NATURAL LANGUAGE PROCESSING

Francesco A. Fabozzi, PhD
Research Director, Yale International Center for Finance

Introduction

Natural language processing (NLP) has become a foundational technology in financial analysis and investment decision making. As modern financial markets are increasingly shaped by the flow of unstructured textual information—from earnings calls and regulatory filings to news articles and social media—investors and analysts face growing pressure to systematically process, interpret, and act on these data.

At the same time, large language models (LLMs) have ushered in a paradigm shift in how textual data can be analyzed. These models—trained on vast corpora of text—can perform a wide range of tasks with minimal supervision, including question answering, summarization, classification, and entity extraction. Their ability to generalize across tasks and domains has made them particularly attractive for financial applications, where bespoke data and limited training labels often hinder the use of conventional supervised learning approaches.

This chapter explores the evolving role of NLP in finance, with a particular emphasis on the transformative impact of LLMs. I argue that the flexibility, scalability, and adaptability of LLMs have opened up new frontiers for analyzing financial text, enabling both discretionary and systematic investors to extract insights that were previously inaccessible. The integration of these models into financial workflows, however, also introduces new challenges related to trust, evaluation, infrastructure, and regulation. The sections that follow trace the technical evolution of NLP methods, examine the capabilities and limitations of generative models, and highlight the most relevant applications, risks, and future directions for financial professionals.

Evolution of NLP Techniques

Although today's LLMs offer flexible and high-performing solutions for a wide range of text-based tasks, their development has been built upon decades of progress in the field of NLP. Understanding the historical evolution of NLP methods is valuable not only for appreciating how the field arrived at its current capabilities but also for identifying use cases where simpler, more computationally efficient techniques may still be appropriate. In many financial applications, classical NLP tools remain relevant and cost-effective when high predictive accuracy or linguistic nuance is not essential.

Computers cannot naturally interpret language in the same way they handle structured, numerical data. As a result, early NLP efforts focused on converting textual information into numerical representations suitable for modeling. Over time, methods have evolved from treating language as unordered collections of words to more sophisticated approaches capable of capturing syntax, semantics, and even pragmatic meaning.

Rule-Based and Dictionary Approaches

The earliest NLP techniques, dating back to the 1950s, were based on rule-based systems and hand-crafted dictionaries. These approaches involved predefined word lists associated with particular categories, such as sentiment, emotion, or subject matter. For example, a sentiment dictionary might classify "excellent," "profit," or "growth" as positive terms, while "loss," "decline," or "risk" would be labeled negative. An analyst could then quantify the sentiment of a document by counting the number of positive and negative words it contained.

To illustrate, consider a simple example. Suppose a sentence reads, "The company reported strong revenue growth but noted increased supply chain risks." Using a basic sentiment dictionary, this sentence might register both positive (e.g., "strong," "growth") and negative (e.g., "risks") words, with the final classification depending on their relative counts. These methods were intuitive and easy to implement, but they came with substantial limitations:

- *Lack of context:* Dictionary methods ignore the surrounding context of words. For instance, they typically fail to account for negations ("not profitable") or modifiers ("barely profitable"), leading to misclassification.
- *Polysemy and ambiguity:* Words with multiple meanings, such as "interest" (loan interest versus personal interest), are treated uniformly, often introducing noise into analysis.
- *Equal weighting:* All words in the dictionary are treated as equally informative, failing to capture intensity differences—for example, "good" and "great" may both be classified as positive, despite differing in strength.
- *Subjectivity in word selection:* Manually assigning words to categories can reflect human bias and domain insensitivity.

These limitations became particularly apparent in financial applications. General-purpose sentiment dictionaries developed for consumer product reviews or news articles often performed poorly when applied to financial documents. For instance, the word "liability" may be negative in everyday language, but in accounting, it is a neutral technical term. Recognizing this issue, Loughran and McDonald (2011) analyzed common financial texts and found that approximately two-thirds of words classified as negative by standard dictionaries were not actually negative in a financial context. To address this, they introduced the "Loughran–McDonald Master Dictionary w/Sentiment Word Lists," specifically tailored for financial analysis. This development marked a turning point in domain-specific NLP and highlighted the importance of context-aware tools.

Statistical and Count-Based Methods

To address the limitations of rule-based and dictionary approaches, the next phase in NLP involved representing textual data in structured, statistical formats that could be analyzed using traditional machine learning models. This approach centers on the *document–term matrix*, where each row represents a document (such as a news article or earnings call) and each column corresponds to a unique word, or *term*, in the corpus. The entries in this matrix indicate the frequency with which each word appears in each document.

This representation enables words to be assigned weights by a statistical model, rather than relying on predefined dictionary labels. In effect, the model learns the relationship between

words and an outcome variable—such as stock returns or sentiment labels—based on observed co-occurrence patterns, allowing for a more data-driven understanding of language.

To illustrate, consider two documents:

- Document A: "The company reported strong earnings."
- Document B: "The company faced weak earnings."

A document–term matrix would represent these as follows (simplified for clarity):

	Company	Reported	Strong	Earnings	Faced	Weak
Document A	1	1	1	1	0	0
Document B	1	0	0	1	1	1

These *count vectors* can then be used as inputs to standard classifiers, such as logistic regression or naive Bayes, enabling empirical estimation of which words are associated with positive or negative outcomes.

This approach introduces new challenges, however, including dimensionality and sparsity: As the vocabulary size increases—particularly in financial text where domain-specific terminology is abundant—the document–term matrix becomes high-dimensional and sparse. That is, most entries are zero because any given document contains only a small subset of the total vocabulary. This sparsity can reduce model performance and increase computational burden.

To mitigate this drawback, practitioners often apply *text preprocessing* techniques, including the following:

- *Stopword removal:* Common, noninformative words, such as "the," "and," and "of," are removed.
- *Stemming and lemmatization:* Words are reduced to their base or root forms. For example, "running," "ran," and "runs" may all be reduced to "run," improving consistency and reducing dimensionality.
- *TF–IDF weighting:* In raw count-based matrices, common words dominate. To address this problem, *term frequency–inverse document frequency* (TF–IDF) weighting is used. TF–IDF downweights words that appear frequently across all documents (such as "company") and upweights words that are more unique to a specific document, helping highlight discriminative terms.

Despite these refinements, count vector–based methods still have significant limitations:

- *No context or ordering:* Word order is ignored, making it impossible to distinguish between "not profitable" and "profitable." This feature limits the model's ability to capture negation and other contextual modifiers.
- *No semantic relationships:* Each word is treated as an independent token, with no recognition of synonymy or antonymy. For instance, "good" and "great" are no more similar than "good" and "terrible" in this representation.

These drawbacks motivated the development of more-advanced models that can capture contextual and semantic meaning in language, which are discussed in the following sections.

Neural Models and Word Embeddings

The techniques discussed thus far focus on representing text in structured formats that can be used as inputs to traditional machine learning models. These representations—whether based on word counts or term frequency weighting—treat words as independent and do not attempt to capture the underlying relationships among them. Although useful for many tasks, these models fall short in their ability to *understand* language in a deeper, semantic sense.

As machine learning models and computational power advanced, NLP shifted from merely *representing* text to *learning from* text. Rather than manually crafting features or counting word occurrences, models began using neural networks to learn relationships between words directly from large corpora of text. This process is typically done through semisupervised learning: The model is trained on a simple prediction task, such as predicting a missing word based on its surrounding context. In doing so, the model develops an internal representation of word meaning—referred to as *word embeddings*.

A word embedding is a high-dimensional vector that encodes semantic information about a word. Words that appear in similar contexts tend to have similar embeddings. One of the most well-known approaches for generating word embeddings is *Word2Vec*, introduced by Mikolov, Sutskever, Chen, Corrado, and Dean (2013). Word2Vec is trained on such tasks as the following:

- *Continuous bag of words:* Predict a word based on its surrounding context (e.g., given "The company [...] strong earnings," predict "reported").
- *Skip-gram:* Predict surrounding words given a central word.

Through training, the model learns to associate each word in the vocabulary with a dense vector of real numbers. These vectors capture meaningful linguistic relationships. For example, consider the famous analogy from Mikolov et al. (2013):

$$\text{vector}(\text{"king"}) - \text{vector}(\text{"man"}) + \text{vector}(\text{"woman"}) \approx \text{vector}(\text{"queen"}).$$

It demonstrates how arithmetic operations on word embeddings can reveal latent semantic structure. What this says is that if we take the word embedding from "king," subtract that vector with the vector of the word embedding of "man," and add the vector for "woman," it will be more similar to the word embedding for "queen," illustrating the semantic meaning embedded in these word vectors.

The output of a model such as Word2Vec is an *embedding matrix*, where each row corresponds to a word in the vocabulary and each column represents a learned feature of the word. These embeddings can then be used as inputs to downstream models (i.e., models specializing in a specific task) for classification, clustering, or regression tasks—offering a more compact and semantically rich representation relative to the sparse count vectors discussed in the previous section.

These methods still have limitations, however, such as *static embeddings*: Once trained, each word has a single fixed embedding, regardless of context. This situation creates

problems for words with multiple meanings. For instance, consider the word "position" in two financial contexts:

- "The fund increased its position in Apple."
- "The trader accepted a new position at the firm."

In both cases, the same word vector would be used, even though "position" refers to an investment in one case and a job title in the other.

To address this issue, researchers introduced *recurrent neural networks* (RNNs) that process sequences of words in order, allowing models to generate *contextual embeddings*—word representations that change depending on surrounding words. Two widely used architectures in this family are the *long short-term memory* (LSTM) network and the *gated recurrent unit*. These models read a sentence one word at a time and retain a memory of the preceding words, enabling them to generate richer, context-aware representations.

Although sequential models such as LSTM networks improved performance on many NLP tasks, they also introduced new challenges:

- *Computational inefficiency:* Processing words one at a time limits parallelization and increases inference time (i.e., how long it takes a model to output predictions or embeddings).
- *Unidirectional context:* Standard LSTM networks typically read text in one direction—usually left to right—so they can condition only on previous words and not on future context.

These limitations paved the way for the next major breakthrough in NLP: the transformer architecture, which enables more efficient and flexible modeling of text with full context awareness. This development is explored in the following section.

The Transformer Revolution

The introduction of the *transformer architecture*, proposed by Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosukhin (2017), in the landmark paper "Attention Is All You Need," marked a turning point in NLP development. The transformer not only outperformed prior models across a wide range of NLP benchmarks but also became the foundational architecture for all modern LLMs, including BERT, GPT, and their successors.

Although the transformer is a neural network like those discussed in the previous section, it departs significantly from earlier sequential models such as RNNs and LSTM networks. To understand its impact and how it differs from past architectures, this section presents a high-level conceptual overview of the transformer architecture and its two key innovations: *self-attention* and *encoder–decoder separation*.

Self-Attention: Capturing Relationships Across Text

At the core of the transformer is the *self-attention mechanism*, which allows the model to consider the relationship between all words (or *tokens*) in a given input sequence, regardless of their position. In contrast to LSTM networks, which process sequences word by word and typically look only at previous words (i.e., left to right), self-attention enables the model to compute

relevance scores across the entire sequence in parallel. This approach means that each word can attend to every other word, with the model weighting them based on learned relevance.

For example, in the sentence, "The investor sold the stock because it was overvalued," the word "it" could plausibly refer to either "stock" or "investor." A model equipped with self-attention can learn to focus on the correct referent—"stock"—by assigning higher attention weight to it. This capability to model long-range dependencies and ambiguity is one of the reasons why transformer-based models have proven so effective in capturing the complexity of human language.

The Encoder–Decoder Architecture

The original transformer architecture consists of two major components:

- *Encoder:* processes the input text and converts it into a contextualized vector representation
- *Decoder:* uses this representation to generate an output sequence, such as translated text, a summary, or a response

This encoder–decoder configuration was designed for such tasks as *machine translation*, *summarization*, and *question answering*—where the model must first understand the full input before producing a meaningful output. The encoder portion of the model captures the semantic content of the source text, while the decoder portion generates new text conditioned on that information captured by the encoder.

In practice, however, encoder and decoder components are often used *independently*, depending on the task:

- *Encoder-only models* (e.g., BERT)[1] are designed to create high-quality, contextualized embeddings of the input text. These embeddings can then be used as features for downstream tasks, such as sentiment analysis, classification, or information retrieval. Encoder-only models are especially valuable in settings where the goal is to interpret or score a piece of text rather than generate new text.
- *Decoder-only models* (e.g., GPT)[2] are optimized for *language generation*. These models predict the next token in a sequence given all previous tokens—a formulation known as *causal* or *autoregressive language modeling*. This approach enables the model to generate coherent text, answer questions, and engage in dialogue. Decoder-only models form the backbone of modern chat-based systems and instruction-following agents.

The transformer architecture introduced two breakthroughs: the ability to process all tokens in parallel through self-attention and the modular encoder–decoder framework that enables both understanding and generation of new text. These innovations not only improved performance

[1]BERT refers to the bidirectional encoder representations from transformers model introduced by Devlin, Chang, Lee, and Toutanova (2019).

[2]GPT refers to the generative pretrained transformer model introduced by OpenAI through a series of breakthrough papers (Radford, Narasimhan, Salimans, and Sutskever 2018; Radford, Wu, Child, Luan, Amodei, and Sutskever 2019; Brown, Mann, Ryder, Subbiah, Kaplan, Dhariwal, Neelakantan, et al. 2020). GPT also refers to the class of decoder-only models popularized through modern LLMs.

across a wide range of NLP tasks but also made it feasible to scale models to unprecedented size, giving rise to today's LLMs.

The next section explores how these architectural advances, coupled with massive pretraining, have enabled a new paradigm of generative modeling that can solve a broad range of tasks with little or no task-specific data.

Pretraining, Fine-Tuning, and the Modular Nature of LLMs

To understand how LLMs work—and why they have become so powerful—it is helpful to break down their training into distinct phases: *pretraining* and *fine-tuning*. These phases reflect a modular and scalable approach to training LLMs that separates the acquisition of general linguistic and world knowledge from the adaptation of the model to specific tasks. Although the underlying model architecture (encoder, decoder, or both) plays an important role in shaping the model's capabilities, it is ultimately the training process that determines what the model can do in practice.

Pretraining: Building General Linguistic and World Knowledge

Pretraining is the foundational phase of LLM development. In this stage, the model is exposed to a massive corpus of raw textbooks, articles, websites, and other publicly available documents—and trained to predict missing or future words. This process does not require labeled data, making it highly scalable. The goal is to instill the model with a statistical understanding of language and a broad (though implicit) knowledge of the world.

The specific training objective depends on the architecture:

- *Masked language modeling (MLM):* Used primarily for training *encoder-only* models (e.g., BERT), MLM involves hiding or "masking" certain words in a sentence and training the model to predict the masked word(s) based on surrounding context. For example, given the input "The company reported a [MASK] gain in profits," the model learns to infer that "strong" might be an appropriate word. Because encoder models attend to the entire input at once, they generate rich contextual embeddings useful for classification and retrieval tasks.
- *Causal language modeling (CLM):* Used to train *decoder-only* models (e.g., GPT), CLM trains the model to predict the next word in a sequence given all previous words. For instance, given "The company reported a strong gain in," the model learns to predict "profits." This autoregressive objective aligns naturally with generative tasks and enables the model to produce coherent, fluent text. Radford, Narasimhan, Salimans, and Sutskever (2018) demonstrated that performance improves substantially with model scale, leading to the development of highly parameterized models capable of general-purpose text generation.

This pretraining phase is computationally intensive and typically carried out by large technology firms with access to significant compute infrastructure and massive datasets. The resulting pretrained models can be distributed and reused, however, providing a strong foundation for downstream customization.

Fine-Tuning: Adapting Models to Tasks and Domains

Although pretraining equips LLMs with general capabilities, it does not tailor them to specific tasks. Pretrained models may be able to generate fluent text or encode sentences into embeddings, but they often lack the domain-specific behavior needed for practical applications in such areas as finance. This is where *fine-tuning* comes in.

Fine-tuning involves continuing the training of a pretrained model on a smaller, domain-specific or task-specific dataset. This process allows users to adapt general-purpose models for their own use cases without starting from scratch.

For *encoder-only models*, fine-tuning typically involves supervised learning. For example, to build a sentiment classifier, a practitioner would start with a pretrained BERT model and fine-tune it on labeled data consisting of text samples paired with sentiment labels. The resulting model becomes highly specialized for sentiment classification. The resulting model will be able to produce meaningful outputs only for that specific task, however, which is why fine-tuned encoder-only models are also referred to as "narrow" or "task-specific" models.

For *decoder-only models*, fine-tuning can be more general. After pretraining on internet-scale data, these models are often fine-tuned on curated datasets of human interactions—such as question-and-answer pairs or chat logs—to teach the model how to be helpful, safe, and responsive in a conversational setting. Without this additional tuning, a model might respond to a prompt such as "What is your name?" with a story or unrelated prose. With fine-tuning, it learns to respond appropriately to user queries.

One of the reasons decoder-only models have become so widely used is their *flexibility*. Rather than needing a new model for each task (e.g., sentiment classification, summarization, ESG scoring), a single decoder-only model can be fine-tuned to perform a wide range of tasks simply by changing the *prompt*—the input instruction or context provided to the model. For instance:

- "Classify the sentiment of this text: 'Markets rallied after the rate cut.'"
- "Summarize the following earnings report."
- "List three ESG risks mentioned in this document."

This ability makes decoder-only models attractive in such domains as finance, where labeled data are often scarce but a wide range of tasks need to be performed on unstructured text. By shifting the complexity to the prompt design, practitioners can avoid the costly process of training narrow models for every new application.

Prompting vs. Fine-Tuning

Although *prompting* allows users to extract useful behavior from a pretrained or instruction-tuned model without any additional training, it has its limits. In cases where the model fails to follow instructions, struggles with domain-specific jargon, or underperforms on niche tasks, *fine-tuning* can still play an important role.

Fine-tuning can be used for the following:

- *Task-specific adaptation:* training the model for a narrow task, such as legal clause extraction or credit rating prediction

- *Domain adaptation:* exposing the model to domain-specific data (e.g., financial filings, regulatory texts) to improve its familiarity with specialized vocabulary and formats
- *Knowledge injection:* teaching the model facts or behaviors that are underrepresented in the pretraining corpus

In practice, many financial workflows now combine these approaches, using off-the-shelf LLMs where prompt engineering suffices and fine-tuning smaller versions of the model when reliability and domain expertise are critical.

Computational Efficiency and Fine-tuning

Although LLMs with billions of parameters tend to exhibit superior performance across a range of tasks, this ability comes at a significant computational cost. Both training and deploying such models require high-end hardware—typically graphics processing units (GPUs) or tensor processing units (TPUs)—and scale roughly with model size. As a result, the use of very large models can be prohibitive in environments with limited compute resources or strict latency constraints, such as real-time financial systems.

One solution is to *fine-tune smaller models to replicate the behavior of larger ones*, commonly done through a process known as *knowledge distillation*. In this setup, a high-performing, large model (the *teacher*) generates high-quality outputs for a given task and a smaller model (the *student*) is trained to imitate these outputs. For instance, if a large model excels at summarizing long financial documents, its summaries can serve as labeled training data to fine-tune a smaller model to perform the same task. This process allows practitioners to deploy a compact model that behaves similarly to a much larger one but with far lower computational requirements.

In addition to distillation, another strategy for improving efficiency is *quantization*. Quantization reduces the numerical precision of a model's parameters—from 32-bit floating point numbers (FP32) to lower-precision formats, such as 8-bit integers (INT8) or 4-bit floats. Although this approach can slightly reduce model performance, it dramatically reduces the memory footprint and speeds up inference, particularly on GPU hardware. Because inference cost is one of the most significant factors in large-scale deployment, quantization plays a key role in bringing LLM capabilities to environments with limited resources.

Taken together, these techniques—fine-tuning, distillation, and quantization—offer powerful levers for balancing performance and efficiency. They enable organizations to deploy LLMs that are adapted to specific tasks and constraints, without bearing the full costs of operating state-of-the-art, billion-parameter models. In the context of finance, where tasks are often repeated at scale and latency can be critical, these efficiency gains are not just convenient; they are essential.

Fine-Tuning Techniques

The specific strategy used to fine-tune a language model depends largely on its architecture—whether the model is *encoder only* or *decoder only*. These two types of models serve different purposes and therefore require distinct approaches when adapting them to downstream tasks.

Fine-Tuning Encoder-Only Models

Encoder-only models, such as *BERT*, are designed to generate rich, contextual embeddings of input text sequences. These embeddings are high-dimensional vectors that capture the semantic meaning of the text, incorporating the full context of surrounding words through self-attention.

Fine-tuning these models generally involves one of two approaches:

- *Feature extraction:* In this setup, the pretrained model is used to generate embeddings for each input document, and these embeddings are then used as input features to a separate, downstream machine learning model (e.g., logistic regression, random forest). The weights of the language model itself remain fixed.
- *End-to-end fine-tuning:* A classification or regression head (i.e., an additional fully connected feedforward layer) is added on top of the encoder model, and the entire architecture—including the weights of the pretrained model—is trained jointly on labeled data. For example, if a BERT model outputs a 768-dimensional embedding vector, a single-layer neural network can be added to map this vector to a three-class output for sentiment classification (e.g., positive, neutral, negative). This process fine-tunes the entire network for the target task.

End-to-end fine-tuning often leads to better performance but is more computationally demanding. Variants of this approach include *partial fine-tuning*, where only the top few layers of the model are updated while the rest are frozen (i.e., unchanged), and *layer-wise freezing*, where layers are gradually unfrozen during training. These techniques reduce computational cost and help prevent overfitting, especially when labeled data are scarce.

Fine-Tuning Decoder-Only Models

Decoder-only models, such as *GPT*, are designed for text generation and contain significantly more parameters than encoder-only models. Fine-tuning these models in a traditional end-to-end fashion is often prohibitively expensive in terms of memory and compute resources, especially when dealing with billions of parameters. To address this problem, researchers have developed more efficient fine-tuning methods, the most prominent of which is *low-rank adaptation* (LoRA).

LoRA was introduced by Hu, Shen, Wallis, Allen-Zhu, Li, Wang, and Chen (2021) as a *parameter-efficient fine-tuning* method for large models. Instead of updating all weights of the pretrained model, LoRA inserts small, trainable weight matrices into the existing architecture while freezing the original weights. These low-rank matrices capture the necessary task-specific adjustments without needing to train the entire model. As a result, LoRA significantly reduces the number of trainable parameters, lowers GPU memory usage, and allows for faster training.

Importantly, LoRA also helps mitigate *catastrophic forgetting*—a phenomenon where a model forgets previously learned knowledge when fine-tuned on a narrow task—because the original pretrained weights remain intact. This feature offers a form of regularization when training highly parameterized models and makes LoRA particularly attractive for adapting general-purpose LLMs to specialized domains, such as finance, without compromising their broader language capabilities.

Deployment Strategies and Infrastructure Considerations for LLMs in Finance

The Challenge of Real-Time Information in Finance

One of the most significant challenges in applying LLMs in financial contexts is the pace at which information changes. News articles, social media posts, SEC filings, and earnings call transcripts can alter the investment landscape in real time. Traditional LLMs, however, are static once trained; they can make use only of the information available at the time of their pretraining or fine-tuning. In high-stakes, time-sensitive domains, such as finance, this fact poses a fundamental limitation: Retraining a large model every time new information becomes available is both impractical and cost prohibitive.

To overcome this constraint, many LLM-based systems now use a design pattern known as *retrieval-augmented generation* (RAG). RAG is not a model itself but an architectural framework that allows LLMs to dynamically incorporate external information at inference time. This ability enables a form of real-time updating without retraining the model.

RAG systems typically consist of two main components:

- An *encoder-based retriever* processes the user's query and searches an external document repository (such as a database of recent filings or news articles) for relevant information. This component is typically built using an encoder-only model, which generates embeddings for both the query and the documents and then identifies the most semantically similar content through measuring *cosine similarity* of the embedding of the user's query and embeddings of the document repository.
- A *decoder-only language model* receives the retrieved documents as context and generates a coherent answer to the user's query. This model does not need to have seen the documents during training—it uses the retrieved information as context provided via the prompt.

This modular approach solves a key problem in financial NLP: incorporating *new and proprietary data* on the fly. Rather than training the model to "know everything," encoders are used to structure and fetch the relevant information, and decoders are used to synthesize and reason over that information. This process is particularly powerful in such workflows as

- summarizing recent earnings call transcripts,
- responding to ESG-related inquiries with up-to-date regulatory documents, and
- generating investment commentary based on newly filed 10-Ks or 8-Ks.

By separating retrieval from generation, RAG frameworks enable dynamic, domain-aware responses without compromising latency or requiring model retraining.

Agentic Frameworks and Autonomous Systems

Although retrieval-augmented systems allow LLMs to access up-to-date information, many real-world financial workflows require more than just a single question-and-answer interaction. Such tasks as portfolio monitoring, document triage, regulatory surveillance,

and risk assessment often require *multistep reasoning*, *conditional execution*, and *coordination across multiple tools or models*. This is where agentic frameworks come into play.

Agentic Frameworks

An *agentic framework* refers to an architectural approach in which LLMs are embedded within a broader decision-making system that can sequence multiple steps to achieve a complex objective. Rather than providing a single response to a single prompt, the LLM acts as a "thinking component" that can

- break down a goal into sub-tasks,
- decide what action to take at each step,
- retrieve or generate intermediate information,
- use external tools (e.g., calculators, databases, search APIs), and
- determine when the task is complete.

In financial applications, this might involve chaining together such tasks as

- reading and extracting risk factors from a 10-K filing,
- identifying which factors are new or material relative to prior filings,
- summarizing their potential impact on valuation, and
- flagging the result for human review if certain thresholds are exceeded.

Rather than relying on a single LLM to do all this, the system orchestrates multiple steps, often using different models or tools tailored to the specific subtasks. This modular and compositional approach enables more interpretable, reliable, and extensible pipelines—crucial attributes in high-stakes domains such as finance.

Autonomous Agents

Agentic systems can be taken a step further with *autonomous agents*: LLM-based systems that are goal directed and capable of operating with *minimal human oversight*. These agents maintain a persistent objective (e.g., "monitor the top 500 firms for changes in litigation risk"), autonomously initiate actions to gather information, and adapt their behavior according to the results.

What distinguishes autonomous agents from simpler pipelines is their ability to

- formulate plans based on their objective,
- monitor their own progress and iterate,
- communicate with multiple systems or data sources, and
- run continuously or on demand.

Examples in finance might include

- an *autonomous compliance monitor* that scans regulatory filings, compares them with internal policies, and flags discrepancies;

- a *market intelligence agent* that compiles real-time information across sources to maintain an up-to-date dashboard for analysts; or
- a *due diligence assistant* that autonomously reads through multiple data sources and summarizes potential red flags during deal evaluation.

These systems remain experimental in many financial institutions, but early results suggest that autonomy can reduce human workload on repetitive tasks while increasing responsiveness to new developments. They also introduce new risks, however, particularly around reliability, traceability, and control. Because the system is capable of executing its own plan, rigorous evaluation and monitoring infrastructure are needed to ensure safety, especially in regulated environments.

Deployment Models: API Access vs. Self-Hosting

As LLMs become increasingly integral to financial workflows, institutions must carefully consider *how* these models are deployed. Deployment choices involve trade-offs across cost, control, compliance, performance, and security—particularly when dealing with sensitive or proprietary data. Broadly, there are two primary approaches to deploying LLMs: *commercial API access* and *self-hosting open-source models*.

Commercial API Access

The most accessible way to use LLMs is through cloud-based APIs provided by such companies as OpenAI, Anthropic, Google, or Cohere. These models are hosted on proprietary infrastructure and accessed via subscription or usage-based pricing. They offer the following advantages:

- *Ease of integration:* These models allow for quick setup with standard APIs.
- *Performance:* They provide access to state-of-the-art models trained on massive datasets and enhancing models with additional tools, such as internet search or code execution.
- *Scalability:* Providers manage infrastructure, allowing dynamic scaling of workloads.

The models have limitations as well, however:

- *Data privacy concerns:* Sensitive or proprietary information must be transmitted to third-party servers, raising concerns about confidentiality, data retention, and compliance with data protection regulations.
- *Lack of control:* The model architecture, parameters, training data, and update schedule are fully managed by the provider, limiting transparency and customization.
- *Ongoing cost exposure:* Pay-as-you-go pricing can become expensive at scale, especially for high-frequency or latency-sensitive applications.

API-based access is suitable for exploratory research, public-facing applications, and tasks that do not involve proprietary financial data. It is often insufficient, however, for enterprise-grade use cases that require full data custody and customized model behavior.

Self-Hosting and Open-Source Models

An alternative is to deploy open-source LLMs, such as *LLaMA*, *Mistral*, *Falcon*, or *GPT-J*, on private infrastructure. This approach allows organizations to run inference and fine-tuning on their own servers—either on premises or in private cloud environments—and provides the following advantages:

- *Data security and compliance:* No data can leave the organization's environment, which is critical when handling client information, trading strategies, or internal documents.
- *Customizability:* Models can be fine-tuned, distilled, or augmented to meet domain-specific requirements or to improve performance on financial texts.
- *Cost control over time:* Although upfront compute and staffing costs are high, long-term operational costs may be lower than continued API usage at scale.

This approach has two main limitations:

- *Infrastructure complexity:* Running large models requires specialized hardware (e.g., GPUs), as well as specialized human capital for deployment, scaling, and monitoring.
- *Lag behind frontier models:* Open-source models often trail commercial models in absolute performance, although the gap has narrowed considerably in recent releases.

Hybrid and Strategic Considerations

Many institutions adopt *hybrid strategies*, using commercial APIs for low-risk or generic use cases and deploying internal models for high-sensitivity or proprietary tasks. Two examples follow:

- Use ChatGPT to summarize public market news.
- Use an in-house model to analyze internal credit memos or regulatory filings.

Ultimately, the choice of deployment model must align with the organization's data governance policies, use-case sensitivity, latency requirements, and long-term cost structure. In finance—where intellectual property, compliance, and risk management are paramount—these considerations are not peripheral but central to the successful deployment of LLMs.

Applications of LLMs in Finance

Sentiment analysis and topic modeling have long been central to the application of NLP in finance. Seminal works, such as Tetlock's (2007) study on the predictive power of media pessimism and Loughran and McDonald's (2011) research on domain-specific sentiment dictionaries, showed that textual sentiment contains information relevant to returns, volatility, and trading behavior. Early methods relied on lexicons, dictionaries, or simple machine learning classifiers, but these techniques have evolved alongside the growing sophistication of NLP models, as discussed previously.

In this section, I outline how LLMs have opened new frontiers in financial NLP—moving beyond traditional tasks toward applications in compliance, ESG monitoring, risk surveillance, and financial summarization. I then present a practical tutorial demonstrating how LLMs

can be implemented for sentiment analysis, focusing on open-source models accessible to practitioners and researchers. Although LLM applications extend much further, this hands-on example provides a foundational approach for integrating LLM techniques into financial workflows.

Classic NLP Tasks and How LLMs Changed Them

Before the rise of LLMs, many core financial NLP tasks—such as named entity recognition (NER), event extraction, and relation extraction—were tackled using rule-based methods, feature engineering, or early machine learning models, such as conditional random fields (CRFs). For example, Wang, Xu, Liu, Gui, and Zhou (2014) applied CRFs augmented with domain-specific dictionaries to identify company names, tickers, and monetary amounts in financial news, and Yang, Chen, Liu, Xiao, and Zhao (2018) developed document-level event extraction systems for the Chinese stock market.

With the advent of LLMs, many of these tasks have become almost trivial: Modern models can perform zero-shot or few-shot NER, relation extraction, or event detection without task-specific architectures or large, labeled datasets. Recent evaluations (e.g., Lu and Huo 2025) have shown, however, that although general-purpose LLMs perform reasonably well, fine-tuned domain-specific models still outperform them in precision-critical extraction tasks, especially when dealing with subtle distinctions in entity types or regulatory language.

New Frontiers Enabled by LLMs

LLMs have significantly expanded the scope of NLP applications in finance.

- *Compliance monitoring and regulatory reporting:* Hillebrand, Berger, Deußer, Dilmaghani, Khaled, Kliem, Loitz, et al. (2023) introduced ZeroShotALI, demonstrating that LLMs can match financial documents against complex regulatory requirements in a zero-shot setting, potentially automating parts of compliance checks and audits.
- *ESG analysis:* Mehra, Louka, and Zhang (2022) fine-tuned BERT to develop ESGBERT, a model capable of classifying and extracting ESG-related content from sustainability and CSR reports, improving the automated detection of environmental, social, and governance themes.
- *Risk monitoring via news and events:* Guo, Jamet, Betrix, Piquet, and Hauptmann (2020) built ESG2Risk, a pipeline using LLM-based models to track ESG-related news and predict abnormal stock volatility, illustrating how LLMs can detect material nonfinancial risks from unstructured text.
- *Financial document summarization:* Kim, Muhn, and Nikolaev (2023) showed that ChatGPT can effectively summarize long, complex 10-K filings, helping investors distill key risks and opportunities. Their results suggest that LLM summarization improves market efficiency by reducing information asymmetry.

These examples illustrate that LLMs are no longer limited to traditional classification or extraction. They now support more complex workflows involving multistep reasoning, summarization, and regulatory interpretation.

Sentiment Analysis, Topic Modeling, and Quantitative Investing

Sentiment analysis has historically been one of the most widely used NLP applications in quantitative finance. Early approaches, such as FinBERT (Araci 2019), fine-tuned BERT models on financial news and earnings calls, offering more accurate sentiment classifications than traditional dictionary-based methods. These sentiment signals became important features in quantitative models, helping improve predictions of stock returns or volatility.

Recent advances in LLMs have expanded how text-derived signals are used in investing—moving beyond simple sentiment scores to richer representations that directly predict returns or explain price movements. For example, Lopez-Lira and Tang (2023) tested whether prompting generative LLMs, such as GPT-4, on company news headlines could predict next-day stock returns. They found that even without fine-tuning, the model's outputs contained predictive signals, suggesting that LLMs implicitly capture financial sentiment and context well enough to inform trading strategies. This line of research connects closely with the work of Chen, Kelly, and Xiu (2024), who used LLM-derived *embeddings* as features for predicting returns across 16 international equity markets. They found that these embedding-based features, paired with a simple penalized linear model, improve the risk-adjusted returns of the resulting portfolios over traditional approaches.

Another emerging approach focuses on organizing and clustering the patterns found in textual data. Cong, Liang, Zhang, and Zhu (2024) introduced the idea of *textual factors*—interpretable clusters of themes or topics identified by applying clustering techniques to LLM-derived embeddings. These textual factors can be incorporated into economic and financial models, providing scalable, data-driven ways to analyze large sets of unstructured information, from macroeconomic news to corporate filings.

Wang, Izumi, and Sakaji (2024) combined generative prompting and quantitative modeling by using LLMs to produce explainable summaries of news, corporate events, or leadership tone, which were then transformed into structured *factors* for predicting stock movements. Together, these advances highlight the versatility of modern LLMs: They can be used to generate direct predictions, to build structured thematic clusters, or to extract interpretable features through carefully designed prompts.

LLMs in Practice

In this section, I demonstrate how to apply LLMs for sentiment classification using Python. To follow along with this tutorial, I recommend using Google Colab because it provides easy access to GPUs (i.e., speeding up model outputs) and a reproducible environment for testing.

When working with LLMs, it is important to be familiar with Hugging Face,[3] the platform where open-source datasets and language models are shared. The models here range from pretrained LLMs to models fine-tuned from those pretrained LLMs. Hugging Face has two Python packages that can be used to easily load models and datasets: *transformers* and *datasets*.

In this exercise, we will use the Financial Phrasebank dataset introduced by Malo, Sinha, Korhonen, Wallenius, and Takala (2014). This dataset consists of 4,840 sentences randomly

[3]For more information, visit https://huggingface.co/.

sampled from financial news, with each sentence being labeled by a set of annotators as "positive," "negative," or "neutral."

I will demonstrate two methods for sentiment analysis using LLMs: (1) a fine-tuned feature extraction model using SBERT[4] and (2) a pretrained language model using OpenAI's API.

SBERT Feature Extraction Model

SBERT (Reimers and Gurevych 2019) is a BERT-based model that specializes in sentence-level embeddings. That is, text is input to the model and a vector is returned. Because sentiment scores are not directly output by the model, we need to train the model to predict sentiment scores. To do so, we create a training and test set (in practice, we would use a validation set for hyperparameter tuning).

```
>> !pip install -U datasets

>> from datasets import load_dataset
>> data = load_dataset("financial_phrasebank","sentences_50agree")
>> data = data['train'].to_pandas()

>> train = data.sample(frac=0.8, random_state=200)
>> test = data.drop(train.index)
```

SBERT can be loaded through the *sentence-transformers* Python package, which handles much of the preprocessing required to extract vector embeddings. Next, I show how to install the package, load the model, and extract embeddings for each sentence in the training and test datasets. Note that it is necessary to split the text into chunks or "batches" because computational restraints prohibit evaluating all sentences at once.

```
>> !pip install sentence-transformers

>> from sentence_transformers import SentenceTransformer
>> model = SentenceTransformer('all-MiniLM-L6-v2')

>> def batch_sentences(sentences,batch_size=100):
    batch_sentences = []
    for i in range(0,len(sentences),batch_size):
        batch_sentences.append(sentences[i:i+batch_size])
    return batch_sentences

train_batches = batch_sentences(train['sentence'].tolist())
test_batches = batch_sentences(test['sentence'].tolist())

X_train = [model.encode(batch) for batch in train_batches]
X_test = [model.encode(batch) for batch in test_batches]
```

[4]Available at https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

Once the embeddings are loaded, they can be concatenated to formulate a matrix of size $n \times 384$. You can then fit a ridge classification model on those embeddings to create a mapping between embedding vectors to sentiment labels. Predictions on the test set are obtained by passing the test embeddings through the fitted model.

```
>> import numpy as np
>> from sklearn.linear_model import RidgeClassifier

>> # Train model
>> clf = RidgeClassifier()
>> clf.fit(np.vstack(X_train),train['label'])

>> # Predictions: 0 = Neg, 1 = Neut, 2 = Pos
>> preds = clf.predict(np.vstack(X_test))
```

In practice, you would want to create a validation set to fit the model with different penalization terms (i.e., alpha terms) in the ridge regression to determine the parameter that produces the best fit. For this example, I simply demonstrate obtaining predictions.

OpenAI ChatGPT

Although open-source LLMs can be run using the transformers package in Python, I will focus on an easy-to-implement pipeline using OpenAI's API. Instead of sourcing computational resources or getting caught up in the technical aspects of running LLMs, you can instead rely on an API-based approach to do this for you. The idea is that you can send your prompted text to the API, and the output from the model will be returned.

Because I am using a sophisticated, pretrained LLM like GPT-3.5, I do not need to train the model but instead simply prompt the text for the model to respond with a sentiment score:

```
>> prompt = """Evaluate the sentence below and determine the sentiment
score.
Respond with either positive, neutral, or negative.

Sentence: %sentence

Response:"""

>> prompted_sentences = [
prompt.replace("%sentence",i) for i in data.sentence
]
```

I use this prompt because it assists in (1) reducing the number of output tokens that I need to obtain from the model (i.e., output tokens are more expensive to produce) and (2) easy parsing of the response because the model will respond immediately with solely "positive," "neutral," or "negative." If you do not prompt it to provide a "Response:" at the end, the model typically provides some preamble to its actual response, making the response more costly as well as harder to parse.

To obtain responses from the API, the *openai* Python package can be used. To obtain responses from the API, go to OpenAI's API platform[5] and generate an API key, which associates your input with your account for billing. Then, obtaining sentiment scores is as easy as looping through the list of prompted sentences and parsing the response.

```
>> !pip install openai
>> from openai import OpenAI
>> client = OpenAI(api_key="ENTER_API_KEY_HERE")
>> # Define function to query model
>> def get_gpt_sentiment(sentence,model_name="gpt-3.5-turbo"):
      response = client.responses.create(
          model="gpt-3.5-turbo",
          input=sentence
      )
      return response.output_text
>> responses = [get_gpt_sentiment(i) for i in prompted_sentences]
>> responses[0] # = "Neutral"
```

Risks, Challenges, and Considerations

As financial institutions adopt LLMs into their workflows, it is important to be aware of the risks and challenges of doing so. This section outlines several of the most pressing challenges: output reliability, evaluation standards, model leakage over time, and legal uncertainty.

Hallucinations and Output Reliability

LLMs are probabilistic models designed to predict the next most likely word given the previous sequence of words. Although this ability makes them flexible and powerful, it also means they are prone to producing *hallucinations*—text that is fluent, plausible, and confidently stated but factually incorrect or fabricated.

A now-famous example involves a lawyer who used ChatGPT to draft a legal filing. The model generated convincing citations to court cases that, upon inspection, did not exist. This episode underscores the difficulty in relying on LLMs in domains where factual precision is essential—such as regulatory interpretation, earnings analysis, or compliance reporting.

One solution is the use of RAG frameworks, discussed earlier, which allow the model to consult an external knowledge base and leverage the relevant information in the knowledge base

[5]Visit https://openai.com/api/.

in crafting its response. Even with RAG, however, verifying that the model used the information appropriately remains a challenge.

In practice, evaluating output reliability often requires *secondary evaluation mechanisms* such as the following:

- *Human review*, particularly for complex or high-impact use cases
- *Secondary LLMs*, which can be prompted to judge whether the model or workflow being evaluated is generating the expected output and/or sourcing the correct information based on the prompt

Evaluating output becomes even more challenging in *multi-agent pipelines*—systems where different LLMs or modules hand off intermediate outputs. If one model produces a hallucinated output, that output may silently propagate through the system, breaking task logic or triggering unintended behaviors. Thus, output validation and fallback systems are essential components in production-grade deployments.

Evaluation in Domain-Specific Contexts

In general NLP research, model capabilities are evaluated using benchmark datasets, which are labeled datasets used to determine LLM performance in different contexts or tasks. Domain-specific evaluation in finance remains an underdeveloped area, however. Most financial tasks—such as interpreting a 10-K filing, generating risk summaries, or classifying ESG disclosures—lack standardized ground truth labels.

This situation creates several challenges:

- *No universal "correct" answer* exists for such tasks as summarization or narrative analysis.
- *Crafting an evaluation metric* is not as straightforward as evaluating classification performance. Often, one must be creative in constructing a single metric to determine performance or reliability in a pipeline.
- *Task-specific benchmarks* must often be constructed in house, with significant human effort.

For such applications as return prediction or sentiment classification, evaluation may be grounded in market response (e.g., future returns). But for more qualitative tasks—such as policy interpretation or stock recommendations—quantifying model performance requires human expertise and domain fluency. Without rigorous evaluation, firms risk deploying models whose performance may degrade silently under new conditions or inputs.

Forward-Looking Contamination in Backtests

Another risk in using LLMs for alpha signal generation is the issue of *forward-looking bias*. Most large LLMs have been pretrained on internet-scale text up to a certain cutoff date. If that training corpus includes data from the backtest period, then the model may "know" future outcomes during historical simulations, even if inadvertently.

For instance, if a model is used to predict the sentiment or return of stocks using news headlines from 2010, the model may implicitly encode information that was not available at the time. That is, if a news article has the headline "NVIDIA Introduces X Chip" and the model implicitly knows that this chip was very popular from forward-looking training data, this will inherently bias the model's response to be more positive.

Some studies have noted, however, that this forward-looking information can sometimes negatively influence out-of-sample performance because the model is distracted by the future narratives and information, not focusing on the current information available. Glasserman and Lin (2024) referred to this phenomenon as the *distraction effect*.

Compliance, Data Security, and IP Risk

As discussed earlier in the context of deployment models, the use of third-party APIs for LLM inference raises serious compliance concerns. But beyond data privacy, a new class of legal and intellectual property (IP) issues is emerging around ownership and liability.

Key concerns include the following:

- *Ownership of model outputs:* Does the firm own the generated text? What if that text closely resembles a copyrighted source seen during pretraining?
- *Liability for misinformation:* If a model generates a misleading investment recommendation or incorrect regulatory interpretation, who is responsible?
- *Internal data leakage:* If proprietary data are used in prompts or fine-tuning and processed by a third-party model, the data may be inadvertently retained or exposed, depending on provider policies.

These concerns are amplified in finance, where data governance and regulatory compliance are not optional. Firms should establish clear policies around

- internal versus external LLM usage,
- data anonymization and minimization practices, and
- logging, audit trails, and human-in-the-loop reviews.

Conclusion

The advances in NLP have revolutionized financial workflows through the development of LLMs. Although traditional, task-specific NLP models found their way into earlier quantitative models, the flexibility granted by conversational models has made LLMs accessible to nearly every facet of the investment process. As the technology continues to advance, it is important to understand the different types of models, their applications, and implementation frameworks.

The power of these models comes with important caveats, however. Effective deployment requires an understanding of their architectural foundations, training processes, and the trade-offs involved in different implementation strategies. Moreover, issues of reliability,

interpretability, computational cost, compliance, and domain-specific evaluation demand careful consideration, particularly in high-stakes financial contexts.

The future of NLP in finance will likely be shaped by hybrid approaches that combine the generative power of LLMs with structured data pipelines and traditional time-series models. As models continue to evolve—integrating multimodal inputs, improving reasoning abilities, and adapting to specialized domains—success will depend not only on technical sophistication but also on rigorous evaluation, sound governance, and a deep understanding of the financial domain.

In short, LLMs are a powerful technology that will make information assimilation in financial markets much faster and efficient. Like any powerful tool, however, their value lies not in raw capability but in careful, context-aware application.

References

Araci, Dogu. 2019. "FinBERT: Financial Sentiment Analysis with Pre-Trained Language Models." Working paper (27 August). doi:10.48550/arXiv.1908.10063.

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, et al. 2020. "Language Models are Few-Shot Learners." In *NIPS'20: Proceedings of the 34th International Conference on Neural Information Processing Systems*, 1877–901. doi:10.48550/arXiv.2005.14165.

Chen, Yifei, Bryan T. Kelly, and Dacheng Xiu. 2024. "Expected Returns and Large Language Models." Working paper (23 August). https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4416687.

Cong, Lin William, Tengyuan Liang, Xiao Zhang, and Wu Zhu. 2024. "Textual Factors: A Scalable, Interpretable, and Data-Driven Approach to Analyzing Unstructured Information." NBER Working Paper 33168 (November). doi:10.3386/w33168.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–86. doi:10.48550/arXiv.1810.04805.

Glasserman, Paul, and Caden Lin. 2024. "Assessing Look-Ahead Bias in Stock Return Predictions Generated by GPT Sentiment Analysis." *Journal of Financial Data Science* 6 (1): 25–42. doi:10.3905/jfds.2023.1.143.

Guo, Tian, Nicolas Jamet, Valentin Betrix, Louis-Alexandre Piquet, and Emmanuel Hauptmann. 2020. "ESG2Risk: A Deep Learning Framework from ESG News to Stock Volatility Prediction." Working paper (5 May). doi:10.48550/arXiv.2005.02527.

Hillebrand, Lars, Armin Berger, Tobias Deußer, Tim Dilmaghani, Mohamed Khaled, Bernd Kliem, Rüdiger Loitz, et al. 2023. "Improving Zero-Shot Text Matching for Financial Auditing with Large Language Models." In *DocEng '23: Proceedings of the ACM Symposium on Document Engineering 2023*, 1–4. doi:10.1145/3573128.3609344.

Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzi Li, Shean Wang, and Weizhu Chen. 2021. "LoRA: Low-Rank Adaptation of Large Language Models." Working paper (16 October). doi:10.48550/arXiv.2106.09685.

Kim, Alex G., Maximilian Muhn, and Valeri Nikolaev. 2023. "Bloated Disclosures: Can ChatGPT Help Investors Process Financial Information?" Working paper, Becker Friedman Institute for Economics (26 April). doi:10.48550/arXiv.2306.10224.

Lopez-Lira, Alejandro, and Yueha Tang. 2023. "Can ChatGPT Forecast Stock Price Movements? Return Predictability and Large Language Models." Working paper, University of Florida (6 April). doi:10.2139/ssrn.4412788.

Loughran, Tim, and Bill McDonald. 2011. "When Is a Liability Not a Liability? Textual Analysis, Dictionaries, and 10-Ks." *Journal of Finance* 65 (4): 35–65.

Lu, Yi-Te, and Yintong Huo. 2025. "Financial Named Entity Recognition: How Far Can LLM Go?" Working paper (4 January). doi:10.48550/arXiv.2501.02237.

Malo, P., A. Sinha, P. Korhonen, J. Wallenius, and P. Takala. 2014. "Good Debt or Bad Debt: Detecting Semantic Orientations in Economic Texts." *Journal of the American Society for Information Science and Technology* 65 (4): 782–96. doi:10.1002/asi.23062.

Mehra, Srishti, Robert Louka, and Yixun Zhang. 2022. "ESGBERT: Language Model to Help with Classification Tasks Related to Companies' Environmental, Social, and Governance Practices." Working paper (31 March).

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. "Distributed Representations of Words and Phrases and Their Compositionality." Working paper (16 October). doi:10.48550/arXiv.1310.4546.

Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. "Improving Language Understanding by Generative Pre-Training." OpenAI. www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. "Language Models Are Unsupervised Multitask Learners." OpenAI. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Reimers, Nils, and Iryna Gurevych. 2019. "Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks." Working paper (27 August). doi:10.48550/arXiv.1908.10084.

Tetlock, Paul C. 2007. "Giving Content to Investor Sentiment: The Role of Media in the Stock Market." *Journal of Finance* 62 (3): 1139–68. doi:10.1111/j.1540-6261.2007.01232.x.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. "Attention Is All You Need." In *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–10.

Wang, Meiyun, Kiyoshi Izumi, and Hiroki Sakaji. 2024. "LLMFactor: Extracting Profitable Factors Through Prompts for Explainable Stock Movement Prediction." Working paper (16 June). doi:10.48550/arXiv.2406.10811.

Wang, Shuwei, Ruifeng Xu, Bin Liu, Lin Gui, and Yu Zhou. 2014. "Financial Named Entity Recognition Based on Conditional Random Fields and Information Entropy." In *2014 International Conference on Machine Learning and Cybernetics*, 838–43. doi:10.1109/ICMLC.2014.7009718.

Yang, Hang, Yubo Chen, Kang Liu, Yang Xiao, and Jun Zhao. 2018. "DCFEE: A Document-Level Chinese Financial Event Extraction System Based on Automatically Labeled Training Data." In *Proceedings of ACL 2018, System Demonstrations*, 50–55.