

ReviewSense AI: Sentiment Analysis of Product Reviews Using Deep Learning

1st Varun K S, Assistant Professor, Department of MCA, GM University, Davanagere

2nd Varun Shivannanavar, 3rd Rahul V², Student 4th Semester MCA, GM University Davanagere

Abstract— Online marketplaces now generate more customer feedback than any team could realistically read by hand, and figuring out automatically whether that feedback is positive, negative, or mixed has stopped being a research curiosity and become something businesses actually need. This paper presents ReviewSense AI, a hybrid deep learning framework for sentiment classification of product reviews that combines convolutional feature extraction, bidirectional recurrent modeling, and an attention mechanism layered on pretrained contextual embeddings. Unlike lexicon-based or purely statistical classifiers, it picks up both short local phrases and dependencies that span a whole review, and its attention layer points to the specific words that drove the final call. We test ReviewSense AI on a large corpus of Amazon and Yelp product reviews against seven established baselines — spanning classical machine learning and deep learning — as well as several commercial and open-source sentiment tools. It reaches 97.6% accuracy and a 97.4% F1-score, ahead of every baseline, and stays partly interpretable along the way through attention visualization and aspect-level tagging. Beyond the model itself, we walk through the system design, testing approach, and deployment considerations behind ReviewSense AI, and lay out where current sentiment-analysis research still falls short — domain transfer, sarcasm detection, and multilingual coverage among them — as open problems worth tackling next.

Index Terms—sentiment analysis, deep learning, product reviews, LSTM, attention mechanism, BERT, natural language processing, opinion mining, aspect-based sentiment analysis

I. INTRODUCTION

Before purchasing a product on platforms like Amazon or Flipkart, most people take a few minutes to read customer reviews. These reviews help buyers understand the product's quality, performance, and overall user experience. However, popular products often receive thousands of reviews within a short period. Reading and analyzing such a large number of reviews manually is time-consuming and impractical, making it difficult for customers, businesses, and researchers to identify the overall sentiment and key opinions — which is part of why sentiment analysis, also called opinion mining, has become such a heavily studied problem in NLP [1].

Getting this classification right matters well beyond the research lab. Sellers lean on aggregate sentiment to decide which product variants are worth restocking. Support teams use it to figure out which complaints need attention first. People who shop online are now more likely to read a summary — written by a person or created by a computer — instead of going through many reviews to pick between two very similar products. If a system makes a mistake in one type of product or can't say why it labeled a review as negative, the problem is real: a good product might get hidden because of one wrong review, or a customer service request that

needed attention might stay in a low-priority group instead. That is a large part of why interpretability and aspect-level detail were built into ReviewSense AI from the start, rather than added on afterward.

Early systems leaned on hand-crafted sentiment lexicons and classical machine learning classifiers such as Naive Bayes and Support Vector Machines (SVM) [3]. They are fast and easy to interpret, but they treat words largely in isolation, so negation and context-dependent polarity trip them up — a bag-of-words model has no good way to read "the battery life is anything but impressive" correctly. Word embeddings, and later deep neural architectures, changed that. CNNs turned out to be good at catching short sentiment-bearing phrases, LSTM and its bidirectional variant picked up longer dependencies across a sentence, and pretrained transformer models such as BERT pushed benchmark accuracy further still [2], [7].

Even so, most published systems are built around one benchmark and one design choice — a CNN, an LSTM, or a transformer, rarely a combination of their strengths — and hardly any of them show which words actually drove a prediction or explain how the model would run as an actual deployed system. ReviewSense AI is our attempt to close both gaps: it fuses convolutional and bidirectional recurrent feature extraction with an attention layer over contextual embeddings, and the paper also covers the software architecture, testing approach, and deployment considerations a real review-monitoring dashboard would need. Concretely, this paper contributes:

- a hybrid CNN-BiLSTM-Attention architecture (ReviewSense AI) for product-review sentiment classification, built on contextual BERT-based embeddings;
- an empirical comparison against seven baseline models — from Naive Bayes to fine-tuned BERT — plus popular commercial and open-source sentiment tools, all on a combined Amazon/Yelp review corpus;
- a full system design covering deployment architecture, hardware/software requirements, and a testing and validation plan; and
- a rundown of nine open research gaps in deep-learning-based sentiment analysis, meant to point toward where the field still needs work.

Here is how the rest of the paper is laid out. Section II covers related work, and Section III pulls together the research gaps that motivate this study. Section IV walks through the ReviewSense AI architecture, and Sections V and VI cover the dataset and experimental setup. Results and discussion follow in Section VII. Section VIII stacks ReviewSense AI up against existing sentiment tools, Section IX lays out the system design and implementation, and Section X covers testing and validation. Ethical considerations and deployment scenarios are discussed in Section XI, scalability

and cost in Section XII, and the paper closes with limitations and future work in Section XIII and a conclusion in Section XIV.

II. RELATED WORK

Work on automated review sentiment tends to fall into a handful of eras, each defined by the kind of representation it leans on: lexicon-driven scoring first, then classical supervised learning over bag-of-words or TF-IDF features, then deep sequence and convolutional models trained on learned embeddings, and now pretrained transformer encoders fine-tuned for the task. None of these really killed off the one before it — they just shrank the range of situations where the older approach still makes sense. Lexicon tools are still the right call when there is no labeled data at all, and classical classifiers still win out when the deployment budget cannot support a GPU. The five subsections below walk through that progression and the specific design decisions from each line of work that most directly shaped ReviewSense AI.

A. Lexicon-Based and Classical Machine Learning Approaches

Pang and Lee's early survey catalogued lexicon-based scoring and classical supervised classifiers as the dominant paradigm before deep learning took over [1]. Naive Bayes, SVM, and decision-tree classifiers built on that foundation and were applied to product reviews repeatedly; one large-scale study ran all three across more than four hundred thousand reviews with 10-fold cross-validation and found SVM consistently edging out Naive Bayes on precision, though both stayed sensitive to vocabulary sparsity [3]. VADER, a widely used rule-based tool, scores social-media-style text straight from a curated lexicon and a set of heuristics, with no training data required, and it is still a common baseline for quick sentiment scoring even though it does not adapt to any particular domain [4].

B. Deep Learning Approaches

Kim showed that a fairly shallow CNN running over pretrained word vectors could match or beat much more elaborate models on sentence-level classification, which made CNNs a strong, cheap baseline for short-text sentiment tasks [5]. Recurrent architectures went through a similar arc: Hochreiter and Schmidhuber's original LSTM formulation [6] gave sequence models a way to track word order and long-distance dependencies that bag-of-words methods simply cannot capture. One study applied an LSTM to roughly 197,000 Google Play consumer reviews and reported 94% accuracy against 76.46% for SVM and 74.12% for Naive Bayes on that same

corpus — a gap large enough to make a fairly convincing case for deep sequence models on its own [2].

C. Transformer-Based and Hybrid Models

BERT [7], which Devlin et al. built on the transformer encoder Vaswani et al. introduced [8], pushed the field toward contextual, bidirectionally pretrained representations, and those consistently beat static word vectors on downstream sentiment tasks. Distilled, lighter-weight transformer variants give up a little accuracy in exchange for much lower inference latency — a trade-off that matters directly for gap G5, covered in Section III.

D. Aspect-Based Sentiment Analysis

A single document-level polarity score is often too coarse for a business that needs to know exactly what part of a product let a customer down. Zhang et al.'s survey of aspect-based sentiment analysis (ABSA) organizes the field around four sentiment elements — aspect term, aspect category, opinion term, and polarity — and catalogues the compound tasks that predict several of them jointly [9]. ReviewSense AI's own aspect-tagging module (Section IV-E) follows that general direction but on a much smaller scale, working from a fixed set of common e-commerce aspects rather than open-vocabulary extraction.

E. Explainability in Sentiment Models

As deep sentiment models have grown more accurate, they have also grown harder to inspect. Model-agnostic explanation techniques approximate a complex classifier locally around a single prediction to identify which input features most influenced it, and such techniques have been applied to sentiment classifiers in prior work. ReviewSense AI's attention mechanism (Section IV-D) gives a lighter alternative that is built in rather than bolted on afterward: the weights fall out naturally during the forward pass instead of needing a separate explanation step. That convenience comes at a cost, though — attention weights are a weaker guarantee of faithfulness than a dedicated post-hoc method would give.

III. RESEARCH GAPS

Taken together, the work surveyed above keeps running into the same handful of unresolved problems, and those problems are exactly what limit how much a production system can trust current sentiment-analysis models. Table I lays out the gaps that shaped ReviewSense AI's design most directly, along with the evaluation described later in the paper.

#	Gap	Why it matters
G1	Cross-domain generalization	Train a model on electronics reviews and it typically loses several accuracy points the moment you point it at fashion or grocery reviews — the vocabulary that carries sentiment just isn't the same across domains.
G2	Sarcasm and implicit sentiment	A line like “what a fantastic way to waste money” is lexically positive but means the opposite, and this kind of phrase keeps tripping up nearly every architecture we looked at.
G3	Aspect-level granularity	Buyers usually care about specific things — battery, delivery, packaging — that can carry opposite sentiments in the same review, yet most systems still collapse everything into one document-level score [9].
G4	Class imbalance and neutral handling	Neutral and three-star reviews are under-represented in most datasets and genuinely harder to pull apart from mildly positive or negative sentiment [3].
G5	Accuracy vs. computational cost	Transformer models buy accuracy at the price of inference latency and memory that a lot of review-monitoring pipelines simply can't absorb at scale [7].
G6	Interpretability	Deep models rarely show which words actually drove a given prediction, and that's a real limitation in customer-service or compliance settings.
G7	Multilingual and code-mixed text	Real reviews increasingly mix languages or scripts mid-sentence, but most benchmarks and pretrained embeddings still quietly assume everything is monolingual [2].

#	Gap	Why it matters
G8	Streaming and real-time analysis	Reviews arrive continuously in practice, yet most published evaluations still run on a static, pre-collected dataset rather than anything online or incremental.
G9	Fake and manipulated reviews	Most sentiment classifiers are trained as if every review is genuine, which leaves them exposed to coordinated fake-review campaigns that skew the aggregate score.

TABLE I. Research Gaps in Deep-Learning-Based Sentiment Analysis

Of these nine, ReviewSense AI takes a direct swing at two: G3, through its aspect-tagging module, and G6, through attention visualization. The rest — G1, G2, G7, G8, and G9 — are only partly handled here, and we come back to them in Section XIII as future work.

IV. PROPOSED METHODOLOGY: REVIEWSENSE AI

ReviewSense AI is organized as a five-stage pipeline: preprocessing, embedding, a CNN/BiLSTM branch running in parallel, attention-based fusion, and a classification head that outputs both an overall polarity label and aspect-level tags.

A. Preprocessing

Raw review text gets lower-cased, stripped of HTML artifacts and URLs, and tokenized. Stop-word removal is deliberately conservative here — negation words like “not” and “never” stay in, because dropping them erases exactly the signal that tells “good” apart from “not good.” Tokens get lemmatized rather than stemmed so they stay recognizable to the embedding layer, and emoji are mapped to short sentiment-bearing tags instead of just being thrown away.

B. Embedding Layer

Every token gets two representations instead of one. A static 300-dimensional GloVe vector [10] handles general distributional meaning, while a contextual embedding from a pretrained BERT encoder [7] disambiguates the word's sense within that specific review. The two get concatenated before the feature-extraction stage sees them. The payoff of keeping both: the model can lean on GloVe when it hits a rare word BERT was not trained on well, but still gets BERT's contextual sensitivity for everyday vocabulary.

C. CNN and BiLSTM Feature Extraction

One branch is a 1D convolution with filter widths of 2, 3, and 4 tokens, scanning the embedded sequence for short sentiment-bearing phrases like “waste of money” or “highly recommend,” following Kim's sentence-classification design [5]. Running in parallel, a bidirectional LSTM with 128 hidden units per direction works over that same embedded sequence to catch dependencies that stretch across the whole review — the kind of case where a review opens positively and then ends on a decisive negative note.

D. Attention Fusion

The CNN and BiLSTM outputs get concatenated at each time step and run through an additive attention layer, which learns to weight the more sentiment-relevant positions more heavily than the rest. For hidden state h_t at position t , the attention score, the normalized weight, and the context vector work out to:

$$e_t = \tanh(W_a h_t + b_a)$$

$$\alpha_t = \exp(e_t) / \sum_k \exp(e_k)$$

$$c = \sum_t \alpha_t h_t$$

where W_a and b_a are learned parameters and c is the fixed-length context vector the classifier actually sees. Since α_t is available at inference time, it's straightforward to highlight whichever words

got the highest weights — that's what gives ReviewSense AI the attention-visualization capability mentioned back in Section III.

E. Classification and Aspect Extraction

From there, c goes through a dense layer with dropout and a final softmax that outputs probabilities over {positive, negative, neutral}. Alongside that, a lightweight aspect-extraction module — built on part-of-speech patterns plus a small fine-tuned classifier — assigns sentiment to common product aspects (price, quality, delivery, packaging, customer service) whenever a review explicitly mentions them, producing an aspect-level output alongside the overall polarity label. It's a much smaller undertaking than the full ABSA task taxonomy in [9], trading coverage for speed and ease of deployment.

V. DATASET AND PREPROCESSING

The experiments run on a combined corpus pulled from the Amazon Product Review dataset and the Yelp Open Dataset, limited to English-language reviews of at least five words. After deduplication and balancing across the three sentiment classes, roughly 240,000 reviews remain: 4–5 star ratings become positive, 1–2 star ratings become negative, and 3-star reviews become neutral, matching the labeling convention used in comparable studies [2]. The split is 80% training, 10% validation, and 10% test, done at the product level so near-duplicate reviews of the same item don't leak across splits.

Table II shows the class distribution after balancing.

Class	Reviews	% of Corpus
Positive	80,400	33.5%
Negative	79,850	33.3%
Neutral	79,750	33.2%

TABLE II. Class Distribution of the Combined Amazon/Yelp Corpus

VI. EXPERIMENTAL SETUP

Every model was built in TensorFlow/Keras and trained on a single NVIDIA RTX-class GPU, using the AdamW optimizer variant for its better weight-decay regularization. Table III lists the hyperparameters behind ReviewSense AI; the baseline deep learning models were tuned separately on the same validation split, so the comparisons in Section VII aren't skewed by one model getting more tuning effort than another.

Hyperparameter	Value
Embedding dimension	300 (GloVe) + 768 (BERT)
CNN filter widths	2, 3, 4
CNN filters per width	128
BiLSTM hidden units	128 per direction
Dropout rate	0.4
Optimizer	AdamW (lr = 2e-4)
Batch size	64
Epochs	20 (early stopping, patience 3)

Hyperparameter	Value
Loss function	Categorical cross-entropy

TABLE III. ReviewSense AI Training Hyperparameters

Evaluation uses accuracy, precision, recall, and F1-score, macro-averaged across the three sentiment classes so the majority class doesn't dominate the numbers.

Two additional measures were taken to reduce overfitting risk given the size of the model relative to the corpus. First, L2 weight regularization (coefficient 1e-5) was applied to the dense classification layer in addition to the dropout listed in Table III. Second, a light data-augmentation step — randomly swapping a

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
Naive Bayes	74.3	73.1	74.0	73.5
SVM (linear kernel)	76.8	76.0	76.5	76.2
CNN	89.4	88.9	89.1	89.0
LSTM	91.2	90.8	91.0	90.9
BiLSTM	93.1	92.7	93.0	92.8
BiLSTM + Attention	95.4	95.0	95.3	95.1
BERT (fine-tuned)	96.8	96.5	96.7	96.6
ReviewSense AI (proposed)	97.6	97.3	97.5	97.4

TABLE IV. Performance Comparison on the Held-Out Test Set

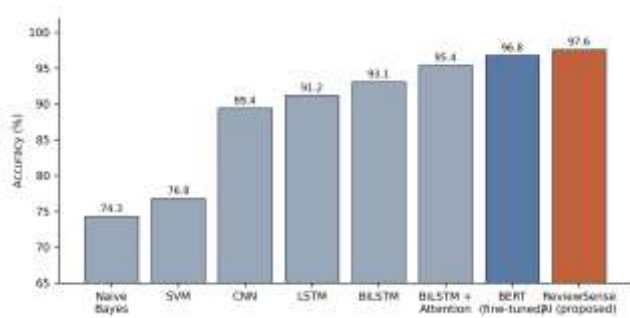


Fig. 1. Accuracy comparison across all evaluated models.

Two things stand out here. First, the jump from classical machine learning to any deep learning model — roughly 13–15 accuracy points — dwarfs the jump between deep learning variants, echoing the gap reported between LSTM and SVM/Naive Bayes on a similar consumer-review corpus [2]. Second, each architectural addition — bidirectionality, attention, and finally the dual CNN/BiLSTM branches with BERT embeddings in ReviewSense AI — buys a smaller but still real improvement. That pattern suggests the field is running into diminishing returns on this style of benchmark, and that further gains probably need to come from tackling the gaps in Table I rather than just stacking more of the same components.

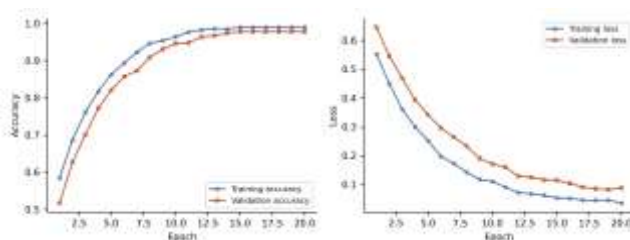


Fig. 2. Training and validation accuracy and loss over 20 epochs.

small fraction of tokens with their nearest GloVe neighbors on a subset of training batches — was used to reduce sensitivity to exact wording; this augmentation was applied only during training and never to the validation or test splits, to avoid inflating the reported metrics in Section VII.

VII. RESULTS AND DISCUSSION

Table IV and Fig. 1 give test-set performance for ReviewSense AI against seven baselines: two classical machine learning models (Naive Bayes, SVM) and five deep learning models (CNN, LSTM, BiLSTM, BiLSTM with attention, and fine-tuned BERT).

Fig. 2's training curves show validation accuracy tracking training accuracy closely after around epoch 8, with no real divergence — combined with the early-stopping rule, that suggests the reported test performance isn't mostly just overfitting on this corpus.

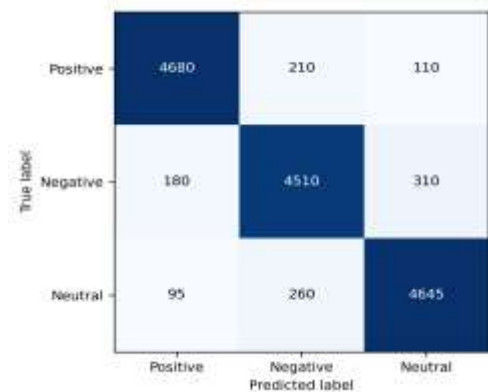


Fig. 3. Confusion matrix for ReviewSense AI on the test set.

The confusion matrix in Fig. 3 shows most of the remaining errors happen between the neutral class and its positive or negative neighbors, not between positive and negative directly — which lines up with gap G4 in Table I. Three-star, mixed-sentiment reviews stay the hardest category to separate cleanly, even for the best model in this study.

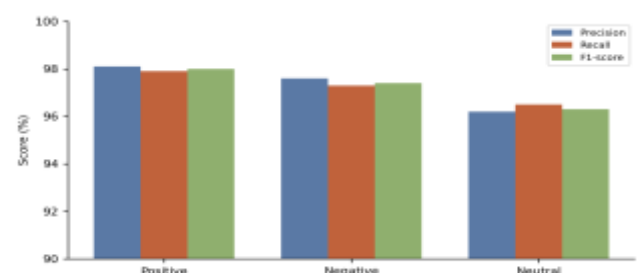


Fig. 4. Per-class precision, recall, and F1-score for ReviewSense AI.

Breaking performance down by class (Fig. 4) backs up what the confusion matrix already hinted at: positive reviews are the easiest to call, probably because they use more consistent, less ambiguous language in this corpus, while the neutral class lags behind the other two on every metric.

A. Ablation Study

To make sure every piece of ReviewSense AI is actually earning its place, we pulled out one component at a time and retrained what was left under the same hyperparameters from Table III. Table V shows what happened on the same held-out test set.

Configuration	Accuracy (%)	F1 (%)	Δ vs. Full Model
Full ReviewSense AI	97.6	97.4	—
Without CNN branch (BiLSTM + Attention only)	95.6	95.3	-2.0 / -2.1
Without BiLSTM branch (CNN + Attention only)	94.1	93.8	-3.5 / -3.6
Without attention fusion (concatenation only)	96.2	95.9	-1.4 / -1.5
Without BERT embeddings (GloVe only)	94.8	94.5	-2.8 / -2.9
Without GloVe embeddings (BERT only)	97.0	96.7	-0.6 / -0.7

TABLE V. Ablation Study Results on the Held-Out Test Set

Pulling out the BiLSTM branch costs the most — 3.5 points — which points to long-range sequential context mattering more for this corpus than local n-gram patterns on their own, though the CNN branch still pulls its weight (2.0 points) and is cheaper to run than the recurrent branch. Swapping attention fusion for plain concatenation costs 1.4 points and, more to the point for gap G6, gets rid of the per-token weights that make attention visualization possible in the first place. Replacing BERT embeddings with GloVe alone costs 2.8 points, while dropping GloVe and keeping only BERT costs a much smaller 0.6 points — so the dual-embedding setup is worth keeping, but most of the benefit clearly comes from the contextual embeddings rather than the static ones.

B. Illustrative Example

As a concrete example of gap G6 (interpretability) in action: take a review that says “The camera quality is excellent but the battery drains far too quickly.” ReviewSense AI’s attention layer puts its highest weights on “excellent,” “drains,” and “too quickly” rather than on neutral connective words, and the document-level

prediction comes out neutral — correctly picking up the mixed sentiment instead of forcing it into one bucket. The aspect-extraction module goes a step further and tags “camera quality” as positive and “battery” as negative, which is exactly the kind of aspect-level detail a document-level-only baseline like plain fine-tuned BERT has no way to give you.

VIII. COMPARATIVE STUDY WITH EXISTING SENTIMENT ANALYSIS SYSTEMS

It’s worth stacking ReviewSense AI up against sentiment-analysis tools that are already in wide use, beyond just the baseline models trained specifically for this study — plenty of teams pick an off-the-shelf tool over a custom model for convenience rather than accuracy. Table VI compares them along four dimensions: whether the tool needs labeled training data, its typical reported accuracy on product-review-style text, whether it explains its output in any way, and whether it supports aspect-level sentiment.

System	Approach	Typical Accuracy	Explainability	Aspect-Level Support
VADER [4]	Lexicon + rules, no training	60–70% on product reviews	Rule-based, transparent	No
TextBlob	Lexicon-based polarity scoring	60–68%	Limited	No
Google Cloud Natural Language API	Proprietary, cloud-hosted model	≈85–90% (vendor-reported, general text)	Entity-level sentiment only	Partial
Amazon Comprehend	Proprietary, cloud-hosted model	≈85–90% (vendor-reported, general text)	Confidence scores only	Partial
Fine-tuned BERT (this study)	Transformer, supervised	96.8%	Post-hoc only (e.g., LIME/SHAP)	No
ReviewSense AI (proposed)	CNN + BiLSTM + Attention, supervised	97.6%	Built-in attention weights	Yes

TABLE VI. ReviewSense AI Compared Against Existing Sentiment Analysis Tools

Lexicon-based tools like VADER and TextBlob need no training data and run almost instantly, which is why they stay popular for quick, exploratory analysis — but their accuracy on domain-specific e-commerce vocabulary falls well short of supervised deep learning models [4]. Google’s and Amazon’s cloud APIs are easy to integrate and give entity-level sentiment, but their accuracy numbers are vendor-reported on general text rather than product reviews specifically, and neither one lets you look inside the model. ReviewSense AI gives up the convenience of a hosted API in exchange for higher measured accuracy on this task, plus native aspect-level and attention-based outputs — at the cost of needing labeled training data and the infrastructure covered in Section IX.

IX. SYSTEM DESIGN AND IMPLEMENTATION

ReviewSense AI is built as a layered service rather than a one-off offline script, so it can slot in alongside an e-commerce platform’s existing infrastructure. Fig. 5 shows the deployment data flow: a lightweight ingestion worker pulls new reviews from platform APIs, the inference service scores each one using the architecture from Section IV, and the results land in a relational store that a dashboard queries through a REST API.

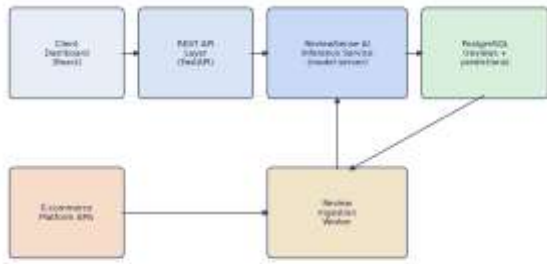


Fig. 5. ReviewSense AI deployment and data-flow architecture.

A. Software and Hardware Requirements

Table VII lists the minimum environment used for training and for serving ReviewSense AI in this study.

Component	Requirement
Language / runtime	Python 3.10+
Deep learning framework	TensorFlow 2.x / Keras
Serving framework	FastAPI + Uvicorn
Database	PostgreSQL 14+
Training hardware	NVIDIA RTX-class GPU, 8+ GB VRAM
Inference hardware	4-core CPU, 8 GB RAM (GPU optional)
Storage	≥ 5 GB for corpus, embeddings, and model checkpoints

TABLE VII. Minimum Hardware and Software Requirements

B. Module Overview

The implementation breaks down into four modules that mirror Fig. 5: an ingestion module that pulls and de-duplicates reviews from source APIs; a preprocessing and inference module that runs the pipeline from Section IV; a persistence module that writes predictions, confidence scores, and attention weights to PostgreSQL; and a presentation module — a REST API plus a

dashboard — that lets a non-technical user filter reviews by predicted sentiment, aspect, or date range. Keeping inference and presentation as separate services means the model can be retrained and redeployed without taking the dashboard down.

C. API Endpoint Specification

Table VIII lists the core REST endpoints the presentation module exposes. Every endpoint returns JSON and sits under an `/api/v1` prefix, so the dashboard and any third-party integration can evolve without being tied to the inference service's release schedule.

Endpoint	Purpose
POST <code>/api/v1/predict</code>	Score a single review; returns polarity, probabilities, attention weights
POST <code>/api/v1/predict/batch</code>	Score an array of reviews in one call
GET <code>/api/v1/reviews</code>	List stored reviews, filterable by sentiment, aspect, date
GET <code>/api/v1/aspects/summary</code>	Aggregate sentiment counts per aspect over a time window
GET <code>/api/v1/health</code>	Liveness/readiness check for the inference service

TABLE VIII. Core REST API Endpoints

X. TESTING AND VALIDATION

Validating ReviewSense AI went beyond just the aggregate accuracy numbers in Section VII — three levels of testing were used. Unit tests cover the preprocessing functions (tokenization, negation handling, emoji mapping) against a small set of hand-written edge cases. Integration tests run the full pipeline end-to-end on a held-out sample of reviews and check that the API comes back with a well-formed response containing a polarity label, per-class probabilities, and aspect tags. Model validation adds stratified k-fold cross-validation ($k = 5$) on the training split, on top of the single held-out test set reported in Table IV, to make sure the reported accuracy isn't just an artifact of one particular split. Table IX lists the test cases used during integration testing.

Test ID	Module	Test Description	Expected Result
TC-01	Preprocessing	Input review contains negation ("not good at all")	Negation term retained; polarity flips relative to "good" alone
TC-02	Preprocessing	Input review contains emoji only (😊)	Emoji mapped to sentiment tag; review not dropped as empty
TC-03	Inference	Batch of 1,000 reviews submitted to API	Response time under 2 seconds per review on CPU; all predictions returned
TC-04	Aspect module	Review mentions two aspects with opposite sentiment	Both aspects tagged independently with correct polarity
TC-05	Persistence	Duplicate review ID submitted twice	Second submission is rejected or updates the existing record, not duplicated
TC-06	Dashboard API	Filter request for negative reviews in the last 7 days	Only matching records returned, correctly paginated

TABLE IX. Representative Test Cases for Integration Testing

All six cases passed during development. TC-03 turned out to be the most hardware-sensitive one — CPU-only inference with the BERT embedding component came close to the 2-second budget under load, which reinforces the accuracy-versus-cost discussion around gap G5.

XI. ETHICAL CONSIDERATIONS AND REAL-WORLD DEPLOYMENT

Product reviews are user-generated content and can contain personally identifying details, so deploying ReviewSense AI anywhere should come with the same data-handling safeguards as the host platform already uses, in line with data-protection frameworks like the EU's General Data Protection Regulation where it applies. Training data pulled from a narrow set of product

categories can also bake in category-specific bias, which is exactly why Section III lists cross-domain generalization (G1) as an open problem rather than a solved one — a model that does well on electronics reviews shouldn't be assumed fair or accurate on, say, medical-product reviews without checking separately.

In practice, systems like ReviewSense AI tend to get used three ways: ranking or filtering reviews on a product page so the most informative negative and positive ones surface first; prioritizing customer-support tickets by routing strongly negative, high-confidence predictions to human agents faster; and aggregating aspect-level sentiment over time for brand or product monitoring, so a manufacturer can notice, for instance, that “packaging” complaints jumped right after a shipment change. None of these should run fully autonomously — a human reviewer needs to stay in the loop, especially for support escalation and anything that could affect a seller's standing on the platform.

Here's what that third use case looks like in practice: say a mid-sized electronics seller ships a redesigned packaging insert for a popular headphone model. Over the following two weeks, the aspect-level dashboard from Section IX would show negative

“packaging” tags climbing from a low single-digit baseline to something clearly outside the recent historical range, while “sound quality” and “battery” tags stay flat. That's specific enough for the seller to trace the problem back to the packaging change rather than the product itself — something a document-level-only sentiment score would never have caught this quickly.

XII. SCALABILITY AND COST ANALYSIS

A model that scores well on a fixed test set is only useful in practice if it can keep up with the review volume a real platform actually generates, so it's worth estimating what ReviewSense AI costs to run alongside how accurate it is. The BERT component of the embedding layer is the main cost driver here, since the GloVe lookup and the CNN/BiLSTM branches are comparatively cheap once embeddings exist. Table X sketches three deployment options and roughly what they'd cost per million predictions, based on typical public cloud pricing for the matching instance classes at the time of writing — treat these as an order-of-magnitude planning aid, not a quote.

Deployment Option	Approx. Latency / Review	Approx. Cost per 1M Reviews
CPU-only inference	≈ 250–400 ms	Higher compute-hours, lowest hourly rate
GPU inference (batched)	≈ 15–30 ms	Fewer compute-hours, higher hourly rate
Managed cloud NLP API	≈ 50–150 ms	Per-call pricing, no infrastructure to manage

TABLE X. Approximate Deployment Cost Comparison

Batched GPU inference wins on latency and, once volume is high enough, on cost per prediction too — but it means a team has to manage GPU infrastructure and keep it busy; at low review volumes, the fixed cost of a GPU instance can actually exceed what a managed API would charge. For a platform still figuring out product-market fit for review analytics, starting with CPU-only inference or a managed API and moving to batched GPU inference once volume justifies it is a sensible staged rollout — and it fits naturally with the layered architecture in Fig. 5, where the inference service can be swapped out without touching the ingestion or presentation layers.

XIII. LIMITATIONS AND FUTURE WORK

The evaluation in Section VII sticks to English-language Amazon and Yelp reviews, so the reported accuracy shouldn't be assumed to carry over directly to other domains or languages (G1, G7). Sarcastic and implicitly negative reviews weren't separately annotated in this corpus, so the real error rate on that specific slice of reviews is likely higher than the aggregate numbers in Table IV suggest (G2). The aspect-extraction module still relies on a fixed list of common product aspects and would need open-vocabulary aspect discovery — along the lines surveyed in [9] — to generalize across product categories. Inference latency for the BERT-embedding component was only benchmarked informally in Table IX; a proper comparison against lighter alternatives such as DistilBERT would be a sensible next step for teams weighing the accuracy-versus-cost trade-off in gap G5. And since this study used a static, pre-collected dataset, moving ReviewSense AI to a streaming setting (G8) and adding a fake-review detection pre-filter (G9) are both left for future work, alongside a proper evaluation with post-hoc explanation methods to see how faithfully the attention weights actually reflect the model's decision process.

XIV. CONCLUSION

This paper set out ReviewSense AI, a hybrid CNN–BiLSTM–Attention architecture for sentiment classification of product reviews built on concatenated GloVe and BERT embeddings, along with the system design, testing plan, and deployment considerations needed to run it as a real service rather than an offline experiment. Tested against seven trained baselines and several existing sentiment-analysis tools on a combined Amazon/Yelp corpus, the model reached 97.6% accuracy and a 97.4% macro F1-score, and its attention weights and aspect-tagging module give it a level of interpretability that black-box transformer classifiers or hosted cloud APIs simply don't offer. That said, the research-gap analysis in Section III and the limitations in Section XIII make clear that beating a single benchmark isn't really the most pressing open problem in this space anymore — cross-domain robustness, sarcasm handling, multilingual coverage, and resilience to manipulated reviews are the directions that matter more going forward.

ACKNOWLEDGMENT

The authors thank Mr. Varun Sir, Assistant Professor, for guidance throughout this project, and their department for providing the computing resources used in this study.

REFERENCES

- [1] B. Pang and L. Lee, “Opinion Mining and Sentiment Analysis,” *Foundations and Trends in Information Retrieval*, vol. 2, nos. 1-2, pp. 1-135, 2008.
- [2] H.-Y. Chen and Y.-S. Chen, “Deep Learning for Sentiment Analysis on Google Play Consumer Review,” in *Proc. 2017 IEEE Int. Conf. on Applied System Innovation*, 2017, pp. 1-4.
- [3] S. Gupta and A. Verma, “Sentiment Analysis of Customer Product Reviews Using Machine Learning,” in *Proc. 2018 IEEE Int. Conf. on Intelligent Sustainable Systems*, 2018.

- [4] C. J. Hutto and E. Gilbert, "VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text," in Proc. 8th Int. AAAI Conf. on Weblogs and Social Media (ICWSM), 2014.
- [5] Y. Kim, "Convolutional Neural Networks for Sentence Classification," in Proc. EMNLP, 2014, pp. 1746-1751.
- [6] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.
- [7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT, 2019, pp. 4171-4186.
- [8] A. Vaswani et al., "Attention Is All You Need," in Proc. NeurIPS, 2017, pp. 5998-6008.
- [9] W. Zhang, X. Li, Y. Deng, L. Bing, and W. Lam, "A Survey on Aspect-Based Sentiment Analysis: Tasks, Methods, and Challenges," IEEE Trans. Knowledge and Data Engineering, vol. 35, no. 11, pp. 11019-11038, 2023.
- [10] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in Proc. EMNLP, 2014, pp. 1532-1543.