than society gather wisdom." This structure retains the sentence's essential meaning, enabling more effective analysis by focusing on root forms. Lemmatization is particularly advantageous in contexts requiring high precision, as it groups words based on shared meanings despite surface differences, supporting a more accurate and cohesive text representation.

Stemming is a straightforward technique in morphological analysis that reduces words to their base form by removing final affixes, such as suffixes and prefixes. Unlike lemmatization, which often relies on complex algorithms to find the precise dictionary base form of a word, stemming is a simpler, albeit cruder, method. For example, stemming transforms "studies", "studying", and "studied" into "stud", and "running" and "runner" into "run". This process helps group different word forms, improving consistency in text data. One popular stemming algorithm is the Porter Stemmer [76], which systematically chops off suffixes based on specific linguistic rules, making it effective for English text. While this approach is less precise than lemmatization (which would convert better to good based on context), it is computationally efficient and widely used for tasks that benefit from simpler, faster preprocessing.

2.1.3 Build the Model

Once we have a clean representation of the input document as a token sequence, we are ready to apply ML to solve the task we are interested in. Early approaches in NLP (such as probabilistic language models and the Naive Bayes) were based on the estimation of the tokens co-occurancies. When humans read text, they don't perceive it as a sequence of isolated tokens. Instead, they interpret each word in relation to others, extracting meaning from the text as a cohesive whole. Therefore, any effort to distill text into meaningful data must similarly account for complex grammatical structures and the intricate relationships among words [45].

[2] An example in Italian is the noun "amico" (meaning friend), which changes based on gender and number. It appears as "amico" for a male friend (singular), "amica" for a female friend (singular), "amici" for male or mixed-gender groups (plural), and "amiche" for female groups (plural). Lemmatizing these variations to the base form "amico" allows for consistent recognition of all references to friend, regardless of gender or number, facilitating a more accurate analysis of social connections in various contexts.

With the rise of the Deep Learning era, the focus of NLP shifted to answering the question "How do we represent the meaning of a text?". This allows to decompose almost all the NLP tasks in two subtask (as we have already highlighted in Figure 1):

1. transform the text into a fixed-size representation which captures its meaning;
2. learn a readout model (e.g. a classifier) that solve the task starting from the vectorial representation of the text.

This approach is often denoted as *representation learning* since we move most of the effort in learning a meaningful representation of the input data. Clearly, learning a representation for text is very challenging: even the definition of the meaning of a text is ambiguous and it depends on the task we are solving. However, there are a lot of characteristics that are task-independent: for example, the lexical properties of words, or the relation among words in a sentence. If we succeed to obtain a representation that can capture this task-agnostic characteristics, we can use it as a good starting point to solve any task we are interested in.

Nowadays, to solve a NLP task, we heavily leverage *pre-trained* models, i.e. model that have been trained on a large amount of data to build good text representation. Then, we use our data (which can be relative small) to train only the readout model. For example, in order to solve a text classification model, we could employ (1) a pre-trained Transformer model such as BERT to obtain a good text representation, and (2) a Random Forest to predict the correct label starting from the BERT representation.

In this chapter, we mainly focus on the models that can be used to produce textual representations. However, before going into these details, we breifly introduce some models that can be used as readout to solve NLP tasks.

Readout Models aim to solve the NLP task starting from a textual representation (usually a fixed size vector, as we will show later in this chapter). For the sake of simplicity, we consider a sentiment classification task: given a text as input, we would like to predict the sentiment (negative, neutral, positive). Given a pretrained model that maps the input text into a fixed-size vector of feature, we can employ any classification model as a readout (e.g. Random Forest, Support Vector Machine, Logistic Regression, etc...). To train the readout model, we employ a dataset of pairs $\{(\boldsymbol{h}_i, y_i)\}$, where $\boldsymbol{h}_i$ is the representation of the i-th input text obtained from the pretrained model and y_i is the sentiment that we use as supervision. In this scenario, the pretrained model is used as a fixed function that map the text into a vector space. This approach is often sub-optimal because the representation is not optimized for the task we are considering. For example, it could be that positive and negative adjective can have similar representations even if they encode opposite

sentiments. To mitigate this drawback, it is possible to train the representation model jointly with the readout. This approach is usually denoted as *fine-tuning*. The idea is to move a little bit the parameter of the pretrained model in such a way the classifier can better distinguish between the output labels. To apply the fine-tuning, we need to choose a readout model that can propagate back to the representation model information about the errors: common choices are the Multi-Layer Perceptron or the Logistic Regression.

2.2 Words Representation

A simple way to represent a word is by considering it as an index in a vocabulary list V. Usually, the vocabulary V contains only the words that appear at least once in the corpus we are considering. By using this representation we are just considering words as lemma, i.e. a sequence of characters; indeed, a word is more than that since it carries (at least) one meaning.

For instance, some words share similar meanings (e.g., *car* and *vehicle*), while others are opposites, such as *bright* and *dark*. Additionally, certain words carry positive connotations (*success*), whereas others convey negative connotations (*failure*).

Vector semantics has become the standard method in NLP for representing word meaning, allowing us to model various aspects of semantics effectively. The foundations of this approach trace back to the 1950s, when two major ideas emerged. First, Osgood's 1957 [71] concept proposed representing a word's connotation as a point in three-dimensional space. Second, linguists like Joos (1950), Harris (1954)[50], and Firth (1957)[44] introduced the notion of defining a word's meaning based on its distribution in language use. According to this distributional hypothesis, words that frequently appear in similar contexts or with similar neighboring words tend to share similar meanings [52].

Vector semantics aims to represent a word as a point within a multidimensional semantic space, constructed based on the distribution of neighboring words. These vectors, known as *embeddings*, serve as mappings from one structure (word meanings) to another (the vector space). While the term *embedding* generally refers to any word representation, it is sometimes reserved specifically for dense vector models like Word2Vec (see Section 2.2.2); in this work, we refer to embedding as any vector representation of words (both sparse and dense)..

The main advantage of building a vector semantic is to have all the tools from vector spaces to manipulate the concepts. For instance, the similarity between two words can be quantified by using a metric that measures the closeness of their embeddings. The most widely used metric for this purpose is the cosine similarity,

which calculates the cosine of the angle between the vectors representing the words:

$$\text{cosine}(\boldsymbol{v}, \boldsymbol{w}) = \frac{\boldsymbol{v}^\top \boldsymbol{w}}{|\boldsymbol{v}||\boldsymbol{w}|}, \tag{1}$$

where $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^{|V|}$ are the vector representations of two words.

The cosine similarity ranges from 1 for vectors aligned in the same direction, through 0 for orthogonal vectors, down to -1 for vectors pointing in opposite directions [52]. A "good" vector semantics should be able to capture the relations between words by the meaning of their representation: two words that are synonyms should be mapped to embeddings that have a cosine similarity near 1; On the contrary, antonyms should be mapped to embeddings with a cosine similarity near -1; a cosine similarity of 0 indicates that the word are not related (as two orthogonal vectors).

In the next sections, we deepen three different vector semantics which are based on the distributional assumption. The first one is based on counting how many times a word appears near another one. Then we move to a neural approach which allow to obtain a *static* dense vectorial representation of words. Finally, we move to the state-of-the-art approach based on Transformers architecture where words are represented by *contextual* dense vectors.

2.2.1 Words as frequency vectors

The easiest way to represent the distribution of the context of a word is by computing the empirical distribution on a corpus. The context could be the document, or more commonly, it is a window around the word (for example, of 4 words to the left and 4 words to the right). Let w be a word, its representation is a vector $x_w \in \mathbb{N}^{|V|}$ such that its i-th value indicates how many times the i-th word in the vocabulary appears in the context of w in all the documents in the corpus. We denote this representation as *frequency embeddings*.

For example, we report in Table 1 the co-occurrence matrix in the Wikipedia corpus. Such a matrix has size $|V| \times |V|$ and the value in the i-th row and j-th cell indicates how many times the j-th word in the vocabulary appears in the context of the i-th word. For example, the word *computer* appears two times in the context of *cherry*, while the word *pie* appears 442. Thus, each row of the matrix represents a vector representation of a different word in the corpus. By observing the reported values, it is clear that the word *cherry* is similar to the word *strawberry* since they have a similar context. On the contrary, *cherry* and *computer* appear in completely different contexts; thus, they have different meanings.

The frequency-based representation does not take into account that there are words that appear more frequently in a language. For example, it is very likely that the word "and" appears in the context of almost all the other english words:

	...	computer	data	results	pie	sugar	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
cherry	...	2	8	9	442	25	...
strawberry	...	0	0	1	60	19	...
digital	...	1670	1683	85	5	4	...
information	...	3325	3982	378	5	13	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 1: Co-occurrence vectors for four words in the Wikipedia corpus [52].

thus, the count of how many times the word "and" appears in the context of w does not bring any information about the w. A common approach to overcome this limitation is the introduction a weighting function which allows us to put more emphasis on some words. Positivie Pointwise Mutual Information (PPMI) [32] has been introduced for this purpose and it is defined as:

$$\text{PPMI}(w, c) = \max\left(\log_2 \frac{P(w, c)}{P(w)P(c)}, 0\right), \tag{2}$$

where w and c are two words, and $P(w, c), P(w), P(c)$ are probability distribution that can be estimated by frequencies. The denominatore of PPMI allow us to take into account what is the probability that c appears in the context of w by chance, e.g. only because c is very frequent word in the considered language.

While the frequency embeddings can capture some aspects of the words, we are not taking advantage of the whole vector space. For example, the cosine similarity between frequency embeddings cannot be negative (it ranges from 0 to 1); thus, we cannot represent antonyms with vectors pointing in opposite directions (i.e. with a negative cosine similarity) as we would expect. Another key limitation of this vector semantics is the size of the embeddings since it scales with the vocabulary size $|V|$. Usually, $|V|$ can take values of the order 10k-100k; moreover, this value depends on the corpus considered during the counting. Despite their size, the frequency embeddings are sparse, i.e. most of their entries are equal to zero. This sparse vector representation may fail to capture the similarity between two words that have synonyms in their context. For example, the embedding of a word with "teacher" as a neighbor will differ from the embedding of a word that has "instructor" as a neighbor, as the dimensions representing synonyms like "teacher" and "instructor" remain distinct, despite their related meanings.

2.2.2 Words as fixed neural embeddings

With the advent of the Deep Learning era, seminal work such as [13, 14, 35] returned the focus to modelling words by means of neural language models. These works tackled the problem of building neural language models[3] by introducing the *self-supervision*. The revolutionary intuition is that we train a classifier that predicts the next word without any hand-labelled supervision signal: the supervision is given by the text itself. The self-supervision paved the way for the birth of neural models that produce *dense* vector representations of words, such as the word2vec library [65].

Word2vec embeddings are static embeddings, meaning that the method learns one fixed embedding for each word in the vocabulary [65]. The intuition of word2vec is that instead of counting how often each word w occurs near, say, *apricot*, we instead train a classifier on a binary prediction task: "Is word w likely to show up near *apricot*?" [52]. At first, we assign two random vectors to each word: one represents the word as a context, and the other as a target; we denote by w_i and c_i, the target and the context representation, respectively, of the i-th word in the vocabulary. Our goal is to compute how likely is that a word appears in the context of another target word. The intuition behind word2vec is to model the probability of a word j appearing near a target word i based on vector similarity: word j is more likely to occur close to the target i if its context vector c_j closely resembles the target embedding w_i. To measure the similarity between these dense vectors, word2vec uses their dot product, which is an un-normalised version of the cosine similarity:

$$P(+ \mid w_i, c_j) = \text{sigmoid}(c_j^\top w_i), \tag{3}$$

$$P(- \mid w_i, c_j) = 1 - P(+ \mid w_i, c_j). \tag{4}$$

To learn these dense representations for all the words in the vocabulary, we need a dataset. The dataset is obtained by decomposing the raw-text in a corpus into a set of positive and negative samples. A positive (negative) sample (w_i, c_j) indicates that the word j appears (does not appear) in the context of i in the corpus we are considering. It it worth highlighting that, even if we are solving a supervised task, we do not need of external supervision to determine if a couple of words is a positive or negative sample. Indeed, this information is already in the text; this is why this approach is called *self-supervised* learning.

At the end of the training procedure, the embedding of a word i is obtained by summing its context and target vector, w_i and c_i. The word embeddings obtained in this way are not specific to solving a particular NLP task. On the contrary, they

[3]A neural langua model is a neural network that learned to predict the next word from prior words.

capture words' meaning that should be useful for any downstream task. For this reason, we usually talk about *pre-trained* word embeddings.

Numerous extensions of word2vec have been developed to enhance its capabilities. For instance, fastText [18] tackles the challenge of representing unknown words —those appearing in a test corpus but absent from the training corpus— by using a subword model. In this approach, each word is represented by itself plus a "bag" of its character n-grams. For instance, the word "penguin" could be represented as the sequence <penguin> along with its character 3-grams: <pe, pen, eng, ngu, gui, uin, in>. An embedding is learned for each of these n-grams, and the word "penguin" is then represented by the sum of all its n-gram embeddings. This approach allows unknown words to be represented solely by the sum of their constituent n-grams. FastText is an open-source library [4], which includes pretrained embeddings for 157 languages.

Another widely used static embedding model is GloVe [72], short for Global Vectors, as it captures global statistical information from the corpus. GloVe operates by calculating ratios of probabilities from the word-word co-occurrence matrix, combining the strengths of count-based models like PPMI (see Section 2.2.1) while also capturing the linear relationships that methods like word2vec leverage.

Interestingly, these kinds of dense embeddings have an elegant mathematical relationship with sparse embeddings such as PPMI; they can be understood as implicitly optimizing a shifted version of a PPMI matrix [57]. Another popular properties of these embeddings is their ability to capture relational meanings by means of the parallelogram model [80]. For instance, given the analogy *apple : tree :: grape : ?*, where *apple* is to *tree* as *grape* is to ?, the model predicts the answer (*vine*) as the nearest word to the point $\vec{tree} - \vec{apple} + \vec{grape}$ (see Figure 4).

In Figure 3 we report some example of relations in the GloVe Model.The operation $\vec{king} - \vec{man} + \vec{woman}$ produces a vector close to $\vec{queen}$, while $\vec{Paris} - \vec{France} + \vec{Italy}$ results in a vector near $\vec{Rome}$. This suggests that the embedding model is capturing representations of relationships such as *MALE-FEMALE*, *CAPITAL-CITY*, or even *COMPARATIVE-SUPERLATIVE*.

However, there are certain limitations. For instance, in word2vec or GloVe embedding spaces, the closest vector identified by the parallelogram method is often not b^* but one of the three input words or their morphological variants (e.g., for the analogy *cherry:red :: potato:x*, the output might return *potato* or *potatoes* instead of the expected *brown*). To address this, input words and their variants must be explicitly excluded from the results.

Moreover, while embedding spaces perform effectively for analogies involving frequent words, small vector distances, and certain relationships (such as countries

[4]The library is available at FastText.

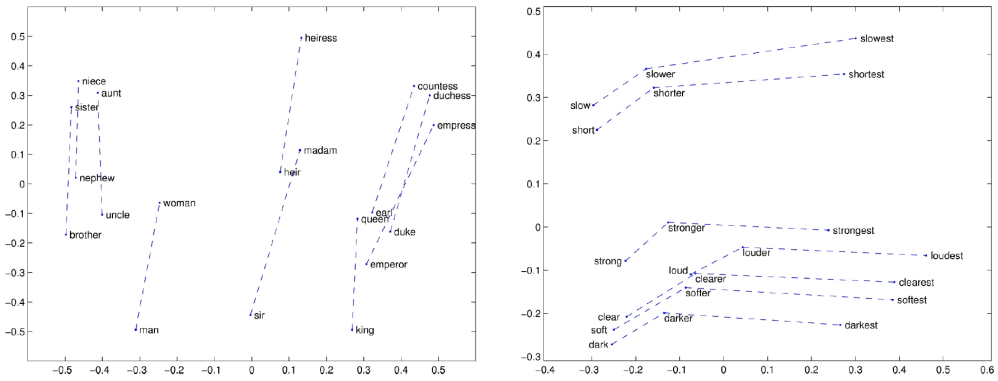


Figure 3: Relational properties of GloVe vectors in a two-dimensional projection.(I) $\vec{king} - \vec{man} + \vec{woman}$ is close to $\vec{queen}$. (II) The offsets appear to capture patterns in comparative and superlative morphology. [72]

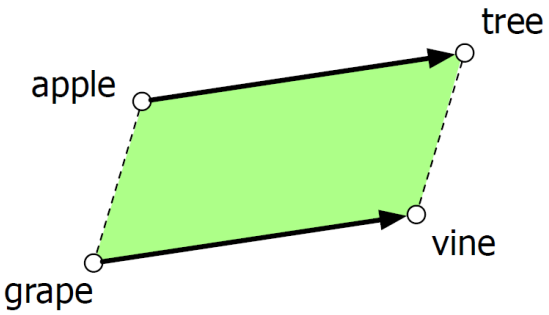


Figure 4: The parallelogram model for analogy problems.

with capitals or verbs/nouns with their inflected forms), the parallelogram approach often fails for other types of relationships [83, 43]. [74] further argue that the parallelogram method may be too simplistic to fully capture the cognitive complexity involved in human analogy formation.

Nevertheless, the main limitation of static neural embeddings, such as Word2vec, GloVe, and FastText, is their inability to adapt based on context, assigning each word a single, fixed representation. This one-to-one mapping overlooks the nuanced meanings a word can carry in different contexts, often resulting in inaccuracies. For instance, consider the word "Apple" in the sentences, "Apple Inc performed well this year" and "Apple fruits are exported to various countries." In the first sentence, "Apple" denotes the technology company, while in the second, it refers to the fruit. Static embeddings, however, assign the same vector to "Apple" in both contexts, disregarding these distinct meanings [66]. This limitation emphasizes the need for context-aware embeddings to accurately capture varied word meanings, especially in applications where semantic precision is crucial.

2.2.3 Words as contextual-aware embeddings

In 2018, researchers at the Allen Institute for Artificial Intelligence introduced ELMo (Embeddings from Language Models) [73], an advanced word encoder that generates contextual embeddings using a deep bidirectional language model (biLM) pre-trained on a large corpus of text. Unlike traditional static embeddings, ELMo produces dynamic, context-sensitive representations by taking into account the surrounding text for each instance of a word. This means that the embedding for a word like "bank" will vary depending on its usage in phrases like "river bank" versus "bank account," addressing the limitations of previous encoders that produce a single, fixed embedding regardless of context. Additionally, ELMo incorporates character-derived embeddings to handle out-of-vocabulary words, drawing on morphological patterns to improve representation quality, especially for unfamiliar or complex tokens [66].

Nowadays, state-of-the-art contextual word embeddings are obtained from bidirectional transformer models. As for the static ones, these architectures are trained by self-supervision. In the next paragraphs, we introduce the most widely-used version of such models called BERT, which stands for Bidirectional Encoder Representations from Transformers [40].

BERT Architecture. The goal is to map a sequence of input vectors $(x_1, \ldots, x_N)$ to a sequence of output vectors $(z_1, \ldots, z_N)$ of the same length. Here, the input consists of a sequence of words, and the output is a sequence of contextual word embeddings. The idea is that, through multiple layers of Transformers [89], progressively richer contextualized representations of each word's meaning are developed.

At each layer, to compute the representation of a word w, we combine its representation from the previous layer with information from neighboring words' representations. This approach, known as *self-attention*, is the central innovation in Transformer models, enabling the model to capture context dynamically for each word in relation to its surrounding text.

Self-attention is derived from the *attention* mechanism, which aims to selectively choose from an input set of concepts, the most relevant ones for a specific query. Formally, let the input set of concepts we can retrieve be a set of couples of d-th dimensional vectors $\{(\boldsymbol{k}_1, \boldsymbol{v}_1), \ldots, (\boldsymbol{k}_N, \boldsymbol{v}_N)\}$, and the vector $\boldsymbol{q}_i \in \mathbb{R}^d$ the query, the attention mechanism computes the vector answer $\boldsymbol{z} \in \mathbb{R}^d$ as:

$$\boldsymbol{z} = \sum_{i=1}^{N} \alpha_i \boldsymbol{v}_i, \qquad \alpha_i = \sigma\left(\frac{\boldsymbol{q}^\top \boldsymbol{k}_i}{\sqrt{d}}\right), \tag{5}$$

where σ is the sigmoid function, and $\alpha_i \in [0, 1]$, $\forall i \in \{1, N\}$, are denoted as *attention weights*. Thus, the answer $\boldsymbol{z}$ is obtained as the weighted average of all the values in the input set. The weight α_i measures how much the i-th concept is relevant for the query: the relevance is proportional to the similarity between the query vector $\boldsymbol{q}$ and the key vector $\boldsymbol{k}_i$.

In self-attention, we do not explicitly have either the query or the set of concepts. Instead, we build them for each element of the input sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$:

$$\boldsymbol{q}_i = W^Q \boldsymbol{x}_i; \quad \boldsymbol{k}_i = W^K \boldsymbol{x}_i; \quad \boldsymbol{v}_i = W^V \boldsymbol{x}_i, \tag{6}$$

where $\boldsymbol{q}_i, \boldsymbol{k}_i, \boldsymbol{v}_i$ are the projection of the input vector x_i into its specific role as a *key*, *query*, or *value*, respectively. The projection is obtained through the weight matrices W^Q, W^K, and W^V. In this case, the vector result of the query $\boldsymbol{q}_i$ is:

$$z_i = \sum_{j=1}^{N} \alpha_{ij} \boldsymbol{v}_j, \qquad \alpha_{ij} = \sigma\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{d}}\right) \tag{7}$$

The vectors $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ represent the output of the attention layer. The attention weights α_{ij} can be interpreted as a measure of how much the j-th word is important to determine the meaning of the i-th word. It is worth highlighting that the approach described in Equation (7) is called **bidirectional** since we are considering both words on the left and on the right of the i-th word to determine its representation. On the contrary, transformer-based language models consider only words on the left (i.e. $\alpha_{ij} = 0$ if $j > i$) to mimic a generative auto-regressive process. This other approach is usually referred to as *masked attention* since we are masking the contribution of the words on the right.

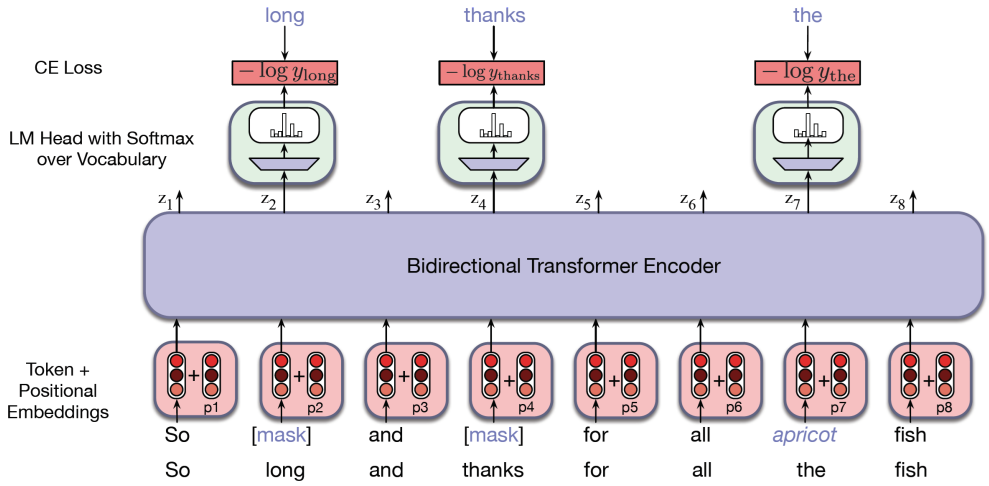


Figure 5: Example of the BERT training [52]

Usually, the transformer layer applies the self-attention multiple times (with different weights). This is called *multi-head* attention. The computation can be carried out in parallel, taking full advantage of the modern hardware such GPUs. Then, all the attention results are concatenated and linearly transformed to obtain the output of the layer. The cost to pay for such expressiveness is that the computational complexity scales with the square of the sequence length since we have to compute all the attention weight α_{ij}.

Training of BERT is performed by self-supervision. However, the task is different from the one considered by word2vec. Instead of predicting the probability that two words appear near each other, in BERT we use solve a task called Masked Language Modeling (MLM) [40]. Masked Language Modeling (MLM) utilizes unannotated text from a large corpus, where a random sample of tokens from each training sequence is selected for prediction tasks. Each chosen token is processed in one of three ways [52]:

- it is replaced with the special vocabulary token named `[MASK]`;
- it is replaced with another token from the vocabulary, randomly sampled based on token unigram probabilities;
- it is left unchanged .

Figure 5 shows an example of training step. The input sentence "So long and thanks for all the fish" is first tokenized with a subword model, then transformed into vector representations via an embedding matrix E and combined with positional

embeddings. To expedite training, the embedding matrix is often initialized with pre-trained static word embeddings. After passing the input through the transformer layers, the model generates vector representations for all the input tokens, but only some of them are used for the training. In the example, we have masked the words "long" and "thanks", while we have replaced the word "the" with "apricot". Thus, we use only the vector representation associated with these tokens (i.e. z_2, z_4, z_7) during the training. In particular, we use them to produce a probability distribution over the vocabulary by using the softmax function[5]; by minimisng the cross-entropy loss, BERT learns to predict the correct word in each position.

2.3 Text Representation

The meaning of a text, whether it is a sentence, a paragraph, or an entire document, depends on the meaning of the words in it and how they relate to each other. In this section, we deepen the approaches that are mainly used to represent text. Without any surprise, the goal is to obtain a vectorial (possibly dense) representation of the whole text we are considering.

The first approach we discuss represents a document by simply counting how many times a word appear in it. Then, we move to neural approaches that are able to process text as a sequence of token. Finally, we discuss how we can employ current transformer based architecture to obtain a representation of the text.

2.3.1 Bag of Words

In a Bag-of-Words (BoW) representation, text is converted into an unordered collection of words, disregarding word position and order. For instance, in the example in Figure 6, rather than preserving the sequence in phrases like "Alright, the Answer to the Great Question of Life, the Universe and Everything.." and "..is Forty-two, said Deep Thought, with infinite majesty and calm," we simply record that the word *the* appears 3 times, while word like *and* appears 2 times or the words like *calm*, *deep* appear once across the text. This approach captures word frequency while omitting syntax and structural details.

[5]The softmax function is a generalisation of the logistic function over K dimensions. Let $\boldsymbol{x} \in \mathbb{R}^K$, the output of the softmax is:

$$\boldsymbol{z}_i = \text{softmax}(\boldsymbol{x})_i = \frac{\exp^{x_i}}{\sum_{j=1}^{K} \exp^{x_j}}.$$

The output vector $\boldsymbol{z}$ lies on $K-1$-simplex and thus represent the parameters of a categorical distribution wiht K possible outcome. In fact, softmax is used to solve classification task with more than two classes.

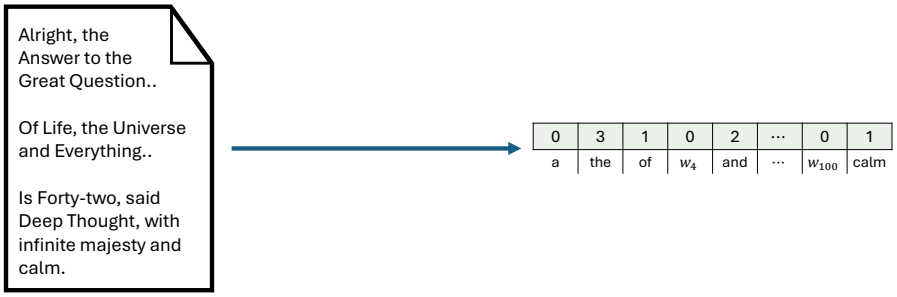


Figure 6: Representation by Bag-of-Words

More formally, the BoW representation of a text d is a vector $x_d \in \mathbb{R}^{|V|}$ obtained by:

$$x_{d,v} = \sum_n \mathbb{I}(w_{d,n} = v),$$

where the function $\mathbb{I}(p)$ is the indicator function which returns the value 1 if and only if the predicate p is true.

The BoW representation is strictly related to the Naive Bayes, a probabilistic classifier that predicts the most likely class for a given input (a document in this case) by applying Bayes' rule. The classifier estimates the class $\hat{c}$ that maximizes the posterior probability $P(c \mid d)$, where c is a class and d is the document. This is transformed using Bayes' rule into $P(d \mid c)P(c)$, which simplifies classification by ignoring $P(d)$ since it remains constant across all classes.

Naive Bayes makes two simplifying assumptions:

1. **Bag-of-Words assumption**: The order of words doesn't matter, so we only consider word frequencies and ignore their positions.

2. **Conditional independence assumption**: The features (e.g., words) are conditionally independent given the class, allowing the joint probability $P(w_1, \ldots, w_N \mid c)$ to be expressed as the product of individual probabilities $P(w_1 \mid c)P(w_2 \mid c) \ldots P(w_n \mid c)$.

The final equation for Naive Bayes classification is:

$$\hat{c}_{NB} = \arg\max_{c \in C} P(c) \prod_{i=1}^{N} P(w_i \mid c), \tag{8}$$

which is usually computed in the log space[6] to avoid numerical errors

The BoW representation is strictly related to the frequency embedding of words introduced in Section 2.2.1, and they share the same limitations. First, BoW representation results in a large number of columns, as the vocabulary size $|V|$ can reach tens of thousands, even for small corpora. Second, the vector x_d is sparse, with most vocabulary terms absent from any given document. This means $x_{d,v} = 0$ for nearly all $v \in \{1, \ldots, |V|\}$[6].

The NB highlights another limitation of Bow: the order of the words is completely ignored. To mitigate this aspect, we can count *n-grams*[7]. This representation is sometimes denoted as Bag-of-n-grams. However, Bag-of-n-grams are even more sparse than BoW since the number of n-grams scales with $O(|V|^n)$.

2.3.2 Recursive Neural Networks

Recurrent Neural Networks (RNNs) are extensions of simple Multilayer Perceptron[8] suitable for analyzing sequential data rather than fixed-size vector inputs. The main idea of RNNs is to recursively process the input sequence by applying the following computation at each time-step t:

$$h_0 = \mathbf{0}, \tag{9}$$

$$h_t = f_\theta(h_{t-1}, x_t), \tag{10}$$

where $(x_1, \ldots, x_T)$ is the input sequence (where $x_t \in \mathbb{R}^m$), and $(h_1, \ldots, h_T)$ is the sequence of the *hidden vectors* (where $h_t \in \mathbb{R}^d$). The role of the hidden vector h_t is to store all the useful information contained in the sequence up to the t-th

[6]By applying the logarithm, we obtain:

$$\hat{c}_{NB} = \arg\max_{c \in C} \log P(c) + \sum_{i=1}^{N} \log P(w_i \mid c).$$

This makes Naive Bayes a linear classifier, as it uses a linear combination of input features.

[7]An n-gram is a sequence of n adjacent symbols in particular order. For example, given the input sentence *"the sky is blue"*, the bi-grams ($n = 2$) are: $\{(\perp, \textit{The}), (\textit{The}, \textit{sky}), (\textit{sky}, \textit{is}), (\textit{is}, \textit{blue}), (\textit{blue}, \perp)\}$, where $\perp$ is a symbol to denote the start and the end of the sentence.

[8]A Multilayer Perceptron (MLP) refers to a modern feedforward artificial neural network composed of fully connected neurons with nonlinear activation functions, arranged in at least three layers. It is particularly notable for its ability to classify data that is not linearly separable.

time step. The state-transition function f_θ is realized through a single-layer neural network with parameters $\theta = \{W, Q\}$:

$$h_t = \sigma(Wx_t + Qh_{t-1} + b), \tag{11}$$

where $\sigma(\cdot)$ is a non-linear activation[9]. The parameter matrices W and Q are the same at each time step, by highlighting the recursive nature of the process.

In practice, training RNNs for tasks that require information from far back in the sequence is challenging [52]. Although RNNs have access to the entire

[9]In neural networks, three popular non-linear functions $f()$ are commonly used: the **sigmoid**, the **tanh**, and the **rectified linear unit (ReLU)**. The sigmoid function is:

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$

The sigmoid has several advantages. It maps the output into the range $(0, 1)$, which helps squash outliers toward 0 or 1. Moreover, it is differentiable, which is beneficial for learning. Let's walk through an example to build intuition. Suppose we have a neural unit with the following weight vector and bias:

$$w = [0.2, 0.3, 0.9], \quad b = 0.5$$

And suppose the input vector is:

$$x = [0.5, 0.6, 0.1]$$

The resulting output y would be:

$$\begin{aligned} y = \sigma(w^T x + b) &= \frac{1}{1 + e^{-(0.5\times0.2+0.6\times0.3+0.1\times0.9+0.5)}} \\ &= \frac{1}{1 + e^{-0.87}} = 0.70 \end{aligned}$$

In practice, the sigmoid is not commonly used as an activation function. A similar but almost always superior alternative is the **tanh**, which ranges from -1 to $+1$:

$$y = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

The simplest activation function, and perhaps the most widely used, is the **rectified linear unit (ReLU)**. The ReLU function is defined as:

$$y = \text{ReLU}(z) = \max(0, z)$$

These activation functions have different properties that make them useful for different applications or network architectures. For example, the **tanh** function has the advantage of being smoothly differentiable and squashing outlier values toward the mean. On the other hand, the **ReLU** function is close to linear, which brings some favorable properties.

One issue with both the sigmoid and tanh functions is that very high values of z saturate the output, resulting in y values close to 1. This saturation leads to derivatives close to 0, which can cause problems during learning. As described in the section, training networks involves propagating an error signal backwards through the network by multiplying gradients (partial derivatives) across layers. Gradients close to 0 can cause the error signal to diminish, a phenomenon known as the **vanishing gradient problem**.

In contrast, rectifiers like ReLU do not suffer from this issue since the derivative of ReLU for large z is 1, rather than close to 0.

preceding sequence, the information encoded in their hidden states often remains fairly localized, more relevant to recent parts of the input and recent decisions. However, long-distance information is essential for many language applications. For example, consider the sentence: *The books recommended by the teacher were out of stock.*

Assigning a high probability to *were* following *teacher* is straightforward, as *teacher* provides strong local context for singular agreement. However, assigning an appropriate probability to *were* is more difficult—not only because the plural *books* is relatively distant, but also because the singular noun *teacher* appears closer in the context. Ideally, a network should retain the distant information about the plural *books* until it is needed, while accurately processing the intervening parts of the sequence.

One reason RNNs struggle to carry critical information forward is that the hidden layers—and the weights that determine their values—are tasked with dual roles: they must both provide information useful for the current decision and simultaneously update and retain information necessary for future decisions.

A second challenge in training RNNs is the need to backpropagate the error signal through time. Since the hidden layer at time t influences the loss at the subsequent time step, it participates in every calculation along the sequence. Consequently, during the backward pass, hidden layers undergo repeated multiplications based on the sequence length. This often leads to gradients diminishing toward zero, known as the **vanishing gradients** problem [52].

Long Short-Term Memory Networks (LSTMs) are popular recursive models which address the vanishing gradient problem by introducing a more complex definition of the state-transition fucntion f_θ [51]. The key idea is to divide the challenge of context management (i.e. which information should I store in h_t)into two parts: removing outdated information and adding relevant information likely to be needed for future decisions. Rather than hard-coding a fixed strategy, LSTMs address this by introducing an explicit context layer in addition to the usual recurrent hidden layer, along with specialized units equipped with **gates**. These gates dynamically regulate the flow of information into and out of each unit, enabling adaptive context management by learning which information to keep or discard. The gates achieve this by applying additional weights sequentially to the input, previous hidden layer, and context layers, allowing the model to adjust in real-time based on the sequence data [46].

Gates in LSTMs follow a common design pattern: each gate comprises a feedforward layer, followed by a sigmoid activation function, and then an element-wise multiplication with the layer it is modulating. The choice of the sigmoid

function is intentional, as it drives outputs toward 0 or 1. When combined with element-wise multiplication, this structure acts like a binary mask: values aligned with 1 in the mask pass through almost unchanged, while those aligned with values near 0 are effectively removed.

The first gate, the **forget gate**, is responsible for delete information in the context that is no longer needed. It computes a weighted sum of the previous hidden state and the current input, which is then passed through a sigmoid function to produce the mask. This mask is then multiplied element-wise with the context vector, selectively removing outdated information. This element-wise multiplication, denoted by the operator $\odot$ (also known as the Hadamard product), produces a vector of the same dimension as the inputs, with each element i being the product of element i in the two input vectors:

$$f_t = \sigma(U_f h_{t-1} + W_f x_t) \tag{12}$$

$$k_t = c_{t-1} \odot f_t \tag{13}$$

This mechanism enables the LSTM to dynamically manage the flow of information, selectively retaining relevant details while discarding what is no longer useful.

The next step involves determining the specific information to extract from the previous hidden state and current inputs. This process follows the core computation used across recurrent networks:

$$g_t = \tanh(U_g h_{t-1} + W_g x_t) \tag{14}$$

Next, we create the mask for the **add gate** to determine which information should be added to the current context.

$$i_t = \sigma(U_i h_{t-1} + W_i x_t) \tag{15}$$

$$j_t = g_t \odot i_t \tag{16}$$

After that we combine this with the modified context vector to produce the updated context vector.

$$c_t = j_t + k_t \tag{17}$$

The final gate is the **output gate**, which determines what information is needed for the current hidden state, as opposed to what should be retained for future decisions:

$$o_t = \sigma(U_o h_{t-1} + W_o x_t) h_t = o_t \odot \tanh(c_t) \tag{18}$$

LSTMs have higher computational and memory requirements than traditional RNNs. Variants such as the Gated Recurrent Unit (GRU) [29] offer simplified architectures with reduced computational overhead.

2.3.3 RNN in NLP

The combination of (possibly deep) RNNs, especially LSTM architecture, with dense pre-trained word embeddings such as wrod2vec, was the state-of-the-art approach before the Transformers era to process raw text in almost all NLP tasks, from sentiment analysis to machine translation. Let $d = \{w_1, \ldots, w_N\}$ be a tokenized raw-text, the first step to apply RNN is to build the input sequence of vectors $(x_1, \ldots, , x_N$. Here the static word embeddings come into play: the vector x_t is the word embeddings of the token w_t. Note that here we do not aim to learn new word representations. On the contrary, we leverage the word embeddings pre-trained on a different (usually big) corpus. We hope these embeddings have captured most of the words' meaning that then are processed by the RNN to generate a representation of the whole text. There are different ways to generate a vectorial text representation by using RNNs, according to the task we are considering.

For example, in sentence classification tasks such as sentiment analysis, the last hidden state h_N is used as a text embedding. This is not surprising since, by definition, the last hidden state should somehow contain all the useful information to solve the task. During the training, the RNN should learn which information should be propagated (or not) in the last hidden state. While in theory this is feasible, in practice it is quite difficult due to the vanishing gradient. Even if we use LSTMs, we need to truncate text (i.e. sequences) that are too long. This method takes the name of Truncate BPTT (TBPTT): when we truncate at the token i, the information of the error afterwards the token i does not flow anymore to the tokens backwards. A possible solution is to capture more context in RNN is to process the input sequence along both forward (from left to right) and backward (from right to left). This approach takes the name of *bidirectional* RNN: a different set of parameters are used for each direction; then, hidden state of a token is obtained by combining the hidden representation of the forward and backward RNN.

In more complex tasks, using only the last hidden state to represent the input text can be an information bottleneck. For example, in Machine Translation, the standard architecture is the encoder-decoder: the *encoder* aims to build a representation of the input text in the source language, while the *decoder* aims to generate a corresponding text in a target language. Clearly, the generation process of the decoder must be conditioned on the representation generated by the encoder. If we use an RNN as the encoder, we can take its final hidden state to represent the input text. However, this representation creates a bottleneck, as it must encapsulate the entire meaning of the source text—being the only information the decoder has about the source. The *attention mechanism*, previously introduced in 2.2.3, addresses this bottleneck by allowing the decoder to access all the encoder's hidden states. Attention constructs a fixed-length vector that dynamically represents the input text by taking a weighted sum of all encoder hidden states. These weights

emphasize (or “attend to”) the part of the source text most relevant to the token the decoder is currently generating. This way, attention replaces the static context vector with a dynamic one derived from the encoder's hidden states, adapting with each token during decoding.

2.3.4 Transformers

The advent of the Transformer architecture revolutionized the NLP field. Unlike RNNs, Transformers are capable of processing entire sequences of text in parallel, utilizing mechanisms like self-attention to capture long-range dependencies. This shift has paved the way for developing state-of-the-art language models like GPT and BERT, marking the beginning of the "Transformers era" in NLP.

We have already discussed the Transformers architecture in Section 2.2.3, where we have deepened its role to obtain contextual word embeddings. Here, we focus on how Transformers can be used to obtain a vectorial representation of a whole text.

Unlike RNNs, in Transformers no a vector is responsible for the whole sequence. Let $(x_1, \ldots, x_N)$ the input sequence, Transformers-based models produce an output sequence $(h_1, \ldots, h_N)$, where h_t is a contextual representation of the token x_t. The most straightforward way to condense the contextual word embeddings into a single vector is by taking their vector. However, in tasks like sentence classification, the common approach is to add a special token `[CLS]` which acts as the embedding for the entire text. This token is added to the vocabulary and prepended to the beginning of all input sequences during both pre-training and encoding. The output vector associated with the `[CLS]` token in the model's final layer represents the entire input sequence, serving as the input to the final classifier.

As in RNN, in more complex tasks such as machine translation, we would like to compress the information of the source text into a fixed vector. Instead, we would like to have a dynamic representation which different for each token in decoding. This mechanism takes the name of *cross-attention* since we would like to compute, for each output token, a different representation of the input text. In practice, cross-attention is structured similarly to the multi-head attention discussed in Section 2.2.3. However, while the queries are derived from the previous layer of the decoder, the keys and values are sourced from the encoder's output.

The main limitation of Transformers is that we need to fix in advance the sequence length. While this is negligible in many tasks, it is not when we deal with very long documents. In Section 3, we discuss this aspect in detail.

3 Models for Long Texts

This section introduces how long text can be managed within NLP by starting with a classification of documents across varying levels of scope. A text can be analyzed at four distinct levels, each with a unique focus on document structure:

- **Document level:** At this level, the algorithm classifies an entire document, identifying the relevant categories based on the document's overall content and structure [15].
- **Paragraph level:** The algorithm narrows its analysis to single paragraphs, categorizing each as a unit within the document. This allows for a more nuanced understanding of sections within a larger context [36, 2].
- **Sentence level:** Focusing further, the algorithm identifies relevant categories for individual sentences, interpreting each sentence as a coherent unit within paragraphs and capturing more localized meanings [20].
- **Sub-sentence level:** At this granular level, the algorithm extracts categories for specific sub-expressions within a sentence, allowing for fine-grained analysis of clauses or phrases that may carry unique significance within the larger context.

By employing these hierarchical levels, NLP techniques can process and manage long texts effectively, selecting the appropriate scope for each analysis and accommodating the complex structure of extensive documents.

Our objective is to analyze the document at the document level, yet a critical challenge remains: determining whether the document in question falls into the category of long or short text. This distinction fundamentally shapes our approach, as long documents demand techniques that account for complex, extended dependencies, whereas short texts typically require less intricate processing.

3.0.1 What is a (Long) Document?

A document is a core concept in information science, typically defined as a recorded piece of information intended for communication or reference. It can take multiple forms—such as written texts, images, or digital files—and serves various purposes in conveying knowledge. The definition of a document can shift based on context, often highlighting its role in information retrieval and management [24].

This broad definition encompasses a variety of document types—tweets, reviews, research papers, and financial reports—and it not helps distinguish between short and long text. We can say that short texts typically lack the depth and long-range dependencies found in more extensive documents and contain fewer words.

Conversely, long texts feature intricate structures and dependencies that extend across the entire content.

Despite this differentiation, no universal definition exists for "long text" or "long document." The literature often categorizes specific genres of text as "long" or "short" based on practical usage rather than strict criteria. For example, short texts like tweets and brief reviews are consistently categorized as "short," while research papers, financial documents, and books are usually classified as "long" [56]. However, certain types, such as news articles or research abstracts, are often arbitrarily assigned to either category.

To further illustrate these distinctions, [56] argue that short texts are generally brief, structured, and fact-driven, with limited long-range dependencies or causal links. These characteristics make them more manageable for NLP models, which highlights the complexity of defining long documents through purely quantitative metrics. Even long texts may vary in processing complexity, depending on structural and thematic features.

For the purpose of this section, long documents include types such as research papers and financial document (10-K or Business Plan). Research papers, frequently ranging between 3,000 and 10,000 words [17], are widely employed in NLP tasks [33, 97, 12]. Financial documents, which range from 70,000 to 150,000 words (only one section of 10-K is in average 13,000 words [94]), and large-scale document datasets such as arXiv [33], are also relevant in this category.

Based on these references, a “long document” consists of documents with an average word count of at least 2,000 words, extending up to 150,000 words or more. Additionally, each document should maintain semantic continuity, where context and entity relationships can span across multiple paragraphs and sections. NLP algorithms designed for long documents must therefore accommodate both the relatively large average size and instances that significantly exceed this average. Moreover, they must capture and analyze dependencies and contexts that persist across extensive sections of text.

Datasets tailored for long-text analysis are essential for exploring the complexities inherent in processing extensive documents. Resources like LongForm, *LongFin* [64], and 20NewsGroups provide documents with substantial average word counts, which allow models to capture nuanced relationships and thematic elements across extended content. These datasets are frequently employed in long document analysis due to their capacity to support the study of long-span dependencies and intricate content structures.

3.0.2 Key Challenges in Long-Text Processing

Algorithms for long document analysis face a variety of challenges due to the inherent complexity of their input. While some of these challenges are common to

general text processing, others are unique to handling lengthy documents. Here is an overview of the primary obstacles:

- *Curse of Dimensionality*: As vocabulary size expands, the number of possible word combinations during testing grows exponentially, while the training dataset remains limited [14]. This results in an increased likelihood of encountering unseen sentence patterns, particularly in long documents, which complicates generalization and reduces inference accuracy.

- *Polysemy*: Words can carry multiple meanings depending on its context. For instance, the word "dish" might refer to a plate or a food course. The models need to learn to differentiate a word's intended meaning by analyzing both the local context and the broader global context

- *Homonymy*: Homonyms, which share spelling or pronunciation but differ entirely in meaning, add another layer of complexity. Unlike polysemous words, homonyms lack any inherent semantic connection (e.g. "bark" might refer to the sound a dog makes or the outer layer of a tree).

- *Figurative Language*: Sarcasm, irony, and metaphor present challenges for NLP systems, as the intended meaning often differs from the literal one [53]. This issue is more pronounced in long texts where figurative language may extend across sections.

- *Unstructured Text*: Long documents, such as novels or academic papers, often exhibit complex, evolving structures, making it difficult to maintain coherent meaning across large sections [92]. This challenge is particularly acute for financial documents, such as Business Plan, due to non-standardized and complex structure (as we will discuss in4).

- *Limited Foreign Language Data*: There is limited availability of training content for less commonly used languages beyond the most popular ones (e.g., English, French, Spanish). Since both word embedding networks and task-specific NLP models are usually trained on a single language at a time, the lack of extensive datasets in languages like Italian can lead to suboptimal embeddings and, subsequently, reduced performance in downstream tasks. For example, Italian, despite being a widely spoken language, has comparatively fewer comprehensive datasets, making it challenging to achieve the same accuracy in NLP tasks as in English.

Table 2: Comparing of Transformer-based models by model complexity and input tokens limit.

Name	Year	#Parameters	Input size (#tokens)
BERT [40]	2019	110M	512
BART [58]	2019	140M	1024
GPT [77]	2018	120M	–
GPT-2 [78]	2018	1.5B	1024
GPT-3 [23]	2020	175B	4096
PaLM [31]	2022	540B	3072
LLaMA [86]	2023	7B to 65B	2048
GPT-4 [70]	2023	Unknown	128,000 (as written)
Gemini [85]	2023	1.8B	30,720 (as written)

3.1 Common Mitigation Strategies for Transofmers

Processing long-text is particularly challenging for transformer-based architectures. These models such as BERT are often constrained by token sequence length limits (usually 512 or 1024 tokens, see Table 2). The limit on the number of tokens is necessary to reduce the time complexity (and the number of parameters) of transformers: due to the self-attention mechanism (see Section 2.2.3), they require a number of operations that scales quadratically with the input size.

A naive approach could be aggressively selecting tokens during inference; however, this result in performance degradation on longer texts since the model may inadequately capture dependencies that span large sections of text[37].

Thus, ad-hoc mechanisms are required when we need to apply transformers architectures on long text. These strategies can be divided into four macro strands:

Feature Selection approaches reduce the length of the input text by discarding elements in it. The aim is to discard as much text as possible while minimizing the information loss.

Hierarchical aggregation solutions split the input document into segments that adhere to the Transformer's input size. Then, a separate Deep Neural Network (DNN) is employed to aggregate the representation obtained for each segment.

Sparse Attention proposes to modify the self-attention mechanism in the Transformers to reduce the computational cost of computing the attention weights from quadratic to linear (w.r.t the input size).

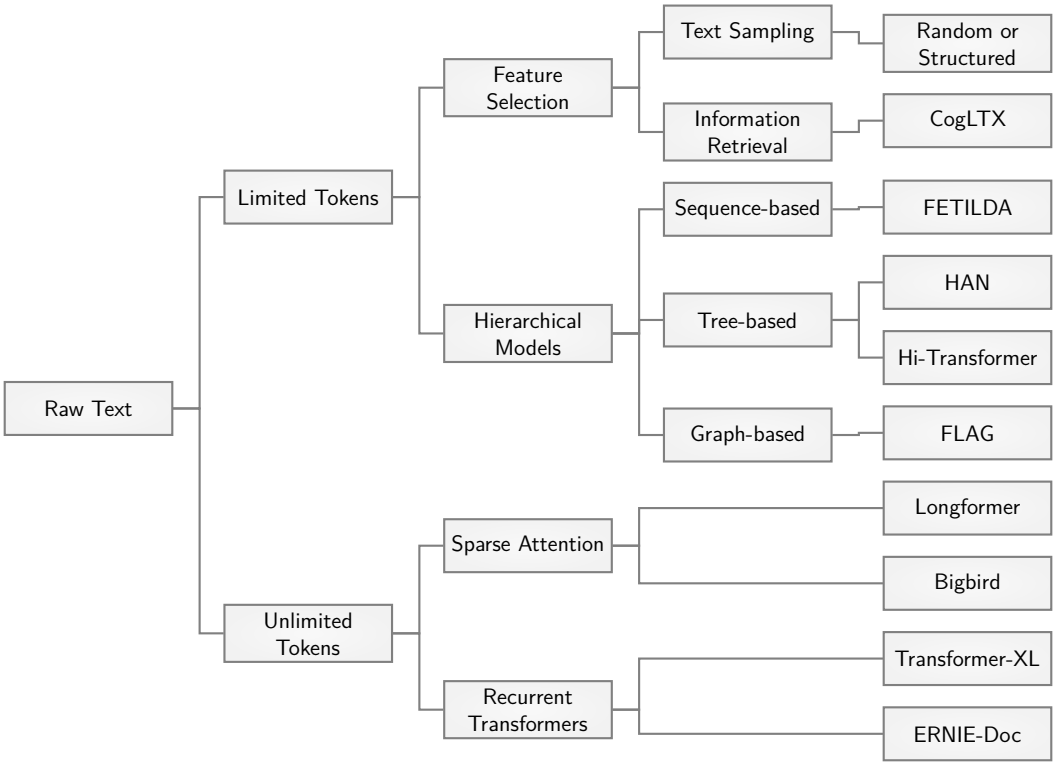


Figure 7: Taxonomy of Transformer architectures for long text.

Recurrent Transformers incorporate recurrence to go beyond the fixed input size. The input text is still segmented into chunks, but each chunks is processed by considering also a global context.

The first two approaches adhere to the Transformer token input limit, while the last two modify the Transformer architecture to support much larger token limits, as shown in the figure 7.

3.1.1 Feature Selection

The aim of feature selection approaches is to reduce the length of the input document by discarding part of it. We briefly discuss two example of feature selection methods, one suitable for document with a fixed and known structure, and one more general based on information retrieval algorithms.

Text sampling is the simplest way to reduce the input text length. Instead of using the whole text as input for the Transformers, we use only part of it. If we have some prior knowledge on the input text, we can use the part of the text that is more informative. For example, in a research paper, we could take the abstract

or the introduction to obtain a short-text with (almost) the same meaning. If the input text is not structured, we can sample the text by following a combination of user-defined rules and random chance.

Information Retrieval approaches to feature selection allow the selection of the blocks of the input text that hold the largest semantic significance; these can then be fed to a computationally demanding DNN such as BERT. For example, the method proposed in Zhu et al. [99] relies on local query-block pre-ranking; the segmentation of the document into blocks is accomplished using the CogLTX [41]. CogLTX is a dynamic programming segmentation approach that prioritizes segments containing punctuation marks like periods (" . ") and exclamation points (" ! "). This method enforces an upper limit on tokens per block, creating a manageable and consistent structure. Each block is ranked based on its Retrieval Status Value (RSV), calculated either by the BM25 ranking function—a commonly used metric in search engines [60]—or by cosine similarity.

3.1.2 Hierarchical Models

Hierarchical architectures aim to address the large input size of long texts by appropriately building upon original Transformers, rather than modifying them. As before, these architectures are task-agnostic and generic.For our purposes, we concentrate on models that divide the input document into chunks matching the Transformer's input size. The network (e.g., BERT) then generates embeddings for each segment. A separate DNN combines these segment embeddings to predict the document label. We divide these approach according to how the segments are combined:

Sequence-based approaches treat the embeddings generated by the Transformer for each chunk as a sequence. Thus, they are processed by employing a DNN for sequences such as LSTM.

Tree-based approaches combine the chunks in a hierarchical way by gradually obtaining an embedding of the whole text, e.g. "sentence" embeddings are combined to obtain "paragraph" embeddings that are in turn combined to obtain "section" embeddings, etc.

Graph-based approaches combine chunk representations in a graph-like structure. Here the aggregation must be carried out by employing graph-specific DNN.

FETILDA [94] proposes and implements a deep learning framework that processes long documents by splitting them into chunks, and utilizing pre-trained transformer models to generate vector representations for these chunks. This is

followed by attention-based sequence modelling to extract valuable document-level features.

Particularly, the generation of the chunk embeddings is carried out by using advanced language models such as BERT and FinBERT. The focus is on generating contextual embeddings for each chunk, which are then pooled to create effective chunk embedding vectors. To combine these vectors, they employs a Bi-LSTM (see the 2.3.2) model with attention to obtain a document representation. Thanks to the attention, it is possible to to determine which chunks contribute more significantly to the overall document representation.

During the training, the document feature vector is passed through fully connected layers to predict a target numeric variable.

Hierarchical Attention Network (HAN) [25] is a tree-based model which splits the input document into paragraphs that are truncated to fit into BERT. The paragraph embeddings obtained are combined by the succeeding Hierarchical Attention Network (HAN) into a single output. The end result is an increased ability to handle long texts by exploiting the natural partitioning of documents into paragraphs.

The initial attempts to create a Hierarchical Transformer architecture were built upon standard BERT. In this approach, the input document is divided into paragraphs, each truncated to fit within BERT's input limit. BERT then generates independent embeddings for each paragraph, which are subsequently combined by a Hierarchical Attention Network (HAN) to produce a single output. This architecture enhances the model's capability to handle long texts by leveraging the natural structure of documents divided into paragraphs.

A similar method was later applied using models such as BERT and RoBERTa [61], demonstrating competitive results on long-text [25].

Another variation, presented in[55], also relies on BERT, but replaces the HAN with a LSTM architecture. This configuration achieved the best performance among hierarchical models. .

Hi-Transformer [93] is another tree-based hierarchical model which employs a hierarchy of Transforemrs to produce a fixed-size document-level representation. Initially, a Transformer functions as a "Sentence Encoder" (SE), aggregating and transforming individual word embeddings from the document into sentence-level embeddings. After each sentence has been processed by the SE, these sentence embeddings are ordered using positional embeddings and passed to a subsequent Transformer known as the "Document Encoder" (DE). The DE outputs a context-aware document embedding, which is then fed back to the next SE, along with the original sentence embeddings. This iterative process ensures that sentence

embeddings are enriched with global document-level context. This sequence of encoding is repeated through multiple stacked layers, culminating in a pooling layer that generates the final document embedding. According to [93], stacking two such layers is often sufficient to surpass Sparse Attention Transformers in long-text tasks due to the effective capture of global context. Importantly, as the SE complexity grows linearly with the number of sentences, computational demands remain comparable to sparse attention methods. Interestingly, the Hi-Transformer's performance has been observed to improve with longer documents, as it captures increasingly relevant contextual information.

Hierarchical BERT-based dynamic fusion [69] is a graph-based method proposed by for handling extremely long documents. It involves integrating hierarchical information through nodes that capture nested relationships and semantic structures within the text. It utilizes contextual information from pretrained Transformer models like BERT, along with a novel dynamic fusion technique. This dynamic fusion merges outputs from earlier stages with an external model to improve overall performance.

HeterGraphLongSum [75] aims to learn a heterogeneous graph structure for long text summarization. Specifically, we model an input document with three types of nodes: word, sentence, and passage nodes, as a heterogeneous graph, using a Graph Attention Network (GAT) [90] to capture information relations among these nodes. The creation of a heterogeneous graph that includes word, sentence, and passage nodes. The passages are formed by concatenating a fixed number of sentences, which is a hyperparameter tuned during validation. The model focuses on specific edge types, such as word-to-sentence and word-to-passage, while avoiding redundant connections like passage-to-word. Sentence nodes are updated based on their neighboring word and passage nodes, allowing for effective information propagation. This mechanism enhances the contextual understanding of sentences by leveraging relationships among different types of nodes. After that the sentence selector layer, which extracts document representation from the hidden states of passage nodes.

FLAG [98], which stands for Financial Long Document Classification via AMR-based Graph, utilizes the Abstract Meaning Representation (AMR) to create document-level graphs from sentence-level AMR graphs, thereby enhancing semantic understanding [8]. It employs specialized transformer-based word embeddings tailored for the financial domain to initialize nodes in the constructed graphs. The approach consists of three main phases: (1) the document is split into sentences, and an AMR graph is built for each of them; (2) all the graph sentences are

combined into a single document graph by introducing sentence and document nodes; (3) a Graph Neural Network (GNN) model [7], specifically GATv2 [21], is applied to generate document representations for classification tasks.

3.1.3 Sparse Attention Transformers

Due to the quadratic computational complexity of Transformer-based models and their extensive application in recent years, numerous efforts have been made to reduce these computational costs. These approaches seek to attain linear complexity with respect to input size by limiting attention to a small, fixed-size window of neighboring tokens around each input token, rather than considering all n tokens of the sequence within each self-attention head. However, this optimization inherently reduces the model's capacity to capture long-term dependencies, which is essential for effectively processing lengthy texts.

Sparse Transformer [28] seeks to reduce the asymptotic complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n\sqrt{n})$. This reduction is accomplished by using sparse factorizations on each self-attention matrix calculated by the Transformer during inference (see Section 2.2.3). These matrices represent the current attention weights of each input element with respect to the others in the sequence and are separately computed for each self-attention head at every layer. Reducing the computational and memory costs of this operation is crucial, and the sparse attention mechanism accomplishes this by replacing full attention with multiple smaller, optimized attention operations that collectively approximate the original process. In essence, each token attends only to a subset of the input sequence. Further optimizations, including efficient sparse attention kernels and refined weight initialization, enhance the architecture's efficiency, allowing it to manage longer sequential inputs. The Sparse Transformer is a flexible model suitable for a range of tasks involving lengthy input sequences, from image compression to document analysis.

Longformer [12] is another approach that achieves linear asymptotic complexity and is tailored specifically for natural language processing (NLP) tasks involving long documents. Its self-attention mechanism can be seamlessly integrated into other Transformer models and leverages a dilated sliding window approach, which is an enhanced version of the classical sliding window method for local attention [81]. In typical use cases, the Longformer offers substantial improvements in computational efficiency for processing long sequential inputs.

Each token's attention window includes D surrounding tokens, with $\frac{D}{2}$ tokens on either side. The core idea is that a token's semantic context is largely influenced by its immediate neighbors, achieving a computational complexity of $\mathcal{O}(N \times D)$. This mechanism bears similarity to the local receptive fields found in Convolutional

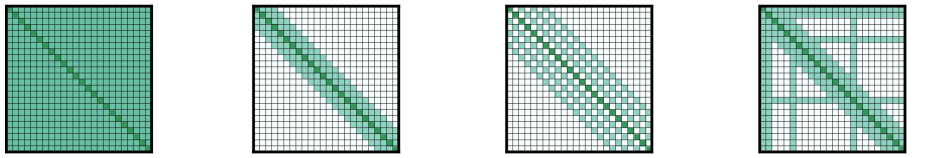


Figure 8: Comparison of the standard full self-attention pattern with the specialized attention configurations utilized in the Longformer model [12].

Neural Networks (CNNs), where stacking l Longformer layers progressively increases the receptive field, allowing tokens far away from the query to be attended to in the deeper layers. As in CNNs, it is also possible to specify a dilation factor K at each step which controls the spacing between adjacent window positions. By introducing "gaps" in the attention pattern, we can increase the receptive field without adding new layers.

Since windowed and dilated attentions alone may not capture an optimal sequence representation, specific positions within the full self-attention matrix are designated as global context. These global tokens can both attend to and be attended by the entire sequence, but their number is kept limited to maintain a computational complexity of $\mathcal{O}(n)$.

The [CLS] token is a typical candidate for selection as a global token, as its representation is intended to summarize the entire sequence. Thanks to these innovations, the Longformer has been successfully applied to sequences of up to 4096 tokens (in comparison to BERT's limit of 512 tokens).

Figure 8 illustrates a comparison of full, sliding, dilated sliding, and global-dilated sliding window self-attention patterns.

Bigbird [97] can be viewed as an extension of the Longformer, designed to handle significantly longer input sequences by reducing the complexity of self-attention from quadratic to linear. Like the Longformer, it utilizes a sliding window mechanism, as in many NLP tasks, the semantic meaning of a token is largely determined by its neighboring tokens. However, instead of using dilated sliding windows, BigBird introduces a random token selection method to capture distant tokens and incorporate broader context. This approach complements the global attention and sliding window attention mechanisms borrowed from the Longformer. It has been mathematically proven that this modified sparse attention mechanism preserves several key theoretical properties of full attention, such as universal approximation and Turing completeness. However, BigBird often requires more layers to maintain accuracy comparable to traditional full attention models.

3.1.4 Recurrent Transformers

A complementary direction in state-of-the-art research is to enable the Transformer to learn and capture long-term dependencies by employing recurrent mechanisms. The long document is split into a set of independent, consecutive segments which are then processed recursively.

Transformer-XL [38] introduces recurrence into the Transformer model, thereby incorporating the RNN concept of retaining hidden states across segments. These hidden states are passed within the Transformer from one segment to the next, effectively transferring previously established context and enabling the model to capture long-term dependencies across sequences. Unlike the recurrence mechanism in RNNs/LSTMs, where each layer's state is passed to the next time step within the same layer, Transformer-XL shifts the transmitted states upwards through the layers. For instance, at layer l, we take into account two output sequence obtained from the previous layer: one is computed on the current segments, while the other is computed on the previous segment. By stacking more layer, we can increase the receptive field of the network.

This mechanism ensures a computational cost of $\mathcal{O}(n \times l)$, where l denotes the number of layers, leading to significantly faster inference compared to the baseline Transformer. This efficiency stems from reusing representations computed for earlier segments, as opposed to recomputing them for each new segment. Additionally, Transformer-XL supports varying input sequence sizes between the training and inference stages. Overall, Transformer-XL achieves competitive performance on extremely long document analysis tasks while being far more efficient than the baseline Transformer.

ERNIE-Doc [42] build it upon Transformer-XL defines an enhanced recurrence mechanism which goes downward between layers; at layer l, we take into account (1) the output sequence obtained from the previous layer on the current segment, and (2) the output sequence obtained from the *same* layer on the previous segment. This mechanism allows a much larger receptive field as long as the segments from the input document are twice fed as input. These two phases are called *skim* and *retrospective*. Figure 9 shows a comparisono between Transformer-XL and ERNIE-Doc.

Additionally, a novel document-level task for self-supervised pretraining is introduced, termed the *Segment Reordering Objective* (SRO). This involves randomly splitting a long document into m segments, shuffling them, and allowing the Transformer to reorganize them to learn their interrelations. After pretraining on both the masked language model (MLM) task and the SRO, ERNIE-Doc demonstrated

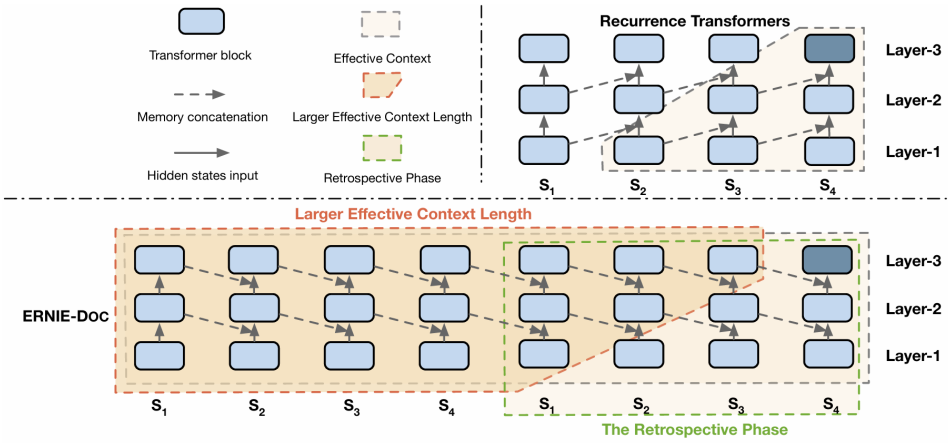


Figure 9: A visual comparison between the recurrent mechanism employed by Transformer-XL and ERNIE-Doc [42]

strong performance across several long document analysis datasets for various tasks such as classification, question answering, and key-phrase extraction.

Sparse attention, hierarchical, and recurrent approaches currently dominate the field of long document analysis using Transformers. Yet, several challenges persist. In Sparse Attention Transformers, often it was experienced accuracy limitations, as they rely on a limited number of global tokens to capture the entire document context. Moreover, Hierarchical Transformers are particularly prone to context fragmentation. Segment embeddings in these models may be derived with insufficient local context or without access to the full document context [37, 93].

There remains no consensus on the best-performing approach or an empirical rule dictating the optimal architecture for specific tasks. Each method presents unique strengths and limitations, which may vary depending on the domain. Consequently, up to now, the literature recommends that users select architectures based on their specific application requirements. However, at a broader level, hybrid algorithms and recurrence-based methods appear promising for advancing long-text analysis.

4 Unstructured Data and Natural Language Processing in Financial Documents

In recent years, there has been growing interest within the research community in extracting insights from unstructured data, including financial documents, through

methods such as document analysis, sentiment analysis, patent classification, and behavioral economics [62, 67, 59, 47, 49, 27]. Large volumes of unstructured data are generated from a wide variety of financial documents such as annual reports, Securities and Exchange Commission (SEC) filings, earnings call transcripts, financial statements, business plan, press releases, and even auditor reports [10]. These documents contain valuable information in the form of text, figures, and tables that organizations and researchers alike seek to analyze for gaining insights into corporate performance, market sentiment, and economic trends.

In response to these challenges, NLP, is emerging as a key solution to automate the processing and analysis of vast quantities of financial documents [10].

In the literature, many authors have applied NLP on annual reports. However, while most textual analysis techniques typically treat an entire document as a single data instance, here the focus is usually in detecting the changes of these documents through time. In NLP, the task of measuring the similarity between two documents is often referred to as Semantic Textual Similarity (STS).

For instance, [3] examine how variations in managerial involvement and effort across firms influence the creation and presentation of public disclosures, such as 10-K filings and conference calls. They highlight differences in who writes these disclosures and the impact on their structure, style, and tone, offering insights into how internal processes shape public communications. Moreover, Cohen et al. apply a similarity measure to quarterly and annual filings of U.S. corporations from 1995-2014, demonstrating that changes in 10-K filings can predict future earnings, profitability, news announcements, and even firm-level bankruptcies [34]. On the contrary, Brown et al. study the opposite direction, i.e. which are the economic changes in a firm that lead to modification in the Management Discussion and Analysis (MD&A) disclosures submitted to the SEC[22].

In all cases, the documents are represented using a BoW representation (see Section 2.2.1) that clearly is not sufficient to capture all the meanings concealed in documents.

If we move our attention to static neural embeddings (see Section 2.2.2), [79] employs word2vec, GloVe and fasttext embeddings to build document representations which are then used to detect changes in 10-k documents. Similarly, Li et al. create a corporate culture dictionary using word embedding models and analyze earnings call transcripts to score five cultural values—innovation, integrity, quality, respect, and teamwork [59]. They show that innovative culture extends beyond traditional measures like R&D and patents. Moreover they find strong correlations between corporate culture and business outcomes, such as operational efficiency, risk-taking, earnings management, executive compensation, and firm value, with the culture-performance link being more pronounced during downturns. Additionally, corporate culture is influenced by major events like mergers and

acquisitions.

Recent literature explores advanced NLP models, specifically transformers, to manage long financial documents effectively. For instance,Xia et al. introduced the FETILDA framework to extract information from specific sections of 10-K reports, such as the risk factors and management's discussion sections[94]. FETILDA addresses lengthy text by segmenting it into chunks, generating embeddings for each with advanced language models (e.g., BERT, Longformer), and then combining these with a neural network (Bi-LSTM) for a final, weighted document representation. This framework was tested on predicting key performance indicators (KPIs) for U.S. banks and stock volatility.

In a related approach, Zaki et al. proposed a graph-based model for creating document-level embeddings for long financial texts [98]. Their method builds document-level graphs from sentence-level graphs, applying deep learning to predict target data from extensive documents. Experiments on quarterly earnings call transcripts and S&P 1500 companies demonstrated the model's effectiveness, with results showing improved prediction of stock price movement trends over various time horizons.

In another study proposed by Sautner et al., it has been shown that firms' climate change exposures can effectively predict key outcomes related to the net-zero transition, such as job creation in disruptive green technologies and green patenting, and reveal valuable information that is reflected in options and equity markets [82]. However, they do no employ a vectorial representation of the input text; instead, the rely on a probabilistic method is developed to extract desired keyword from them.

Other studies investigate the relationship among topics and performance of firms. [11] develop a measure of innovation by applying Latent Dirichlet Allocation (LDA), a topic modeling technique, to the textual analysis of analyst reports for S&P 500 firms. Using this text-based approach, they are able to identify innovation-related themes within the reports, providing a robust measure of innovation that can forecast firm performance and growth opportunities for up to four years. Importantly, the value implications of this measure hold just as strongly for innovative firms that do not engage in patenting, demonstrating the model's broad applicability across different types of innovation-driven firms.

Also [26] provide large-scale evidence of anticipatory effects, showing that firms are impacted by proposed regulations long before they are finalized. By analyzing over 43,000 rule proposals from U.S. federal agencies since 1995, the authors develop a firm-level measure of exposure to the regulatory pipeline using machine learning to extract topics. The findings reveal that firms exposed to more proposals tend to increase overhead costs, reduce capital investments, and report lower profits.

Another valuable source of economic insights comes from newspaper articles and

social media. For instance, Aprigliano et al. demonstrate that newspaper articles can effectively forecast economic activity [5]. They develop two high-frequency indices for Italy based on 1.5 million articles from four major newspapers, utilizing an Italian economic dictionary. Forecasting tests show that these indices enhance short-term predictions of macroeconomic aggregates, particularly during recessions, and improve the accuracy of a weekly GDP tracker across various economic cycles.

On the social media front, Angelico et al. [4] use Italian tweets to create real-time measures of consumer inflation expectations. Keywords identify tweets discussing prices and inflation, and daily indicators are generated using Latent Dirichlet Allocation (LDA) combined with a dictionary-based approach, incorporating manually labeled bi-grams and tri-grams. These Twitter-based indicators show strong correlations with monthly survey-based and daily market-based inflation expectations, effectively capturing consumers' forward-looking inflation sentiments.

A summary is provided in the table 3.

Although NLP algorithms have significantly advanced, they still face limitations in fully replicating the complexity and contextual understanding required for in-depth analysis of financial documents.

Paper	**Representation**	**Documents**
Amel-Zadeh et al. [3]	Bag-of-Words	10-K and Conference Calls
Cohen et al. [34]	Bag-of-Words	10-K and 10-Q
Brown et al. [22]	Bag-of-Words	10-K (7 **)
Rawte et al. [79]	Static Embeddings	10-K
Li et al. [59]	Static Embeddings	Earnings Call
Xia et al. [94]	Contextual Embedding	10-K (1/1A ** and 7 *)
Zaki et al. [98]	Contextual Embedding	Quarterly Earnings Call
Sautner et al. [82]	Bag-of-n-grams	Financial Reports
Bellstam et al. [11]	LDA	Analyst Reports
Chang et al. [26]	LDA	Rule Proposals
Aprigliano et al. [5]	Bag-of-n-grams	Newspaper
Angelico et al. [4]	Bag-of-n-grams	Social Media

* The Item 1/1A regards Risk Factors section. ** The Item 7 is about the Management's Discussion and Analysis (MD&A).

Table 3: Summary of Textual Representation and Document Types in Financial Text Analysis.

4.1 Different types of financial documents

Due to the increasing interest in the application of NLP in the finacial context, in this section we discuss various types of financial documents that could be used as input texts. These documents, although differing in structure and purpose, serve as critical tools for investors, stakeholders, and analysts in assessing a company's current status, potential risks, and future performance.

Financial documents, encompassing a broad spectrum of types and formats, are integral to the industry, containing crucial information that influences market trends, investment strategies, and regulatory compliance. However, these documents differ significantly in their structure, content, and the specific challenges they pose for NLP applications. More in deep, we have several types of financial documents:

Annual Reports are comprehensive documents produced by companies to provide shareholders and the public with information on their financial performance and strategies over the past year. These documents are often dense, featuring not only financial statements but also qualitative descriptions, including management discussions, risk assessments, and future outlooks. The structured data, such as financial tables, coexist with unstructured narrative sections, making it a complex source for NLP applications.

Earnings Calls Transcripts are quarterly teleconferences where company executives discuss the financial results with analysts and investors. The transcripts of these calls are rich in qualitative data, reflecting the tone, sentiment, and subtle cues about the company's performance and future expectations. NLP tasks here often focus on sentiment analysis and speech recognition to extract insights from the spoken word, capturing nuances that are critical for investor sentiment.

Regulatory Filings, such as the 10-K and 10-Q forms submitted to the Securities and Exchange Commission (SEC) in the United States, are standardized documents that provide detailed information on a company's financial health, risk factors, and management's discussion. These documents are highly structured and follow strict formats, making them suitable for rule-based and machine learning-based NLP techniques. However, the legal and technical jargon present in these filings poses unique challenges for accurate information extraction and analysis.

News Articles and Analyst Reports are external documents that provide independent assessments and commentary on financial markets, companies, and economic conditions. These sources are highly unstructured, written in natural language with varying styles, and may contain bias or subjective opinions. NLP applications often focus on extracting entities, detecting

sentiment, and assessing the relevance of the information to specific financial contexts.

Social Media and Forums: in recent years, social media platforms and financial forums have become increasingly important sources of real-time financial information and sentiment. The informal and often abbreviated language used in these platforms, along with the rapid pace at which information spreads, presents a distinct set of challenges for NLP. Techniques like sentiment analysis, entity recognition, and trend detection are commonly applied, but the unstructured and noisy nature of the data requires sophisticated preprocessing and filtering techniques.

Business Plan: forward-looking documents that outline a company's strategic objectives, financial projections, and growth plans. Typically prepared by startups or companies seeking investment, they provide a detailed roadmap that encompasses market analysis, operational strategy, and financial forecasts. These plans are instrumental for venture capitalists and investors to evaluate the viability of a business and assess its potential for success. They often contain qualitative descriptions of business strategies, which are complemented by quantitative financial models projecting future revenue and profitability.

Among the various types of financial documents, business plans stand out as a critical resource for entrepreneurs, investors, and financial analysts. These documents provide a comprehensive roadmap of a company's goals, strategies, market analysis, financial forecasts, and operational plans. However, their unstructured nature and varied content present unique challenges for NLP applications.

4.1.1 Business plans

A Business Plan (BP) typically encompass a wide range of information, often structured into several key sections:

1. *Executive Summary*: The executive summary provides a concise overview of the business, its mission, and its objectives. This section is crucial for capturing the attention of investors, making it a prime target for NLP tasks such as summarization and key information extraction.

2. *Market Analysis*: This section details the market in which the business operates, including target demographics, market size, trends, and competitive analysis. NLP can be applied to extract insights on market conditions, competitive positioning, and potential opportunities or threats.

3. *Company Description*: The company description provides information on the business's history, structure, and objectives. NLP techniques can be used to extract entity information, assess the coherence of the business strategy, and identify key stakeholders.

4. *Organization and Management*: This section outlines the company's organizational structure and the qualifications of its management team. NLP can analyze this section to gauge the experience and competence of the leadership team, which is often a critical factor for investors.

5. *Product Line or Services*: Detailed descriptions of the company's products or services, including their unique selling propositions, are provided here. NLP can be used to analyze product descriptions, identify potential differentiators, and assess the competitive advantages presented.

6. *Marketing and Sales Strategy*: This section outlines how the business plans to attract and retain customers. NLP applications can evaluate the feasibility and innovation of these strategies, compare them with industry norms, and even predict their potential success.

7. *Financial Projections*: Financial projections include revenue forecasts, profit and loss statements, and cash flow analysis. While this section is more structured, NLP can be applied to align textual descriptions with numerical data, ensuring consistency and highlighting discrepancies.

8. *Funding Request*: If the business plan is seeking funding, this section will specify the amount of capital needed and the proposed use of funds. NLP can help assess the clarity and persuasiveness of these requests, as well as benchmark them against industry standards.

In a Nutshell. Analyzing financial documents presents several challenges, primarily due to their length and the specialized language used within financial contexts. These documents often exceed the token limits of models like Transformers, even with extended versions such as Longformer and BigBird, which are still often insufficient for their full length.

Furthermore, financial documents, such as business plans and 10-K reports, are complex not only in length but in content. Business plans may contain strategic, abstract language, while 10-K filings mix structured financial data with qualitative assessments of risks and opportunities. Terms in financial documents, like "equity," often carry different meanings depending on the context, highlighting the challenge of creating accurate, context-sensitive representations.

Contextual language models (Encoder models), like BERT and its variants, offer promising solutions by interpreting word meanings based on surrounding text, which is critical in financial documents where words like "risk" or "return" have specific, nuanced implications. However, developing effective document-level representations for such long texts remains a significant challenge.

Another limitation is the generic nature of training corpora used for pre-training most language models, such as general web content or Wikipedia. These sources lack the specialized terminology and context of financial documents, reducing the effectiveness of such models in finance-specific applications. To address this, recent advancements include pre-training models on financial corpora. For example, FinBERT [95], trained on financial news and filings, is more effective for evaluating business plans or analyzing 10-K reports, offering improved handling of domain-specific language. These domain-adapted models can then be further fine-tuned on specialized datasets to enhance performance in tasks like financial forecasting, risk assessment, and market volatility prediction.

Overall, while financial documents provide essential insights for analysis, their unique structures, terminologies, and length present specific challenges for NLP models. Addressing these complexities requires developing advanced language models that can process lengthy texts and adapt to specialized financial contexts.

5 Conclusion

In conclusion, this paper explores the evolution of NLP text representation techniques, from simpler traditional models to complex, architecture-driven approaches, in addressing the unique challenges of analyzing long financial documents. Early techniques, such as Bag-of-Words and word embeddings, laid the groundwork for transforming text data into structured representations, yet they struggled to capture nuanced relationships and long-range dependencies within extensive documents. With the advent of Transformer-based models, including advanced versions like Longformer and BigBird, NLP has made significant strides in handling the complexity of financial documents. However, these models often face limitations in accurately representing long documents due to token constraints, contextual fragmentation, and scalability issues, which are further compounded by the unique requirements of financial language.

Sparse attention mechanisms in Transformers have been applied to manage computational complexity in long-text analysis, though they typically rely on a limited set of global tokens, resulting in accuracy constraints. Similarly, hierarchical Transformer models attempt to divide text into manageable segments but are prone to context fragmentation, as segment embeddings may lack sufficient access to the full document context [37, 93]. Recurrent and hybrid approaches have shown promise in mitigating these issues by enhancing the contextual continuity across document segments, yet a universally optimal architecture for long-text representation remains elusive. Consequently, the literature encourages users to select models tailored to specific applications, as each architecture presents distinct strengths and weaknesses depending on the document structure and analytical requirements.

The analysis of financial documents further complicates the long-text challenge. Financial texts, such as business plans and 10-K filings, are not only lengthy but also dense with specialized, context-sensitive terminology. Financial language often includes terms with varied meanings based on context, demanding models that can interpret nuanced financial discourse accurately. While contextual language models like BERT have advanced this capacity by interpreting words within their surrounding context, they still struggle to fully accommodate the document-level representation needs of complex financial texts. Domain-specific models like FinBERT, pre-trained on financial corpora, provide a partial solution, enhancing performance by adapting to specialized financial language, but they require further fine-tuning for tasks such as risk assessment and market prediction to achieve optimal results.

In summary, while NLP has made notable progress in addressing the challenges posed by long and complex financial documents, significant obstacles remain. The current approaches—sparse attention, hierarchical, and recurrence-based mod-

els—each offer valuable contributions but are yet to fully resolve the demands of long-text financial analysis. Developing models that can seamlessly manage both length and domain-specific language remains a priority for advancing NLP in finance. Future research should focus on refining hybrid and adaptive architectures that combine sparse, hierarchical, and domain-adapted methods to build robust, contextually aware representations for long financial documents, ultimately enhancing the reliability and depth of insights that NLP can deliver in financial contexts.

References

[1] S. Adhikari et al. "Nlp based machine learning approaches for text summarization". In: *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE. 2020, pp. 535–538.

[2] Q. Ai, L. Yang, J. Guo, and W. B. Croft. "Analysis of the paragraph vector model for information retrieval". In: *Proceedings of the 2016 ACM international conference on the theory of information retrieval*. 2016, pp. 133–142.

[3] A. Amel-Zadeh, A. Scherf, and E. F. Soltes. "Creating firm disclosures". In: *Journal of Financial Reporting* 4.2 (2019), pp. 1–31.

[4] C. Angelico, J. Marcucci, M. Miccoli, and F. Quarta. "Can we measure inflation expectations using Twitter?" In: *Journal of Econometrics* 228.2 (2022), pp. 259–277.

[5] V. Aprigliano, S. Emiliozzi, G. Guaitoli, A. Luciani, J. Marcucci, and L. Monteforte. "The power of text-based indicators in forecasting Italian economic activity". In: *International Journal of Forecasting* 39.2 (2023), pp. 791–808.

[6] E. Ash and S. Hansen. "Text algorithms in economics". In: *Annual Review of Economics* 15.1 (2023), pp. 659–688.

[7] D. Bacciu, F. Errica, A. Micheli, and M. Podda. "A gentle introduction to deep learning for graphs". In: *Neural Networks* 129 (2020), pp. 203–221.

[8] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider. "Abstract meaning representation for sembanking". In: *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*. 2013, pp. 178–186.

[9] L. Barbaglia, S. Consoli, and S. Manzan. "Forecasting with economic news". In: *Journal of Business & Economic Statistics* 41.3 (2023), pp. 708–719.

[10] D. Baviskar, S. Ahirrao, V. Potdar, and K. Kotecha. "Efficient automated processing of the unstructured documents using artificial intelligence: A systematic literature review and future directions". In: *IEEE Access* 9 (2021), pp. 72894–72936.

[11] G. Bellstam, S. Bhagat, and J. A. Cookson. "A text-based analysis of corporate innovation". In: *Management Science* 67.7 (2021), pp. 4004–4031.

[12] I. Beltagy, M. E. Peters, and A. Cohan. "Longformer: The long-document transformer". In: *arXiv preprint arXiv:2004.05150* (2020).

[13] Y. Bengio, R. Ducharme, and P. Vincent. “A Neural Probabilistic Language Model”. In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000.

[14] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. “A Neural Probabilistic Language Model”. In: *Journal of Machine Learning Research* 3 (2003), pp. 1137–1155.

[15] S. Bhatia, J. H. Lau, and T. Baldwin. “An automatic approach for document-level topic model evaluation”. In: *arXiv preprint arXiv:1706.05140* (2017).

[16] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O’Reilly Media, Inc.", 2009.

[17] B.-C. Bjork, A. Roos, and M. Lauri. “Scientific journal publishing: yearly volume and open access availability.” In: *Information Research: An International Electronic Journal* 14.1 (2009).

[18] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. “Enriching word vectors with subword information”. In: *Transactions of the association for computational linguistics* 5 (2017), pp. 135–146.

[19] B. Bok, D. Caratelli, D. Giannone, A. M. Sbordone, and A. Tambalotti. “Macroeconomic nowcasting and forecasting with big data”. In: *Annual Review of Economics* 10.1 (2018), pp. 615–643.

[20] V. K. Bongirwar. “A survey on sentence level sentiment analysis”. In: *International Journal of Computer Science Trends and Technology (IJCST)* 3.3 (2015), pp. 110–113.

[21] S. Brody, U. Alon, and E. Yahav. “How attentive are graph attention networks?” In: *arXiv preprint arXiv:2105.14491* (2021).

[22] S. V. Brown and J. W. Tucker. “Large-sample evidence on firms’ year-over-year MD&A modifications”. In: *Journal of Accounting Research* 49.2 (2011), pp. 309–346.

[23] T. B. Brown. “Language models are few-shot learners”. In: *arXiv preprint arXiv:2005.14165* (2020).

[24] M. K. Buckland. “What is a “document”?” In: *Journal of the American society for information science* 48.9 (1997), pp. 804–809.

[25] I. Chalkidis, I. Androutsopoulos, and N. Aletras. “Neural legal judgment prediction in English”. In: *arXiv preprint arXiv:1906.02059* (2019).

[26] S. Chang, J. Kalmenovitz, and A. Lopez-Lira. “Follow the Pipeline: Anticipatory Effects of Proposed Regulations”. In: *Available at SSRN 4360231* (2023).

[27] M. A. Chen, Q. Wu, and B. Yang. "How valuable is FinTech innovation?" In: *The Review of Financial Studies* 32.5 (2019), pp. 2062–2106.

[28] R. Child, S. Gray, A. Radford, and I. Sutskever. "Generating long sequences with sparse transformers". In: *arXiv preprint arXiv:1904.10509* (2019).

[29] 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". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by A. Moschitti, B. Pang, and W. Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

[30] K. Chowdhary. *Fundamentals of Artificial Intelligence*. 2020.

[31] 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". In: *Journal of Machine Learning Research* 24.240 (2023), pp. 1–113.

[32] K. W. Church and P. Hanks. "Word Association Norms, Mutual Information, and Lexicography". In: *27th Annual Meeting of the Association for Computational Linguistics*. Vancouver, British Columbia, Canada: Association for Computational Linguistics, June 1989, pp. 76–83. DOI: 10.3115/981623.981633.

[33] C. B. Clement, M. Bierbaum, K. P. O'Keeffe, and A. A. Alemi. "On the use of arxiv as a dataset". In: *arXiv preprint arXiv:1905.00075* (2019).

[34] L. Cohen, C. Malloy, and Q. Nguyen. "Lazy Prices". In: *The Journal of Finance* 75.3 (2020), pp. 1371–1415.

[35] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. "Natural language processing (almost) from scratch". In: *Journal of machine learning research* 12 (2011), pp. 2493–2537.

[36] A. M. Dai. "Document embedding with paragraph vectors". In: *arXiv preprint arXiv:1507.07998* (2015).

[37] X. Dai, I. Chalkidis, S. Darkner, and D. Elliott. "Revisiting transformer-based models for long document classification". In: *arXiv preprint arXiv:2204.06683* (2022).

[38] Z. Dai. "Transformer-xl: Attentive language models beyond a fixed-length context". In: *arXiv preprint arXiv:1901.02860* (2019).

[39] F. J. Damerau. "A technique for computer detection and correction of spelling errors". In: *Commun. ACM* 7.3 (Mar. 1964), pp. 171–176. DOI: 10.1145/363958.363994.

[40] J. Devlin. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[41] M. Ding, C. Zhou, H. Yang, and J. Tang. “Cogltx: Applying bert to long texts”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 12792–12804.

[42] S. Ding, J. Shang, S. Wang, Y. Sun, H. Tian, H. Wu, and H. Wang. “ERNIE-Doc: A retrospective long-document modeling transformer”. In: *arXiv preprint arXiv:2012.15688* (2020).

[43] K. Ethayarajh, D. Duvenaud, and G. Hirst. “Towards understanding linear word analogies”. In: *arXiv preprint arXiv:1810.04882* (2018).

[44] J. Firth. “A synopsis of linguistic theory 1930-1955”. In: *Studies in Linguistic Analysis, Special Volume/Blackwell* (1957).

[45] M. Gentzkow, B. Kelly, and M. Taddy. “Text as data”. In: *Journal of Economic Literature* 57.3 (2019), pp. 535–574.

[46] F. A. Gers, J. Schmidhuber, and F. Cummins. “Learning to forget: Continual prediction with LSTM”. In: *Neural computation* 12.10 (2000), pp. 2451–2471.

[47] I. Goldstein, C. S. Spatt, and M. Ye. “Big data in finance”. In: *The Review of Financial Studies* 34.7 (2021), pp. 3213–3225.

[48] J. Grimmer, M. E. Roberts, and B. M. Stewart. *Text as data: A new framework for machine learning and the social sciences*. Princeton University Press, 2022.

[49] K. W. Hanley and G. Hoberg. “Dynamic interpretation of emerging risks in the financial sector”. In: *The Review of Financial Studies* 32.12 (2019), pp. 4543–4603.

[50] Z. Harris. *Distributional structure*. 1954.

[51] S. Hochreiter. “Long Short-term Memory”. In: *Neural Computation MIT-Press* (1997).

[52] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released August 20, 2024. 2024.

[53] D. Karamouzas, I. Mademlis, and I. Pitas. “Neural knowledge transfer for sentiment analysis in texts with figurative language”. In: *2022 IEEE 32nd International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE. 2022, pp. 1–6.

[54] M. Kay and M. Roscheisen. “Text-translation alignment”. In: *Computational linguistics* 19.1 (1993), pp. 121–142.

[55] S. I. Khandve, V. K. Wagh, A. D. Wani, I. M. Joshi, and R. B. Joshi. "Hierarchical neural network approaches for long document classification". In: *Proceedings of the 2022 14th International Conference on Machine Learning and Computing*. 2022, pp. 115–119.

[56] W. Kryściński, N. Rajani, D. Agarwal, C. Xiong, and D. Radev. "BOOKSUM: A Collection of Datasets for Long-form Narrative Summarization". In: *Findings of the Association for Computational Linguistics: EMNLP 2022*. 2022, pp. 6536–6558.

[57] O. Levy and Y. Goldberg. "Neural word embedding as implicit matrix factorization". In: *Advances in neural information processing systems* 27 (2014).

[58] M. Lewis. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension". In: *arXiv preprint arXiv:1910.13461* (2019).

[59] K. Li, F. Mai, R. Shen, and X. Yan. "Measuring corporate culture using machine learning". In: *The Review of Financial Studies* 34.7 (2021), pp. 3265–3315.

[60] L. Liu. *Encyclopedia of database systems*. 2009.

[61] Y. Liu. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[62] T. Loughran and B. McDonald. "When is a liability not a liability? Textual analysis, dictionaries, and 10-Ks". In: *The Journal of Finance* 66.1 (2011), pp. 35–65.

[63] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. "The Stanford CoreNLP natural language processing toolkit". In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 2014, pp. 55–60.

[64] A. Masry and A. Hajian. "LongFin: A Multimodal Document Understanding Model for Long Financial Domain Documents". In: *arXiv preprint arXiv:2401.15050* (2024).

[65] T. Mikolov. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[66] K. Mishev, A. Gjorgjevikj, I. Vodenska, L. T. Chitkushev, and D. Trajanov. "Evaluation of sentiment analysis in finance: from lexicons to transformers". In: *IEEE access* 8 (2020), pp. 131662–131682.

[67] A. K. Nassirtoussi, S. Aghabozorgi, T. Y. Wah, and D. C. L. Ngo. "Text mining for market prediction: A systematic review". In: *Expert Systems with Applications* 41.16 (2014), pp. 7653–7670.

[68] F. J. Och and H. Ney. "Discriminative training and maximum entropy models for statistical machine translation". In: *Proceedings of the 40th Annual meeting of the Association for Computational Linguistics*. 2002, pp. 295–302.

[69] A. Onan. "Hierarchical graph-based text classification framework with contextual node embedding and BERT-based dynamic fusion". In: *Journal of king saud university-computer and information sciences* 35.7 (2023), p. 101610.

[70] OpenAI. "Gpt-4 technical report. arxiv 2303.08774". In: *View in Article* 2.5 (2023).

[71] C. Osgood. "The Measurement of Meaning". In: *Universit of Illinois* (1957).

[72] J. Pennington, R. Socher, and C. D. Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[73] M. E. Peters, M. Neumann, L. Zettlemoyer, and W.-t. Yih. "Deep contextualized word representations". In: *https://arxiv.org/pdf/1802.05365* (2018).

[74] J. C. Peterson, D. Chen, and T. L. Griffiths. "Parallelograms revisited: Exploring the limitations of vector space models for simple analogies". In: *Cognition* 205 (2020), p. 104440.

[75] T. A. Phan, N.-D. N. Nguyen, and K.-H. N. Bui. "Hetergraphlongsum: heterogeneous graph neural network with passage aggregation for extractive long document summarization". In: *Proceedings of the 29th International Conference on Computational Linguistics*. 2022, pp. 6248–6258.

[76] M. F. Porter. "An algorithm for suffix stripping". In: *Program* 40 (1997), pp. 211–218.

[77] A. Radford. "Improving language understanding by generative pre-training". In: (2018).

[78] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[79] V. Rawte, A. Gupta, and M. J. Zaki. "A comparative analysis of temporal long text similarity: Application to financial documents". In: *Workshop on Mining Data for Financial Applications*. Springer. 2020, pp. 77–91.

[80] D. E. Rumelhart and A. A. Abrahamson. "A model for analogical reasoning". In: *Cognitive Psychology* 5.1 (1973), pp. 1–28.

[81] H. A. Saeed, H. Wang, M. Peng, A. Hussain, and A. Nawaz. "Online fault monitoring based on deep neural network & sliding window technique". In: *Progress in Nuclear Energy* 121 (2020), p. 103236.

[82] Z. Sautner, L. Van Lent, G. Vilkov, and R. Zhang. "Firm-level climate change exposure". In: *The Journal of Finance* 78.3 (2023), pp. 1449–1498.

[83] N. Schluter. "The word analogy testing caveat". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Volume 2 (Short Papers)*. Association for Computational Linguistics. 2018, pp. 242–246.

[84] H. Schütze, C. D. Manning, and P. Raghavan. *Introduction to information retrieval*. Vol. 39. Cambridge University Press Cambridge, 2008.

[85] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, et al. "Gemini: a family of highly capable multimodal models". In: *arXiv preprint arXiv:2312.11805* (2023).

[86] 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". In: *arXiv preprint arXiv:2302.13971* (2023).

[87] S. Vajjala, B. Majumder, A. Gupta, and H. Surana. *Practical natural language processing: a comprehensive guide to building real-world NLP systems*. O'Reilly Media, 2020.

[88] 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". In: *arXiv preprint arXiv:1609.03499* 12 (2016).

[89] A. Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems* (2017).

[90] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).

[91] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, et al. "Tacotron: Towards end-to-end speech synthesis". In: *arXiv preprint arXiv:1703.10135* (2017).

[92] J. Worsham and J. Kalita. "Genre identification and the compositional effect of genre in literature". In: *Proceedings of the 27th international conference on computational linguistics*. 2018, pp. 1963–1973.

[93] C. Wu, F. Wu, T. Qi, and Y. Huang. "Hi-transformer: Hierarchical interactive transformer for efficient and effective long document modeling". In: *arXiv preprint arXiv:2106.01040* (2021).

[94] B. Xia, V. D. Rawte, M. J. Zaki, A. Gupta, et al. “Fetilda: An effective framework for fin-tuned embeddings for long financial text documents”. In: *arXiv preprint arXiv:2206.06952* (2022).

[95] Y. Yang, M. C. S. Uy, and A. Huang. “Finbert: A pretrained language model for financial communications”. In: *arXiv preprint arXiv:2006.08097* (2020).

[96] D. Yu and L. Deng. *Automatic speech recognition*. Vol. 1. Springer, 2016.

[97] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al. “Big bird: Transformers for longer sequences”. In: *Advances in neural information processing systems* 33 (2020), pp. 17283–17297.

[98] M. J. Zaki, A. Gupta, et al. “FLAG: Financial Long Document Classification via AMR-based GNN”. In: *arXiv preprint arXiv:2410.02024* (2024).

[99] Y. Zhu, H. Yuan, S. Wang, J. Liu, W. Liu, C. Deng, H. Chen, Z. Dou, and J.-R. Wen. “Large language models for information retrieval: A survey”. In: *arXiv preprint arXiv:2308.07107* (2023).