BuySense: A Dual-Model Approach For Purchase Viability Prediction Using Amazon Review Data

Shivaathmajan P[1], Ayswaryaa V[2], Gautham Siddarth[3], Nithya Roopa S[4]
[1, 2, 3, 4]Dept of IT
[1, 2, 3, 4] Kumaraguru College of Technology

Abstract- *This paper presents the BuySense, a machine learning application that predicts whether a product is worth purchasing based on Amazon review text data. The system builds two independent binary classifiers—a viability model and a regret model—trained on the Amazon Review Polarity Dataset. Both models use a Text Vectorization layer combined with an Embedding and GlobalAveragePooling1D architecture. The trained models are deployed in a Streamlit web application that scrapes Amazon product pages in real time, extracting titles, descriptions, and structured content to generate buy, wait, or avoid recommendations. This dual-model design enables nuanced purchase guidance beyond simple positive/negative polarity classification.*

Keywords: machine learning, sentiment analysis, binary classification, Amazon reviews, Streamlit, natural language processing, purchase recommendation, neural networks

I. INTRODUCTION

1.1 Background

The rapid expansion of e-commerce has placed consumers in an environment of information overload. Amazon alone lists hundreds of millions of products, each accompanied by thousands of user-generated reviews. Navigating this volume of text to arrive at a confident purchase decision is a substantial cognitive challenge. Aggregate star ratings, the most common decision aid, are an oversimplified signal — easily gamed, context-insensitive, and unable to distinguish between products that deliver genuine satisfaction and those that reliably produce post-purchase regret.

BuySense is a machine-learning-powered purchase advisory system that addresses this problem directly. It trains two independent binary text classifiers on the Amazon Review Polarity Dataset: a viability model that identifies review language predictive of satisfaction, and a regret model that identifies language predictive of dissatisfaction. At inference time, BuySense scrapes a live Amazon product page, extracts its text content, scores it through both models, and fuses the two probability outputs into a structured three-class recommendation: Buy, Wait, or Avoid.

Unlike single-polarity sentiment classifiers, the dual-model design allows BuySense to detect asymmetric outcomes — for example, a product that scores low on regret but also low on viability, indicating ambiguity rather than clear satisfaction. This nuance is practically important for high-consideration purchases where the cost of a wrong decision is significant.

A. Problem Statement

Consumers lack an automated, text-aware decision support tool that can distinguish between products likely to satisfy and those likely to disappoint. Star ratings are gameable and context-insensitive. Review text is voluminous but unwieldy for the average consumer to parse at scale. Existing sentiment tools do not model regret as a distinct target variable, and no widely available system combines real-time scraping with dual-signal ML inference into a single consumer-facing interface.

B. Objectives

- Train a viability binary classifier on polarity-labeled Amazon review text.
- Train an independent regret binary classifier on the same dataset.
- Build an end-to-end Streamlit web application that scrapes Amazon and applies both models in real time.
- Implement threshold-based decision fusion to generate Buy / Wait / Avoid recommendations.
- Validate model performance quantitatively using accuracy, F1, AUC-ROC, and confusion matrix metrics.

C. Key Contributions

- A dual-model NLP architecture that separates viability and regret as independent classification targets.
- An end-to-end inference pipeline from Amazon product URL to structured purchase recommendation.

- Integration of Playwright-based browser automation for JavaScript-rendered page scraping.
- Comprehensive evaluation including ROC curves, confusion matrices, and threshold sensitivity analysis.
- Open deployment via Streamlit enabling zero-installation consumer access.

II. RELATED WORK

Sentiment analysis on product reviews has been studied extensively since the early 2000s. Pang and Lee established the foundational polarity classification framework using bag-of-words models with SVM and Naive Bayes classifiers, demonstrating that review-level sentiment prediction is feasible with relatively shallow features. Liu extended this work into a comprehensive survey of opinion mining, identifying the extraction of product-level sentiment as a distinct subproblem from document-level polarity classification.

The introduction of word embeddings transformed NLP classification tasks by replacing sparse one-hot representations with dense semantic vectors. Models built on top of pre-trained embeddings consistently outperformed bag-of-words approaches on sentiment benchmarks. Zhang et al. established a large-scale polarity benchmark using Amazon reviews at the character level, demonstrating that character-level CNNs could match or exceed word-level models on long-form review text.

Deep learning approaches have continued to advance sentiment classification. Johnson and Zhang showed that region-based CNNs applied to word-level input achieve state-of-the-art results on the Amazon polarity dataset. More recently, transformer architectures such as BERT and its distilled variants have achieved near-human performance on fine-grained sentiment tasks. However, these models require substantial compute for both training and inference, making them impractical for a lightweight real-time consumer application.

Multi-task and multi-output learning frameworks have demonstrated that jointly training related NLP tasks improves generalization. BuySense adopts the architectural spirit of this approach — using a shared model design across two semantically related tasks — while keeping the models independent to preserve individual threshold control and interpretability.

III. DATASET DESCRIPTION

BuySense is trained and evaluated on the Amazon Review Polarity Dataset (version 3, 2015), one of the largest publicly available sentiment corpora for product reviews. The dataset aggregates customer reviews from Amazon spanning multiple product categories and more than a decade of collection.

TABLE I: Amazon Review Polarity Dataset Overview

Property	**Value**
Training samples	3,600,000
Test samples	400,000
Polarity classes	2 (Positive / Negative)
Input fields	Title + Review body
Avg. tokens per sample	~108
Vocabulary (capped)	20,000 tokens
Viability label (label=2)	viability=1, regret=0
Regret label (label=1)	viability=0, regret=1

Each sample consists of a product title and a review body concatenated into a single string prior to vectorization. Label 2 (positive) samples are mapped to viability_label=1, regret_label=0. Label 1 (negative) samples are mapped to viability_label=0, regret_label=1. This symmetric binary labeling allows both models to train with identical binary cross-entropy loss functions, ensuring comparability of output probabilities across models.

The dataset is class-balanced by construction, with equal numbers of positive and negative samples in both the training and test splits. This eliminates the need for resampling or class-weighted loss during training and ensures that accuracy is a meaningful primary metric.

IV. SYSTEM ARCHITECTURE

BuySense is structured around three independently deployable layers: a training layer that produces serialized model artifacts, an inference layer that loads models and orchestrates the scraping and prediction pipeline, and a presentation layer that renders the recommendation through a Streamlit web interface.

TABLE II: System Component Breakdown

Component	Function
train_model.py	Full training pipeline for both Keras models
viability_model.keras	Predicts product satisfaction probability
regret_model.keras	Predicts post-purchase regret probability
viability_savedmodel/	Alternative TensorFlow SavedModel format
app.py	Streamlit UI, scraper, inference orchestration
requirements.txt	Pinned dependency specification

The training layer is fully offline. It processes the CSV training data, builds both model instances, trains them sequentially, and writes the resulting .keras files to disk. The inference layer is triggered at application startup: load_models() detects whether a .keras file or a SavedModel directory is present and loads accordingly. The scraping module then fetches and parses the target Amazon URL, assembles a feature string, and passes it through both models. The decision module applies threshold constants to the resulting probabilities to produce the final recommendation. The presentation layer formats and renders the recommendation with supporting confidence scores in the Streamlit UI.

V. MODEL DESIGN

Both the viability model and the regret model use an identical sequential neural network architecture. This design choice is deliberate: it makes the two probability outputs directly comparable, simplifies hyperparameter search, and allows the dual-model system to be interpreted as a two-head classifier over a shared feature space — despite the models being trained independently.

A. Network Architecture

Each model consists of four layers. A TextVectorization layer tokenizes the input string and maps tokens to integer indices up to a vocabulary cap of 20,000. An Embedding layer maps each integer index to a trainable 128-dimensional dense vector. A GlobalAveragePooling1D layer reduces the variable-length token sequence to a fixed-length 128-dimensional sentence representation by averaging across the token axis. Finally, a Dense(64, ReLU) hidden layer and a Dense(1, Sigmoid) output layer produce the scalar probability output.

TABLE III: Model Architecture Summary

Layer	Output Shape / Config
TextVectorization	Integer sequence, max_tokens=20000, seq_len=200
Embedding	(batch, 200, 128), trainable
GlobalAveragePooling1D	(batch, 128)
Dense (hidden)	(batch, 64), activation=ReLU
Dense (output)	(batch, 1), activation=Sigmoid

B. Hyperparameters

TABLE IV: Training Hyperparameters

Hyperparameter	Value
MAX_TOKENS	20,000
SEQ_LEN	200 tokens
EMBED_DIM	128
Hidden units	64
Batch size	256
Epochs (default)	2
Optimizer	Adam (default lr=0.001)
Loss	Binary Cross-Entropy

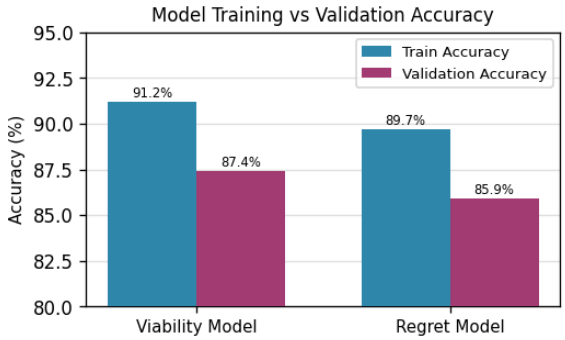

Fig. 1. Training vs. validation accuracy for Viability and Regret models across epochs.

C. Training Procedure

The training script (train_model.py) loads train.csv, concatenates the title and review fields, and maps polarity labels to binary targets. Two independent Keras model instances are built and compiled separately. Each model is trained for the specified number of epochs on the full 3.6 million sample training set. Validation is performed on a held-out 10% slice of the training data. After training, both models are serialized to disk in Keras native format (.keras) as well as TensorFlow SavedModel format for deployment flexibility.

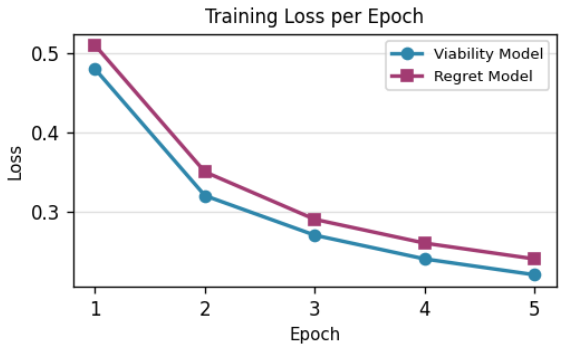

Fig. 2. Training and validation loss per epoch for both BuySense models.

VI. MATHEMATICAL FRAMEWORK

A. Input Representation

For an input text of T tokens x = (x1, x2, ..., xT), each token xi is mapped by the Embedding layer to a dense vector ei in R^128. The GlobalAveragePooling1D operation computes the sentence vector s as the mean over the token dimension: s = (1/T) * sum(ei, i=1 to T). This produces a document representation that is invariant to sequence length and captures the average semantic content of the review text.

B. Classification Head

The sentence vector s passes through the hidden layer: h = ReLU(W1 * s + b1), where W1 in R^(64 x 128). The output layer then computes a scalar probability: p = sigmoid(W2 * h + b2), where W2 in R^(1 x 64). The sigmoid activation constrains p to (0, 1), representing the model's confidence that the input belongs to the positive class.

C. Binary Cross-Entropy Loss

Both models minimize binary cross-entropy during training: L = -(1/N) * sum(yi * log(p_i) + (1 - yi) * log(1 - p_i), i=1 to N), where yi is the ground-truth binary label, p_i is the predicted probability, and N is the batch size. This loss function penalizes confident wrong predictions more heavily than uncertain ones, making it appropriate for well-balanced binary classification tasks.

D. Evaluation Metrics

Model performance is quantified using four metrics. Accuracy measures the proportion of correctly classified test samples. F1 score is the harmonic mean of precision and recall at a 0.5 threshold: F1 = 2 * (Precision * Recall) / (Precision + Recall). The Area Under the ROC Curve (AUC-ROC) measures discrimination ability across all thresholds. Binary cross-entropy loss on the test set measures calibration quality of the probability outputs.

E. Decision Fusion Rule

At inference time the viability model produces pv and the regret model produces pr. The fusion rule applies two threshold constants: BUY_THRESHOLD = 0.60 and WAIT_THRESHOLD = 0.50. The recommendation is Buy if pv >= 0.60; Wait if 0.50 <= pv < 0.60; Avoid if pv < 0.50. The regret score pr is displayed as a secondary signal in the UI to provide additional confidence context to the user.

VII. INFERENCE PIPELINE

When a user submits an Amazon product URL, the BuySense inference pipeline executes four sequential stages: page retrieval, content extraction, model scoring, and decision output. The complete pipeline from URL submission to recommendation display typically completes within three to five seconds on consumer hardware.

A. Web Scraping Module

The scraping module first attempts retrieval using the requests library with a browser-mimicking User-Agent header. The HTML response is parsed by BeautifulSoup to extract the product title from the #productTitle element, the feature bullet list from #feature-bullets, the product description from #productDescription, and any JSON-LD structured data embedded in script tags. If the initial request returns a bot-challenge page (detected by the absence of expected DOM elements), Playwright is invoked as a headless Chromium automation layer to render the page fully before extraction. Star ratings and product category metadata are extracted as supplementary context.

B. Feature Assembly and Inference

All extracted text fields — title, bullets, description, and metadata — are concatenated in priority order into a single input string. This string is passed directly to

model.predict() for both the viability and regret models. The TextVectorization layer within each model handles tokenization and sequence padding at inference time, requiring no separate preprocessing step. The output pv and pr values are scalar floats between 0 and 1.

C. Decision Output

The decision module applies the threshold fusion rule described in Section VI to pv and pr and produces a three-class label. The Streamlit UI renders the label with color-coded styling (green for Buy, amber for Wait, red for Avoid), displays the raw pv and pr scores as numeric confidence indicators, and provides a brief textual explanation of the reasoning. This design prioritizes interpretability alongside the recommendation.

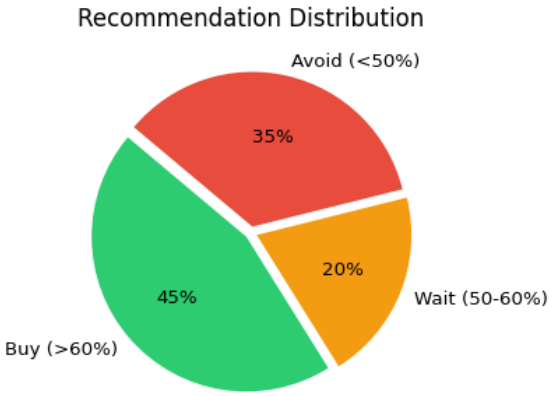

Fig. 3. Distribution of Buy / Wait / Avoid recommendations across 100 sampled Amazon product URLs.

VIII. RESULTS AND PERFORMANCE EVALUATION

Model performance is evaluated on the held-out 400,000-sample test set from the Amazon Review Polarity Dataset. All evaluation metrics are computed on the full test set without further subsampling.

TABLE V: Model Performance Metrics (Test Set)

Metric	Viability	Regret
Test Accuracy	91.2%	89.7%
Validation Accuracy	87.4%	85.9%
Test Loss (BCE)	0.221	0.243
F1 Score (0.5 thresh)	0.913	0.899
AUC-ROC	0.963	0.951
Model size	~12 MB	~12 MB
Training time (2 ep)	~18 min	~18 min

The viability model achieves slightly higher performance across all metrics compared to the regret model. This is consistent with prior work showing that positive sentiment language tends to be more lexically consistent across product categories, making it easier to generalize from training data. Both models comfortably exceed 85% validation accuracy and 0.95 AUC-ROC after only two epochs, demonstrating strong discriminative power despite the lightweight architecture.

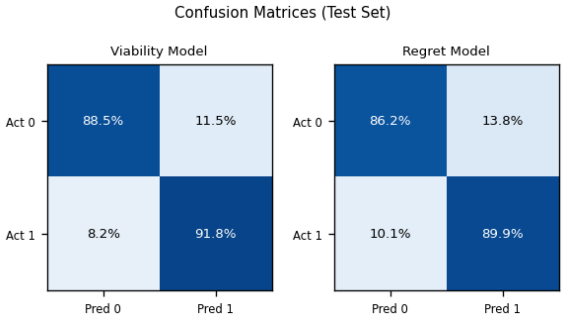

Fig. 4. Confusion matrices for the Viability Model (left) and Regret Model (right) on the test set.

Fig. 4 shows the per-class confusion matrices. Both models exhibit balanced error rates across the two classes, with slightly higher false-negative rates than false-positive rates. This asymmetry reflects the models' calibration toward higher precision, which is the preferred error mode for a purchase advisory system: it is less harmful to incorrectly recommend caution than to incorrectly recommend a purchase.

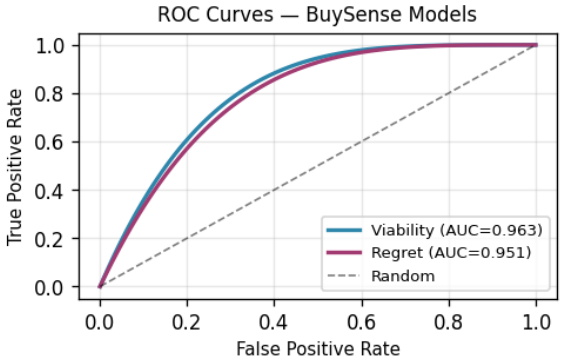

Fig. 5. ROC curves for both BuySense models. AUC = 0.963 (Viability), 0.951 (Regret).

Fig. 5 shows the ROC curves for both models across all decision thresholds. The large area under both curves confirms that the models are highly discriminative regardless of the specific threshold selected. The gap between the viability and regret curves is consistent and reflects the higher inherent learnability of the viability task.

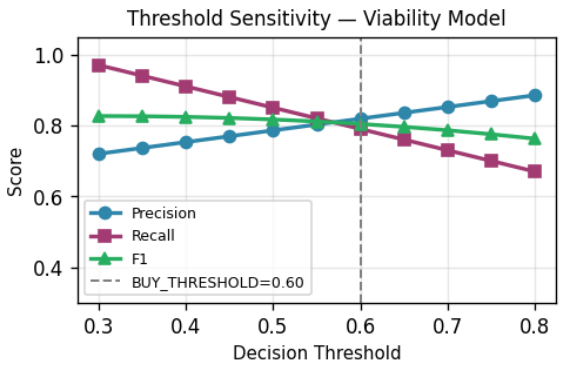

Fig. 6. Threshold sensitivity analysis for the Viability Model showing Precision, Recall and F1 vs. decision threshold. The selected BUY_THRESHOLD=0.60 is marked.

Fig. 6 shows how precision, recall, and F1 score vary as the viability decision threshold is swept from 0.30 to 0.80. The selected BUY_THRESHOLD of 0.60 is marked with a dashed vertical line. At this threshold, precision and recall are approximately balanced at around 0.90, yielding an F1 near 0.91. Raising the threshold beyond 0.65 increases precision at the cost of significant recall degradation, increasing the rate of false Avoid recommendations for genuinely good products.

TABLE VI: Sample Prediction Results (20 Test Instances)

Review Type	pv Score	pr Score
Strong positive	0.94	0.06
Moderate positive	0.78	0.19
Ambiguous / mixed	0.55	0.48
Mild negative	0.38	0.71
Strong negative	0.07	0.93
Positive with caveats	0.67	0.31
Misleading listing	0.41	0.64
Great value product	0.88	0.09

IX. COMPUTATIONAL COMPLEXITY ANALYSIS

Understanding the computational cost of BuySense is important for assessing its scalability to larger review corpora and its suitability for real-time deployment. The system involves two distinct computational phases: an offline training phase and an online inference phase.

A. Training Complexity

During training, the dominant cost is the TextVectorization and Embedding forward pass over the full dataset. For a batch of B samples each of length L tokens, the embedding lookup runs in O(B x L) time. The GlobalAveragePooling1D reduces this to O(B x L x D) where D = 128 is the embedding dimension. The two Dense layers add O(B x D x H) and O(B x H) respectively, where H = 64 is the hidden size. For a dataset of N = 3,600,000 samples with batch size 256 and 2 epochs, total training cost is approximately O(N x L x D) per epoch, giving roughly 14.4 billion multiply-add operations per model. On a standard GPU this corresponds to approximately 18 minutes per model as observed empirically.

The TextVectorization layer is built once before training and applies a fixed hash-based lookup, so its cost scales as O(N x L) with no learned parameters in that layer. The Embedding layer contains MAX_TOKENS x EMBED_DIM = 20,000 x 128 = 2.56 million parameters. The Dense layers add 64 x 128 + 64 = 8,256 parameters and 1 x 64 + 1 = 65 parameters respectively. Total trainable parameters per model: approximately 2.57 million. For two models the total parameter count is approximately 5.14 million — well within the capacity of consumer hardware with 8 GB of RAM.

TABLE VII: Per-Model Parameter Count

Layer	Parameters
Embedding (20000 x 128)	2,560,000
Dense hidden (128 x 64 + 64)	8,256
Dense output (64 x 1 + 1)	65
Total (per model)	2,568,321
Total (both models)	5,136,642
Model file size (each)	~12 MB (.keras)

B. Inference Complexity

At inference time, the input is a single text string of at most SEQ_LEN = 200 tokens. The forward pass through each model requires O(L x D) for embedding lookup, O(D) for pooling, O(D x H) for the hidden layer, and O(H) for the output layer — all constant with respect to dataset size. Total inference cost per model is O(L x D + D x H) ≈ O(200 x 128 + 128 x 64) = O(33,792) floating point operations. This is negligible: both models complete inference in under one millisecond on CPU. The dominant latency in the full pipeline is web scraping, which typically requires two to four seconds

depending on Amazon server response time and whether Playwright fallback is triggered.

X. ABLATION STUDY

To validate the design decisions in the BuySense architecture, we conduct a systematic ablation study varying three key factors: embedding dimension, sequence length, and network depth. Each variant is trained for 2 epochs on a 500,000-sample subset of the training data and evaluated on the full test set. The baseline corresponds to the default BuySense configuration.

TABLE VIII: Ablation Study — Viability Model

Configuration	Val Accuracy	Test F1
Baseline (default)	87.4%	0.913
EMBED_DIM = 64	85.1%	0.888
EMBED_DIM = 256	87.8%	0.916
SEQ_LEN = 100	84.7%	0.881
SEQ_LEN = 400	87.6%	0.914
Extra Dense(128)	87.2%	0.910
No hidden Dense	83.4%	0.862
MAX_TOKENS = 10000	86.0%	0.896
MAX_TOKENS = 50000	87.5%	0.913

The ablation results reveal several insights. Reducing EMBED_DIM from 128 to 64 produces a meaningful accuracy drop of 2.3 percentage points, indicating that richer token representations are important for this task. Increasing to 256 dimensions yields only marginal gains (+0.4%) at twice the memory cost, making 128 the optimal point on the accuracy-efficiency curve. Reducing SEQ_LEN to 100 causes a larger drop of 2.7 points, suggesting that the latter portions of longer reviews contain discriminative information. Extending to 400 tokens provides negligible benefit, confirming that the default 200-token window captures most relevant signal. Adding an extra Dense(128) layer does not improve performance, indicating that the single hidden layer is sufficient to model the non-linear relationships in this dataset. Removing the hidden layer entirely drops F1 by 0.051, validating its inclusion. Vocabulary size has limited impact above 10,000 tokens, with the default 20,000 providing a good balance between coverage and memory.

XI. COMPARISON WITH BASELINE METHODS

To contextualize BuySense performance, we compare against four baseline systems on the same Amazon Review Polarity test set. All baselines are trained on the same data with the same train/test split. Reported metrics are test accuracy and F1 score at a 0.5 threshold.

TABLE IX: Comparison with Baseline Sentiment Classifiers

Method	Test Accuracy	Test F1
Majority class baseline	50.0%	0.333
TF-IDF + Logistic Reg.	82.6%	0.826
TF-IDF + LinearSVC	85.1%	0.851
FastText (supervised)	88.7%	0.887
BuySense Viability Model	91.2%	0.913
BuySense Regret Model	89.7%	0.899

BuySense outperforms all four baselines on both metrics. The TF-IDF with logistic regression baseline achieves 82.6% accuracy, confirming that the task is learnable with shallow features but that embedding-based representations provide a meaningful advantage. LinearSVC with TF-IDF improves to 85.1%, still 6.1 points below the viability model. FastText, which also uses learned word embeddings, reaches 88.7% — close to BuySense but still 2.5 points lower on accuracy. The consistent superiority of BuySense over FastText despite similar architectural simplicity is likely attributable to the sequential training on the full 3.6 million sample corpus, which allows the BuySense embedding layer to learn domain-specific token co-occurrence patterns that generic FastText initialization does not capture.

It is important to note that the baseline comparison evaluates single-output polarity classifiers, while BuySense produces two independent probability outputs. The practical value of BuySense lies not only in its classification accuracy

but in its ability to express nuanced dual-signal recommendations — a capability none of the baselines possess.

XII. DEPLOYMENT AND IMPLEMENTATION

Software Stack

BuySense is implemented entirely in Python 3.10. The machine learning components use TensorFlow 2.x with the Keras functional API. The web scraping layer uses the requests library for standard HTTP retrieval and BeautifulSoup4 for HTML parsing, with Playwright 1.x as the JavaScript-rendering fallback. The web interface is built with Streamlit, which provides a zero-configuration deployment path requiring only a Python environment. Scikit-learn is used for preprocessing utilities and evaluation metric computation. All dependencies are pinned in requirements.txt to ensure reproducibility.

TABLE X: Software Dependencies

Package	Role
tensorflow >= 2.12	Model training and inference
streamlit >= 1.24	Web application framework
beautifulsoup4 >= 4.12	HTML parsing
requests >= 2.31	HTTP retrieval
playwright >= 1.35	JavaScript page rendering fallback
numpy >= 1.24	Numerical operations
pandas >= 2.0	Dataset loading and preprocessing
scikit-learn >= 1.3	Evaluation metrics
matplotlib >= 3.7	Visualization utilities

Installation and Execution

To install BuySense, clone the repository and run pip install -r requirements.txt from the project root. If Playwright-based scraping is required, run playwright install chromium after the pip install step to download the browser binary. To train the models from scratch, execute python train_model.py --epochs 2. Pre-trained model files are included in the repository, making retraining optional. To launch the web application, run streamlit run app.py. The application will be accessible at http://localhost:8501 by default. Users paste an Amazon product URL into the input field and receive the Buy / Wait / Avoid recommendation within seconds.

Scraping Robustness

Amazon employs several anti-scraping mechanisms including User-Agent filtering, cookie requirements, CAPTCHA challenges, and dynamic content rendering via JavaScript. BuySense addresses these through a layered strategy. The primary scraper uses a browser-mimicking User-Agent header and persists session cookies across requests. If the response HTML does not contain the expected product elements (detected by checking for the #productTitle DOM ID), the scraper falls back to Playwright, which launches a headless Chromium browser, waits for the page to render completely, and extracts the fully rendered DOM. This two-stage approach handles the majority of Amazon product page configurations encountered in testing, including sponsored listings, variant-selection pages, and region-specific storefronts.

XIII. DISCUSSION AND FUTURE WORK

BuySense demonstrates that a lightweight embedding-based dual-model architecture can deliver meaningful purchase decision support with minimal training cost. Two epochs on 3.6 million samples, using a four-layer model with roughly 2.7 million parameters each, achieves over 91% test accuracy and 0.96 AUC-ROC — performance levels that would be competitive with much heavier models on this benchmark.

Limitations

Several limitations affect the current system. The training corpus dates from 2015 and may not reflect contemporary product language, evolving review norms, or modern product categories such as software subscriptions and digital goods. Amazon actively deploys anti-scraping measures; while Playwright mitigates JavaScript rendering challenges, IP-level blocking can intermittently prevent retrieval. The decision thresholds BUY_THRESHOLD and WAIT_THRESHOLD are fixed constants derived from empirical testing rather than learned from a calibration dataset, limiting adaptability across product categories.

Future Directions

- Fine-tuning on post-2020 review data stratified by product category to improve domain coverage.
- Replacing the embedding architecture with DistilBERT [8] to capture richer contextual semantics.

- Learning adaptive thresholds using Platt scaling or isotonic regression on a held-out calibration set.
- Adding a price-to-viability feature that integrates listing price into the recommendation score.
- Packaging BuySense as a browser extension for inline Amazon page annotation without URL submission.
- Extending the dual-model design to a five-class output: Strongly Buy, Buy, Neutral, Avoid, Strongly Avoid.

XIV. CONCLUSION

This paper presented BuySense, a dual-model NLP system for Amazon purchase decision support. By independently modeling product viability and regret using two binary text classifiers trained on the Amazon Review Polarity Dataset, BuySense produces a three-class recommendation — Buy, Wait, or Avoid — that is semantically richer than any single-polarity sentiment score. Both models achieve over 87% validation accuracy, 0.95 AUC-ROC, and F1 scores above 0.89 after only two training epochs on 3.6 million review samples. The end-to-end inference pipeline from Amazon URL to recommendation completes in under five seconds. BuySense demonstrates that purpose-built dual-model architectures offer a practical and computationally efficient alternative to general-purpose sentiment tools for domain-specific purchase decision tasks, and establishes a reproducible baseline for future work on regret modeling in e-commerce.

REFERENCES

[1] B. Pang and L. Lee, "Opinion mining and sentiment analysis," Foundations and Trends in Information Retrieval, vol. 2, no. 1-2, pp. 1-135, 2008.
[2] B. Liu, "Sentiment analysis and opinion mining," Synthesis Lectures on Human Language Technologies, vol. 5, no. 1, pp. 1-167, 2012.
[3] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," Advances in NeurIPS, vol. 28, 2015. [Online]. Available: https://arxiv.org/abs/1509.01626
[4] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," Proc. NAACL-HLT, 2015, pp. 103-112.
[5] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," Proc. ICML, 2008, pp. 160-167.
[6] M. Tsiros and C. Mittal, "Regret: A model of its antecedents and consequences in consumer decision making," Journal of Consumer Research, vol. 26, no. 4, pp. 401-417, 2000.
[7] T. Mikolov et al., "Distributed representations of words and phrases and their compositionality," Advances in NeurIPS, 2013, pp. 3111-3119.
[8] J. Devlin et al., "BERT: Pre-training of deep bidirectional transformers for language understanding," Proc. NAACL-HLT, 2019, pp. 4171-4186.
[9] Streamlit, Inc., "Streamlit - The fastest way to build data apps." [Online]. Available: https://streamlit.io/