Fine-Tuned Large Language Models and Agent-Based Frameworks for Automated Price Estimation

Hien Nguyen[1],

[1] Independent Researcher, [Wellington, New Zealand]

1 Introduction

In recent years, the emergence of large language models (LLMs) has transformed the landscape of artificial intelligence and natural language processing, enabling machines to understand and generate human language at unprecedented scales. As the capabilities of LLMs continue to expand—spanning applications such as search, summarization, and task automation—there is growing interest in applying these technologies to structured reasoning tasks, including numeric prediction and market analysis. This report presents an exploration into the use of language models for a novel application: price prediction and deal discovery within the e-commerce domain.

The project begins with a data engineering pipeline that preprocesses product metadata and textual reviews from the Amazon Reviews 2023 dataset. This curated dataset is then used to train and benchmark both classical machine learning models and state-of-the-art LLMs, including fine-tuned variants based on GPT-4o and Meta's LLaMA-3.1. In addition to conventional modeling strategies, the report introduces an agentic framework that leverages modular agents for automated deal scanning, contextual price estimation, real-time execution, and ensemble-based decision-making. Through a series of detailed algorithms and evaluation metrics, we assess the effectiveness of fine-tuning, parameter-efficient methods like QLoRA, retrieval-augmented reasoning, and interactive user interfaces. The resulting multi-agent system provides a platform for discovering high-value product deals powered by AI.

The framework and implementation approaches presented in this report are adapted from the techniques outlined in "LLM Engineering - Master AI and LLMs" repository by Edward Donner [1]. The rest of the report is organized as follows: Section 2 introduces the background and various techniques used in Language AI. Section 3 mentions the data preprocessing step. Chapter 4 covers model building and evaluation, starting with model building in Section 4.1, followed by model evaluation in Section 4.2. Chapter 5 discusses the algorithms in the agentic framework and agentic execution. Finally, the paper concludes in Section 6, and ends with Section 7 Appendix.

2 Background

Artificial intelligence (AI) generally describes computer systems designed to perform tasks that emulate aspects of human intelligence, such as understanding speech, translating languages, or interpreting images. Although AI aims to mimic human-like behaviors, the label is sometimes loosely given to simple rule-based systems, such as basic game characters, which do not exhibit true intelligence.

A specialized branch, called Language AI, involves building technologies that can understand, process, and generate human language. The language AI subfield is closely related to natural language processing (NLP) and has evolved alongside the rise of advanced machine learning methods. Language AI encompasses more than just large language models (LLMs)—systems like retrieval engines also play a key role in empowering LLMs with new capabilities [2].

2.1 Text Encoding Methods

LLMs are sophisticated deep learning models trained on vast text datasets, enabling them to complete a variety of language-related tasks, such as text recognition, summarization, translation, prediction, and generation. One of the major challenges in Language AI is that computers struggle with the language's unstructured and nuanced nature, which has driven ongoing research on representing language in structured formats that make it more accessible for computational processing [2].

2.1.1 Bag of Words The bag-of-words technique is an approach used in language AI to represent unstructured text as numerical data. It starts with tokenization which breaks sentences into individual words or tokens. Typically, tokenization splits text at whitespace, but it can be challenging for languages without clear word boundaries, such as Mandarin. Next, the model creates a vocabulary of all unique words and counts the frequency of each word in the text. The result is a vector representation of the text, where each entry reflects how often a word appears, regardless of the word's order or grammar in the sentence. The vectorized form enables machine learning algorithms to process and analyze textual data efficiently [2].

2.1.2 Word2Vec The bag-of-words approach represents text as collections of words while ignoring their meanings and relationships. Word2vec, introduced in 2013, addressed bag-of-words' limitation by creating word embeddings—numeric vector representations that capture the meaning and semantic similarity of words. Using neural networks, word2vec learns these semantic relationships from large text datasets by predicting which words appear near each other in sentences. The process allows for various types of embeddings, such as word and sentence embeddings, which measure similarity more effectively than the bag-of-words approach, which only creates document-level representations [2].

2.1.3 Attention Traditional word2vec models produce fixed embeddings for words regardless of context, meaning a word like "bank" has the same representation whether it refers to a financial institution or a riverbank. This limitation makes it difficult to capture the nuances of word meaning based on context. Neural network approaches such as recurrent neural networks (RNNs) were developed, capable of processing sequences of words by encoding an input sentence into an internal representation and decoding it to generate output. RNNs use word embeddings such as word2vec as initial inputs. However, compressing an entire sentence into a single embedding can be problematic, especially for longer or more complex sentences. The introduction of attention mechanisms in 2014 significantly improved this process by allowing models to focus on relevant words and weigh their importance within context. With attention, RNNs utilize all hidden states of the input sequence, instead of a single embedding, better capturing sequential and contextual relationships in language. Despite enhancing accuracy, attention mechanisms limit the ability to parallelize training [2].

2.1.4 Transformer The foundation of large language models lies in the concept of attention, first introduced in the influential 2017 paper "Attention Is All You Need." The paper proposed the Transformer architecture, which relies entirely on the attention mechanism and eliminates the previously used recurrent neural networks (RNNs). Unlike RNNs, Transformers can be trained in parallel, significantly accelerating the training process. The architecture includes stacked encoder and decoder blocks, remaining autoregressive by generating one word at a time. Each encoder block consists of self-attention and a feedforward neural network, allowing the model to analyze an entire input sequence simultaneously rather than token by token. The self-attention mechanism enables the model to capture relationships between all tokens in a sequence more effectively. The decoder block adds a layer to attend to the encoder's output and uses masked self-attention to ensure each word is predicted without future context. Together, these components form the core of the Transformer, the architecture behind major models like BERT and GPT-1 [2].

2.2 Large Language Models

In 2018, a decoder-only architecture called Generative Pre-trained Transformer (GPT)—now known as GPT-1—was introduced to address generative tasks, following the encoder-only design of BERT. These generative models, particularly the larger variants, are known as large language models (LLMs). LLMs are sequence-to-sequence systems that take in some text and attempt to complete it. However, their true strength emerged when trained as chatbots. By fine-tuning these models, developers created instruct or chat models capable of following directions and responding to user prompts. These generative models are often referred to as completion models. A key component of these models is their context length or context window. The context window means the maximum number of tokens a model can process at once, with larger windows allowing entire documents to be supplied to the model [2].

2.2.1 Tokenization Before a prompt is input into a language model, it is first processed by a tokenizer that splits the text into smaller units called tokens. These tokens are then converted into a sequence of integers, each representing a unique token such as a character, word, or word segment. The tokenizer is essential not only for preparing data to feed into the model but also for decoding the model's output token IDs back into readable text. The way tokenizers break down input depends on three main factors: the tokenization method chosen by the model creator (e.g., byte pair encoding or WordPiece), the tokenizer's configuration, including vocabulary size and special tokens, and the dataset used to train the tokenizer, which influences the vocabulary it builds. As a result, even tokenizers using the same methods and settings can differ depending on whether they are designed for English, code, or multilingual text. Overall, tokenizers perform a dual function, encoding input into tokens for the model and decoding the model's output back into natural language using subword tokenization [2].

2.2.2 Token embeddings Language can be understood as a sequence of tokens, which are smaller units derived from text through tokenization. Training a sufficiently powerful model on a large set of these tokens allows the model to learn complex linguistic patterns and relationships present in the data. A crucial aspect is determining an effective numerical representation for each token so the model can analyze and learn from the patterns efficiently. These numerical representations are known as embeddings, which map tokens into a high-dimensional vector space where semantic and syntactic relationships between tokens become mathematically expressible. In large language models, each token in the vocabulary has a corresponding embedding vector stored in an embedding matrix. Initially random at the start of training, these vectors are refined during training. Thus, the model can perform tasks like language comprehension and generation more effectively. Unlike static representations, modern language models generate contextualized embeddings, meaning the representation of a token can vary depending on the context in which it appears, allowing for nuanced understanding of language. The tokenizer and language model are tightly linked because the model's embeddings correspond exactly to the token vocabulary produced by its tokenizer, making pretrained models dependent on their specific tokenizers to function correctly. Ultimately, embeddings serve as the fundamental numeric building blocks that enable language models to capture meanings and patterns in language and extend their capabilities to many NLP tasks [2].

2.2.3 Text embeddings Text embedding models are designed to represent entire texts, such as sentences, paragraphs, or documents, as a single vector that captures their overall meaning. While traditional language models rely on token embeddings to function, many applications require a broader understanding of longer text segments. These text embeddings can be created in several ways. One of the most common is the averaging of all token embeddings within a piece of text. However, the most effective text embedding models are typically trained with the specific goal of summarizing and capturing the meaning of extended passages. Popular tools like sentence-transformers enable the generation of these high-quality text embeddings using pretrained models [2].

2.2.4 LLMs architecture Generative AI and large language models (LLMs) can produce coherent text by applying Bayes' theorem, which allows them to predict the next most probable word given the preceding words in a prompt. Its capability is rooted in the extensive data on which LLMs are trained, enabling the models to internalize language patterns and context. Hence, they can generate meaningful and relevant text completions. LLMs leverage probabilities learned from their training corpus, ensuring that the text they generate not only makes sense grammatically but also fits the contextual nuances presented by the user's input [3].

At the core of these models is the transformer architecture, designed to address issues with traditional approaches (RNN), especially regarding long-range dependencies, scalability, and computational efficiency. Unlike previous methods, transformers forgo recurrence and convolution in favor of attention mechanisms that allow the model to focus on different parts of the input sequence simultaneously. The self-attention mechanism is central; it evaluates the importance of each input token concerning the others by computing attention scores based on “query,” “key,” and “value” matrices, which are optimized during training. The mechanism helps the model decide which words in a sentence to concentrate on as it processes information [3].

The transformer is structured into an encoder and decoder. The encoder converts input text into vector representations and adds positional encodings to retain word order, then processes these through stacked layers, including multi-head attention, addition and normalization, and feed-forward transformations. The decoder performs a parallel sequence of operations on the output sequence, which is shifted right to train the model to predict subsequent tokens. Both encoding and decoding leverage layers that transform and normalize the data, culminating in linear and softmax operations that yield a final probability distribution over possible next words. The result is a flexible and efficient architecture that forms the foundation of modern LLMs, which has evolved with numerous variations since the original design [3].

2.2.5 OpenAI - GPT4 Over the past year, the development of large language models (LLMs) has accelerated rapidly, with numerous new releases from various organizations, each offering unique capabilities. Some of these models have set new records in scale and sophistication, advancing far beyond previous state-of-the-art technologies, while others focus on specialized tasks with streamlined architectures. Notably, OpenAI’s GPT-4, introduced in March 2023 along with its variant GPT-4 Turbo, stands out as one of the leading models in the field, even as work on GPT-5 is already underway. GPT-4 represents a generative pretrained transformer (GPT) model with a decoder-only architecture, meaning it predicts the next token in a sequence using only a decoder, rather than the more traditional encoder-decoder structure. The design encodes information within the hidden states of the decoder as it processes input, rather than summarizing information explicitly [3].

GPT-4, like its predecessors, was trained using a combination of publicly available and licensed data, though OpenAI has not disclosed the specifics of its training set. Its training also included reinforcement learning from human feedback (RLHF), which helps better align model outputs with human intent. In various benchmarks, such as the Massive Multitask Language Understanding (MMLU) test, GPT-4 has outperformed prior models in multiple languages, showcasing notable advances in common sense reasoning and analytical skills. An important area of progress is its reduced tendency to generate inaccurate information (“hallucinations”), as demonstrated by improved performance on the TruthfulQA benchmark. Additionally, OpenAI emphasized safety and alignment during GPT-4’s development, involving over 50 experts across disciplines, such as AI alignment, privacy, and cybersecurity [3].

2.2.6 Meta - LlaMA Large Language Model Meta AI 3 (LLaMA-3) is the latest family of models developed by Meta, publicly released in April 2024 as open-source models for free commercial and research use. Continuing the lineage of autoregressive models with an optimized, decoder-only transformer architecture, LLaMA-3 is available in two primary sizes: 8 billion and 70 billion parameters, both trained on an expanded dataset exceeding 15 trillion tokens and supporting a context window of 8,192 tokens for improved reasoning, coding, and multi-step instructions [3].

LLaMA-3 also offers instruction-tuned variants, known as LLaMA-3-Instruct, optimized for dialogue and general-purpose conversational scenarios. The models set new benchmarks in open-source language models, outperforming previous versions in natural language understanding and response generation. The tuning process for LLaMA-3-Instruct incorporates a combination of supervised fine-tuning, which is built on over 10 million human-annotated examples and diverse instruction datasets to enhance helpfulness and safety, and reinforcement learning with human feedback (RLHF), alongside techniques like direct preference optimization (DPO) and iterative data quality refinement. LLaMa’s structure alignment ensures that chat-specific versions are more adaptive, safer, and capable across a wide range of conversational contexts compared to base LLaMA-3 [3].

2.3 Fine-tuning LLMs

Fine-tuning is a transfer learning technique that uses a pretrained neural network’s weights as starting points for training on a new, often related, task. Instead of training a model from scratch, fine-tuning leverages knowledge already learned, which is especially useful when the new task has limited data. Typically, it follows an initial feature extraction phase, where the first layers of the network responsible for generic feature detection are kept frozen, while the last few layers, which are more task-specific, are unfrozen and retrained. The approach allows the model to adjust to the specifics of the new dataset without losing general patterns from the original training. In practice, transfer learning often involves adding a new classifier on top of the base model and training only this added layer, whereas fine-tuning updates some or all of the pretrained layers to better fit the new task. In generative AI, such as large language models, fine-tuning adjusts parameters using task-specific datasets, enhancing performance on particular applications like sentiment analysis. The process includes loading the pretrained model and tokenizer, preparing a suitable dataset, adding a task-specific output layer, training, and evaluating the model. Though less demanding than building a model from scratch, fine-tuning large models requires significant computational resources and careful data handling. Its effectiveness depends on how much retraining is done, the similarity between tasks, dataset size, and available resources, making it ideal for adapting models to specialized tasks with sufficient data [3].

2.3.1 Low-rank adaptation (LoRA) Updating all model parameters can improve performance but involves high costs, slow training, and large storage needs. Parameter-efficient fine-tuning (PEFT) methods have been developed to achieve better computational efficiency when fine-tuning pretrained models. One popular PEFT technique is low-rank adaptation (LoRA), which updates only a small subset of parameters, instead of the entire model. LoRA approximates large weight matrices in the model with smaller matrices that, when multiplied, reconstruct the original matrix. This significantly reduces the number of parameters to update, speeding up training and lowering resource usage. For example, reconstructing a 10×10 matrix (100 weights) with two 10×1 matrices (20 weights) greatly improves efficiency. During fine-tuning, only these smaller matrices are updated, while the original large weight matrices remain frozen and combined with the updated smaller ones [2].

2.3.2 Quantized low-rank adaptation (QLoRA)

LoRA's efficiency can be improved by reducing the memory required for the model's original weights before compressing them into smaller matrices. Large Language Model (LLM) weights, expressed as numbers with a specific bit-precision, use more memory as the bit count increases. Lowering the precision (reducing the number of bits) with quantization reduces memory needs, though it may also reduce accuracy. QLoRA—a quantized variant—applies blockwise quantization, grouping similar high-precision weights into blocks for conversion to lower precision, and minimizing the accuracy loss. The QLoRA process, along with normalization, allows for accurate representation of high-precision values using as few as 4 bits, nearly matching the original performance while requiring substantially less memory. QLoRA benefits both training and inference, as quantized models also consume less VRAM [2].

3 Data preprocessing

The Amazon Reviews 2023 dataset is a comprehensive collection containing over 570 million reviews and 48 million items across 33 different categories. Given the vast size of the dataset, which could result in prolonged training times, a smaller subset has been selected for model training and analysis. Specifically, two subsets have been created: a larger one that includes eight categories related to consumer electronics, and a smaller, more focused subset that contains four of those eight categories. The larger dataset has 400,000 items while the smaller dataset has 25,000 items. The smaller dataset will serve as the primary dataset for further fine-tuning of models.

Algorithm 1 outlines a systematic approach for loading and curating Amazon review items tailored for downstream modeling tasks. It begins by loading a specified product category dataset from the "McAuley-Lab/Amazon-Reviews-2023" collection. The dataset is partitioned into manageable chunks, each processed in parallel across multiple worker processes to leverage computational resources and speed up the operation. For each datapoint, the algorithm filters entries based on a valid price range, ensuring that only relevant and realistic products are included. It then performs thorough text processing by concatenating and cleaning multiple textual fields such as descriptions, features, and details, removing irrelevant content and overly complex tokens that might degrade model quality. To maintain consistency and quality for language modeling, the algorithm imposes constraints on both character length and token count, truncating excessively long texts and ensuring minimum length thresholds are met. Finally, it constructs a standardized prompt that combines a question about the product's cost and the cleaned textual content, followed by the rounded price answer. Only items meeting all criteria are marked as included and stored as curated Item objects. The result is a refined dataset of Amazon review items with prompts, suitable for training or evaluating models in tasks related to price inference or product understanding.

Building upon the above Algorithm's approach for item loading and curation, Algorithm 2 implements the methodology at scale, conducting exploratory analysis, rigorous filtering, and tokenization checks across multiple categories. Together, they form a complete pipeline that transforms raw Amazon review data into structured, balanced, and shareable datasets optimized for supervised price prediction modeling.

Algorithm 2 implements a data engineering workflow for building curated datasets of Amazon product reviews to support supervised price prediction modeling. It begins by setting up the environment, loading necessary API keys, and authenticating with HuggingFace Hub. The workflow starts with exploratory data analysis (EDA) on the "Appliances" category from the McAuley-Lab Amazon Reviews 2023 dataset, inspecting item fields (titles, descriptions, features, details, and prices), and computing statistics. Visualizations illustrate distributions of text length and price to understand the data spread.

The pipeline transforms raw datapoints into domain-specific Item objects that encapsulate cleaned and tokenized content alongside price metadata, enforcing inclusion criteria that filter out unsuitable entries. The workflow is then extended to multiple categories—first a smaller subset of four, then a larger set of eight diverse categories—using an ItemLoader abstraction to batch-load and process data in parallel. For the larger dataset, additional sampling strategies balance category representation and price distribution, mitigating over-representation of categories with various items (e.g., Automotive) by weighted random selection.

Datasets are divided into train and test splits with fixed random seeds for reproducibility. A tokenizer from the Meta-LLaMA pretrained family is instantiated to probe how prices are tokenized; specifically, identifying which price values are represented as single tokens versus multiple tokens. Our objective is to ensure that all prices, including the three-digit representations, correspond to a single token, which is why we specifically choose the Meta-LLaMA tokenizer. It provides a significant advantage for price prediction tasks. When each price is mapped to a single token, the language model can better treat prices as atomic, indivisible labels, simplifying the learning process and minimizing the risk of incorrectly splitting or misinterpreting multi-digit prices across several tokens. It ensures that the output space for prices is both compact and consistent, improving model efficiency, interpretability, and accuracy during generation and training, especially for fine-grained tasks where the model must precisely predict or understand exact price values. As a result, alignment between real-world numeric targets and tokenization leads to clearer and more reliable price-conditioned language outputs.

Finally, the curated datasets are converted into HuggingFace Dataset formats containing paired prompts and prices, and pushed to the HuggingFace Hub. Local pickle serialization preserves the train/test splits for reuse.

4 Model building and evaluation

This section aims to develop and evaluate methods for predicting product prices based on textual descriptions from e-commerce data. A modeling pipeline is implemented, incorporating both traditional machine learning techniques and cutting-edge large language models. The following sub-sections outline the key components and workflow of the pipeline, detailing how different approaches are applied and compared.

4.1 Model building

Algorithm 3 provides a pipeline for modeling and evaluating price prediction from e-commerce product descriptions using both traditional machine learning and modern large language models (LLMs). The workflow begins by setting up the project environment, loading necessary credentials, and importing preprocessed training and test datasets. It first establishes a classical baseline: texts are vectorized with Word2Vec, and these embeddings are used to train a Random Forest regressor to predict item prices.

The pipeline implements LLM-based regression of both proprietary and open-source models. For the OpenAI base model, prompts are constructed from product descriptions, and the LLM is used in a zero-shot setting to estimate prices. The workflow then prepares data in a format suitable for LLM fine-tuning, splitting a small subset for training and validation, and orchestrates the fine-tuning process on the OpenAI platform, including monitoring and evaluating the resulting specialized model.

For open-source approaches, pre-tokenized datasets are loaded, and the Meta-LLaMA model family is configured with efficient quantization (such as 4-bit QLoRA). The process manages parameter-efficient fine-tuning using LoRA techniques, tracks experiments with Weights & Biases (https://wandb.ai/site/), and leverages collators and configuration settings to fully utilize the underlying transformer architectures. After training, both the base and fine-tuned models are evaluated on the test data, with results from all approaches compared for benchmarking.

4.2 Model evaluation

The modeling pipeline described above lays the groundwork for assessing predictive performance across a spectrum of classical and modern approaches. The following evaluation compares these methods using standardized error metrics, providing insights into how each model architecture translates its learned representations into real-world price prediction accuracy.

Metrics	Random Forest	GPT-4o	Fine-tuned GPT-4o	LLaMA-3.1	Fine-tuned LLaMA-3.1
MAE	47.40	23.21	23.97	95.49	24.99
RMSLE	0.96	0.52	0.49	0.91	0.52

Table 1. Price comparison performance of Random Forest and LLMs.

The performance comparison (as shown in Table 1 of the evaluated models is based on two key metrics: Mean Absolute Error (MAE), which measures the average size of errors in price predictions (with lower values indicating greater accuracy), and Root Mean Squared Logarithmic Error (RMSLE), which emphasizes proportional differences and is robust to outliers. Among all models, both the base and fine-tuned versions of GPT-4o achieve the lowest errors, demonstrating strong, consistent performance in price estimation tasks. In contrast, the Random Forest and base LLaMA-3.1 models perform substantially worse, with much higher errors. Notably, the fine-tuned LLaMA-3.1 model, trained using QLoRA, shows a dramatic reduction in error compared to its base counterpart, nearly matching the performance of GPT-4o. However, fine-tuning GPT-4o yields only a modest change, with a slight performance decrease compared to its base version.

The slight underperformance of the fine-tuned GPT-4o relative to the base model can be attributed to several potential factors, including possible insufficient or noisy fine-tuning data, overfitting due to a small or non-representative training set, distribution mismatches between pretraining, fine-tuning, and evaluation data, suboptimal hyperparameter choices during fine-tuning, and catastrophic forgetting of broadly useful pretraining knowledge [4]. The modest difference in MAE (23.21 vs. 23.97) indicates that while fine-tuning did not significantly harm performance, it also failed to yield substantial additional benefits.

In contrast, the impact of QLoRA-based fine-tuning on LLaMA-3.1 is much more pronounced. The base LLaMA-3.1, being a general-purpose language model, lacks domain-specific understanding of price prediction. Fine-tuning on targeted prompts enables the model to specialize, learning both the task format and the numeric relationships between text and prices. QLoRA offers parameter-efficient tuning, allowing the model to adapt with relatively little compute and data while retaining its foundational capabilities [5]. The sharp drop in RMSLE further suggests that the model now handles both small and large price values more proportionally. The improvement is particularly dramatic for LLaMA because, unlike GPT-4o, it begins with little relevant pretraining and fine-tuning, thus closing a much larger gap in task-specific capability.

Fine-tuning large language models is highly advantageous when the base model has not been previously exposed to the specific prediction task, as demonstrated by LLaMA-3.1. However, for state-of-the-art models such as GPT-4o, which already generalize well to structured tasks, the addition of fine-tuning must be carefully considered. Sub-optimal execution of fine-tuning may yield little benefit or even slightly harm performance. Notably, parameter-efficient approaches such as QLoRA can rapidly and cost-effectively adapt open-source models, achieving substantial gains on specialized tasks [5].

5 Agentic framework and execution

The agentic framework provides a modular and orchestrated system for automated deal discovery and price analysis in the retail domain. By leveraging a hierarchy of specialized agents, the framework enables integration of information retrieval, language model inference, contextual reasoning, predictive ensembling, and workflow planning to surface high-value retail opportunities in real time.

5.1 Agentic framework

Algorithms 4 and 5 implement a coordinated, multi-stage process for discovering valuable retail deals, estimating their true prices, and identifying the most attractive opportunities. The system is constructed from a hierarchy of agents, each with a specialized role, working together to automate deal analysis from data ingestion to actionable recommendations.

At the foundation, the ScannerAgent is responsible for retrieving new product deals from sources such as RSS feeds, filtering out any entries that have already been processed, thus avoiding redundancy. The ScannerAgent then summarizes each eligible deal, emphasizing detailed product descriptions and clearly stated prices, while disregarding ambiguous price mentions related to discounts or reductions. The agent reformulates these descriptions to focus strictly on product characteristics rather than promotional language, and utilizes an advanced language model (such as GPT-4o) to ensure all selected deals provide explicit numeric prices. The agent ultimately returns a refined list of five high-potential deals, each accompanied by a comprehensive description and verified price.

Next, the SpecialistAgent leverages remote, fine-tuned large language models hosted via Modal to estimate prices based on the provided product descriptions. This agent supports both GPT-based and LLaMA-based models, calling the appropriate one as specified. By providing flexible model access, it enables comparative or ensemble-based approaches to price prediction, offering robustness to diverse product domains or input styles.

The FrontierAgent extends price estimation capabilities by incorporating retrieval-augmented generation. It encodes new product descriptions, queries a vector database for the most similar historical items (along with their known prices), and builds a context-enriched prompt, presenting both the new description and references to similar products. The prompt is submitted to an LLM (via OpenAI), which uses the additional context to produce a more informed price estimate. The agent carefully extracts a numeric answer from the model's response, enhancing interpretability and reliability.

The EnsembleAgent synthesizes predictions from the SpecialistAgent (both are fine-tuned GPT and LLaMA models) and the FrontierAgent, combining their outputs along with minimum and maximum values into a feature vector. It is then processed by a pre-trained linear regression model, trained offline to weight the contributions of each predictor for optimal final price estimates. The EnsembleAgent enables robust prediction by aggregating diverse model perspectives, mitigating the risk of individual model bias or error.

At the top level, the PlanningAgent orchestrates the full end-to-end workflow. It initiates deal discovery via the ScannerAgent, applies the EnsembleAgent to provide accurate market price estimates, and calculates the discount by comparing the deal's listed price to its predicted fair value. The PlanningAgent packages each considered deal into an Opportunity object, summarizing its description, predicted market price, and discount margin. Finally, it ranks all surfaced opportunities, selecting the deal offering the greatest discount above a predetermined threshold—thereby surfacing only high-value recommendations.

Altogether, the algorithms exemplify modular, multi-agent system design, in which specialized components—focusing respectively on information retrieval, language model inference, retrieval-augmented reasoning, predictive ensembling, and workflow planning—combine to deliver an automated pipeline for the real-time discovery and analysis of exceptional deals.

Metrics	GPT-4o	Frontier Agent	Ensemble Agent
MAE	23.21	20.08	22.59
RMSLE	0.52	0.47	0.43

Table 2. Price comparison performance between LLMs.

Table 2 presents a comparative evaluation of three different approaches to price prediction using large language models and agent-based reasoning. The baseline model is GPT-4o, evaluated in a zero-shot setting without any external context or fine-tuning. It is compared against two advanced agent-based models: the Frontier Agent, which uses retrieval-augmented generation by incorporating similar past items as context, and the Ensemble Agent, which aggregates multiple predictions from various models using a linear regression-based combination strategy.

From the table, we observe that the Frontier Agent achieves the best MAE (Mean Absolute Error) of 20.08, outperforming both the base GPT-4o (23.21) and the Ensemble Agent (22.59). In terms of RMSLE (Root Mean Squared Logarithmic Error), which is sensitive to relative differences and penalizes under/overestimation proportionally, the Ensemble Agent performs best overall at 0.43, followed closely by the Frontier Agent at 0.47, both improving upon GPT-4o's 0.52.

The superior performance of the agent-based approaches compared to the GPT-4o baseline can be attributed to their access to contextual or complementary information. While GPT-4o is a powerful general-purpose model, its predictions rely solely on the prompt provided. In contrast, the Frontier Agent enhances its input by retrieving similar past examples from a vector database, effectively "grounding" the prompt in relevant price contexts. The retrieval-augmented prompting allows the LLM to reason with anchored numerical examples, which improves price estimation in cases where the model might otherwise hallucinate or lack confidence about pricing.

Interestingly, although the Ensemble Agent aggregates multiple sources (including predictions from fine-tuned GPT-based, fine-tuned LLaMA-based, and Frontier models), it does not outperform the Frontier Agent in terms of MAE. It may be due to the linear regression ensemble slightly diluting the high-confidence signal of the Frontier Agent by incorporating weaker predictions from other models. Additionally, ensembling methods are only as strong as the diversity and accuracy of their constituent predictors. If certain sub-models (e.g., one of the SpecialistAgents) introduce consistent bias or noise, the combined output may not yield optimal improvements, especially in absolute error metrics like MAE.

5.2 Agentic execution and logging interface

Algorithm 6 implements an interactive pipeline for evaluating and surfacing the most valuable deals using a real-time agentic system, complete with live monitoring via a graphical user interface. At its core, the system instantiates an agent framework, which is responsible for scanning, analyzing, and recommending product deals. Responsive logging and visualization tools are also included to support both user interaction and transparent monitoring, using Gradio and Plotly.

Upon initialization, the algorithm sets up a thread-safe log queue and attaches a custom logging handler, ensuring that all outputs, status updates, and events produced by the various agents are captured and can be streamed to the interface. The agent framework orchestrates multiple stages of deal evaluation, with each cycle involving the scanning of new opportunities, the estimation of fair prices, and the calculation of discounts relative to current deal prices.

A central feature of this pipeline is the ability to feed back real-time information to the user. As the system progresses through its deal evaluation loop, it continuously retrieves new log messages, formats the latest updates into easy-to-read HTML segments, and pushes them to the web interface. When a candidate deal passes the predefined thresholds for value and discount, the system generates an interactive visualization—typically in the form of a Plotly bubble chart—that highlights the top-ranked opportunity and provides an at-a-glance summary of its relevant attributes.

Additionally, the user interface itself is built using a high-level framework, allowing for seamless user inputs (such as annotated feedback or scenario prompts) and dynamic output updates. The design is intended for robustness and responsiveness: logging and plotting are non-blocking and update in near real-time, while deal evaluation cycles continue in the background until a satisfactory recommendation is produced.

6 Conclusion

This report presented a framework for price prediction and opportunity detection in e-commerce using both classical machine learning and large language models (LLMs). Starting with systematic data curation from the Amazon Reviews 2023 dataset, we established a robust preprocessing pipeline to clean, tokenize, and transform raw product data into high-quality training samples. Leveraging this dataset, we evaluated multiple modeling approaches—from Word2Vec embeddings with Random Forest regression, to state-of-the-art LLMs like GPT-4o and Meta's LLaMA-3.1, both in their base and fine-tuned forms.

The results show that fine-tuning leads to significant performance gains when the base model lacks task-specific exposure, as illustrated by the sharp performance increase in LLaMA-3.1 after applying QLoRA-based fine-tuning. However, for models like GPT-4o that already generalize well to structured tasks, fine-tuning offers limited or marginal benefit, highlighting the need for informed application of adaptation techniques. The importance of tokenizer design, token alignment, and efficient modeling strategies—particularly QLoRA and LoRA—was also emphasized for optimizing performance in resource-constrained settings.

To further enhance prediction reliability, we implemented an agentic framework comprising specialized modules: the ScannerAgent for deal ingestion, the Specialist and Frontier Agents for individual and context-enhanced price estimation, the EnsembleAgent for model aggregation, and the PlanningAgent for decision-making. Evaluation results demonstrate that agent-based reasoning, especially via retrieval-augmented generation, outperforms conventional LLM inference in accuracy and robustness. Moreover, the Frontier Agent consistently delivered the lowest prediction error, while the Ensemble Agent offered competitive and stable performance.

Finally, the interactive execution environment—enabled through a logging interface built with Gradio and Plotly—allowed for real-time feedback and user-centric analysis, making the system both transparent and adaptable for live deployment. Overall, this work highlights the power of combining curated data, fine-tuned open-source models, and modular agentic reasoning to construct intelligent systems capable of providing accurate and explainable price predictions in dynamic retail environments.

7 Appendix

Algorithm 1 Parallelized Item Loading and Curation from Amazon Reviews

```
 1: Input: Category name name, number of worker processes workers
 2: Output: Curated list of Item objects with category and prompt fields

 3: Parameters:
 4:     $CHUNK\_SIZE \leftarrow 1000$
 5:     $MIN\_PRICE \leftarrow 0.5$
 6:     $MAX\_PRICE \leftarrow 999.49$
 7:     $MIN\_TOKENS \leftarrow 150$
 8:     $MAX\_TOKENS \leftarrow 160$
 9:     $MIN\_CHARS \leftarrow 300$
10:     $CEILING\_CHARS \leftarrow MAX\_TOKENS \times 7$

11: Load dataset:
12:     dataset ← load_dataset("McAuley-Lab/Amazon-Reviews-2023",
13:             "raw_meta_" + name, split="full")

14: Partition dataset into chunks of size $CHUNK\_SIZE$
15: for all chunk in dataset do
16:     In parallel using workers processes:
17:     for all datapoint in chunk do
18:         if $MIN\_PRICE \leq price \leq MAX\_PRICE$ then
19:             Parse and clean text fields:
20:             - Concatenate and scrub description, features, details
21:             - Remove irrelevant patterns/tokens from details and description
22:             - Remove overly long words containing numbers
23:             if length of contents > $MIN\_CHARS$ then
24:                 Truncate contents to $CEILING\_CHARS$
25:                 Encode text using Llama tokenizer
26:                 if token count > $MIN\_TOKENS$ then
27:                     Truncate to $MAX\_TOKENS$ tokens
28:                     Decode text
29:                     Construct prompt:
30:                         "How much does this cost to the nearest dollar?"
31:                         [text]
32:                         Price is $[rounded price].00
33:                     Mark item as included
34:                     Store processed Item with title, price, category, prompt, tokens
35:                 end if
36:             end if
37:         end if
38:     end for
39: end for
40: Collect all included Items from parallel workers
41: return List of finalized Item objects with prompts
```

Algorithm 2 Data Loading, Exploration, and Preparation Pipeline

```
1: Load environment variables and authenticate with HuggingFace Hub
2: Set project and output directories
3: Load Amazon review dataset for selected category/categories
4: Perform exploratory data analysis:
5:     - Inspect sample datapoints
6:     - Count items with valid prices
7:     - Compute and plot distributions of text lengths and prices
8: Create Item objects for datapoints with valid prices, applying inclusion filters
9: Visualize token count and price distributions for curated items
10: For multiple categories, use ItemLoader to batch load and curate items
11: Balance large datasets by sampling prices and categories with weighted probabilities
12: Split datasets randomly into training and testing subsets with fixed seeds
13: Analyze price frequency distribution in training data
14: Load and test tokenizer to evaluate price tokenization patterns
15: Convert datasets to HuggingFace Dataset format and push to Hub
16: Serialize datasets locally for efficient reuse
```

Algorithm 3 Price Prediction Modeling Pipeline

```
1: Load environment variables and API keys; authenticate with HuggingFace Hub
2: Set project and output data directories
3: Load preprocessed training and test datasets from pickled files

4: Baseline: Word2Vec + Random Forest
5: Prepare training documents and prices, using test prompts to avoid leakage
6: Preprocess documents into word tokens
7: Train Word2Vec embeddings on tokenized documents
8: Represent each document as the mean of its word vectors
9: Fit Random Forest Regressor on document vectors and prices
10: Evaluate model performance on test data

11: LLM Baselines and Fine-tuning: OpenAI
12: Prepare messages for LLM-based price prediction
13: Use base LLM (e.g., GPT-4) for zero-shot price prediction; evaluate performance
14: For fine-tuning, prepare training and validation sets from first 250 samples
15: Write message-formatted data to JSONL files
16: Upload files and initiate fine-tuning job with OpenAI
17: Monitor fine-tuning status; retrieve fine-tuned model name
18: Create predictor from fine-tuned model; evaluate on test set

19: Open-Source Baseline: Meta-LLaMA
20: Load dataset from HuggingFace repository
21: Prepare model-specific constants (e.g., quantization, max sequence length)
22: Configure and load quantized LLaMA base model and tokenizer
23: Prepare prediction interface using open source LLM; evaluate on test set

24: Fine-tuning LLaMA (QLoRA)
25: Set LoRA and QLoRA hyperparameters (rank, alpha, dropout, target modules)
26: Prepare supervised fine-tuning configuration (epochs, batch size, learning rate, optimizer, logging)
27: Initialize Weights & Biases for experiment tracking if enabled
28: Prepare data collator and templates for completion-based price prediction
29: Configure and instantiate PEFT and SFTTrainer with model, dataset, LoRA, and collator
30: Run fine-tuning process, pushing checkpoints and the final model to HuggingFace Hub
31: Evaluate fine-tuned predictor on test set
32: Load trained LLaMA model using PEFT from HuggingFace Hub
```

Algorithm 4 Intelligent Deal-Analysis Agents - Scanner Agent, Specialist Agent and Frontier Agent

```
1: Initialize all agents: ScannerAgent, SpecialistAgent, FrontierAgent.

2: ScannerAgent: Deal Scanning and Selection
     • Load new deals from RSS feeds, filtering out any deals in memory.
     • For each scraped deal, generate a summarized product description and check if a clear price is stated.
     • Send rephrased summaries as prompts to OpenAI API, receive structured JSON of top 5 deals with detailed descriptions and prices.
     • Return only deals with reliable, numeric prices above zero.

3: SpecialistAgent: LLM-Based Price Estimation
     • For a given product description, select the prediction model ("gpt" or "llama").
     • Call the appropriate remote fine-tuned LLM (deployed via Modal) to obtain a price estimate.
     • Return the predicted price.

4: FrontierAgent: Contextual LLM Price Estimation with Retrieval
     • Given a product description, embed it and search a vector database for 5 most similar items with known prices.
     • Construct a contextual prompt including similar products and prices.
     • Query OpenAI LLM using this context-enhanced prompt to estimate the new price.
     • Extract and return the predicted price from the model's output.
```

Algorithm 5 Intelligent Deal-Analysis Agents - Ensemble Agent and Planning Agent

```
1: Initialize all agents: ScannerAgent, SpecialistAgent, FrontierAgent, EnsembleAgent, PlanningAgent.

2: EnsembleAgent: Model Aggregation for Robust Price Estimation
     • For each product description, request price estimates from:
         – GPT-based SpecialistAgent
         – LLaMA-based SpecialistAgent
         – FrontierAgent
     • Form a feature vector using these predictions, as well as their minimum and maximum.
     • Use a pre-trained Linear Regression model to compute the final ensemble price prediction.
     • Return the ensemble price estimate.

3: PlanningAgent: Deal Opportunity Evaluation Workflow
     • Use ScannerAgent to surface up to 5 promising new deals not previously seen.
     • For each deal, use EnsembleAgent to estimate likely true market price.
     • Calculate the discount by comparing the estimated market price to stated deal price.
     • Create an Opportunity object for each deal, capturing estimated price and discount.
     • Select the deal with the highest discount exceeding a defined threshold.
     • Return that Opportunity as the best deal, or None if no qualifying deals are found.
```

Algorithm 6 Interactive Execution and Logging of Deal Evaluation Agent

```
1:  Initialize a log queue and attach a custom QueueHandler to Python's logging system
2:  Create a DealAgentFramework instance to manage deal-sourcing and analysis agents
3:  Define helper function html_for(log_data) to convert recent logs to HTML for Gradio display
4:  function EVALUATE(feedback)
5:      Reset the agent framework
6:      Initialize an empty list to store log strings
7:      while True do
8:          Read logs from the queue (non-blocking)
9:          Append logs to log_data and display the latest via html_for()
10:         Run one planning cycle from framework.run()
11:         If no opportunity is returned, wait and continue
12:         If a valid opportunity is found:
13:         - Display the opportunity description
14:         - Exit loop
15:     end while
16: end function
17: Define Gradio interface demo with a textbox input and two visual output sections
18: Wire the interface to the evaluate function and run it with demo.launch()
```

References

[1] Ed Donner. *llm_engineering: Repo to Accompany My Mastering LLM Engineering Course*. `https://github.com/ed-donner/llm_engineering`. GitHub repository. 2024. (Visited on 07/21/2025).

[2] Jay Alammar and Maarten Grootendorst. *Hands-On Large Language Models: Language Understanding and Generation*. Expected publication: September 2024. O'Reilly Media, Inc., 2024, p. 428.

[3] Valentina Alto. *Building LLM Powered Applications*. Create intelligent apps and agents with large language models. Packt Publishing, May 2024, p. 342.

[4] Scott Barnett et al. "Fine-Tuning or Fine-Failing? Debunking Performance Myths in Large Language Models". In: *arXiv preprint arXiv:2406.11201*. Available at https://arxiv.org/abs/2406.11201. 2024.

[5] Tim Dettmers et al. "QLoRA: Efficient Finetuning of Quantized LLMs". In: *arXiv preprint arXiv:2305.14314* (2023). https://arxiv.org/abs/2305.14314.