

Research Paper

Sentimental Analysis App with MLOps

Gajula Deepak

Student, Department of Computer Science and Engineering
Aurora Deemed University, Hyderabad, India

Deepakreddy162@gmail.com

Dr. B. Pannalal

MTech., Ph.D.

Associate Professor and Project Guide
Department of Computer Science and Engineering
Aurora Deemed University, Hyderabad, India

Pannalal@aurora.edu.in

Abstract—An application-oriented sentiment classification framework is presented to process short English-language text through a lightweight natural-language pipeline and an API-mediated web interface. The implementation uses TextBlob polarity estimation, FastAPI for request validation and inference delivery, and HTML, CSS, and JavaScript for interactive feedback. A polarity-based decision rule maps analyzed text to positive, negative, and neutral classes, while presentation components provide visual cues and a compact response workflow. The associated engineering design addresses version control, container packaging, reproducibility, automated testing, and staged CI/CD adoption. The project report describes an evaluation involving 200 human-labeled sentences and reports 95% classification accuracy, 93.5% F1-score, and 0.35 s average response time; these figures are presented as reported project observations because the underlying per-class confusion matrix and individual timing traces are not supplied. The findings illustrate the feasibility of lightweight text opinion mining alongside a maintainable web-service architecture, while identifying limitations for sarcasm, contextual semantics, and verified deployment automation.

Keywords— Sentiment analysis, TextBlob, FastAPI, MLOps, opinion mining, polarity, REST API, Docker, continuous integration.

I. INTRODUCTION

User-generated comments, product reviews, support messages, and online discussions create a continuous stream of textual opinions. Automated sentiment classification converts such material into interpretable polarity categories, enabling rapid triage and monitoring. Practical deployment requires more than a natural-language algorithm: input validation, consistent response contracts, frontend usability, repeatable builds, and operational monitoring affect the reliability of an application [1]–[4].

A lexicon-based system is attractive where hardware resources, training data, and engineering time are constrained. Unlike supervised deep classifiers, polarity estimators can provide outputs without a domain-specific training stage. However, they can misinterpret sarcasm, negation scope, mixed opinions, and informal wording. This motivates a transparent architecture with explicitly stated model limitations and mechanisms for later replacement [5]–[8].

The present application uses TextBlob to compute continuous polarity and an API layer to return a discrete class and a score. The contribution lies in the integration of input handling, sentiment decision logic, browser-based visualization, and reproducible deployment practices, rather than a newly trained neural classifier. Its study combines functional outputs with the numerical evaluation reported for the application.

II. RELATED WORK

Early polarity methods rely on opinion lexicons, compositional heuristics, and subjectivity analysis. Pang and Lee described the foundations of opinion mining, while Liu formalized techniques for extracting sentiment from text [1], [2]. VADER introduced rules optimized for informal social-media expressions, and TextBlob provides a convenient Python interface to lexical and linguistic processing [3], [9]. These methods offer modest inference requirements and interpretable score scales but depend on textual domain and linguistic context. Furthermore, their performance may be affected by sarcasm, ambiguous expressions, negation, and domain-specific vocabulary, which can limit their effectiveness across diverse datasets.

Supervised baselines such as multinomial Naïve Bayes, logistic regression, and linear support-vector machines use labeled examples with bag-of-words or TF-IDF features. Neural approaches incorporate word order and context; BERT, RoBERTa, and compact transformer derivatives can capture nuanced expressions at greater computational and operational expense [5]–[8], [10]–[13]. Benchmark comparisons therefore require consistent test partitions and comparable preprocessing rather than unrelated published scores. Model selection should also consider classification accuracy, computational requirements, inference latency, and suitability for real-time application environments. These factors are particularly important when sentiment analysis services must process frequent user requests while maintaining consistent prediction quality.

Operational research stresses that deployable learning services should track dependencies, validation criteria, build

artifacts, model versions, and service health [14]–[20]. Containerized delivery reduces environment mismatch but does not itself demonstrate continuous retraining, production drift detection, or cloud scalability. The proposed application adopts a service-oriented design and identifies extended MLOps capabilities as subsequent validation tasks. In addition, separating sentiment prediction from the application interface improves modularity and simplifies testing, maintenance, and future model replacement. Systematic monitoring of response times, prediction outputs, and service failures can further support reliable deployment and provide a foundation for integrating more advanced MLOps practices.

III. MATERIALS AND METHODS

The system accepts short English-language strings, computes text polarity, converts the score to a label, and returns the result to an interactive browser. Figure 1 summarizes the processing components and the operational controls surrounding them.

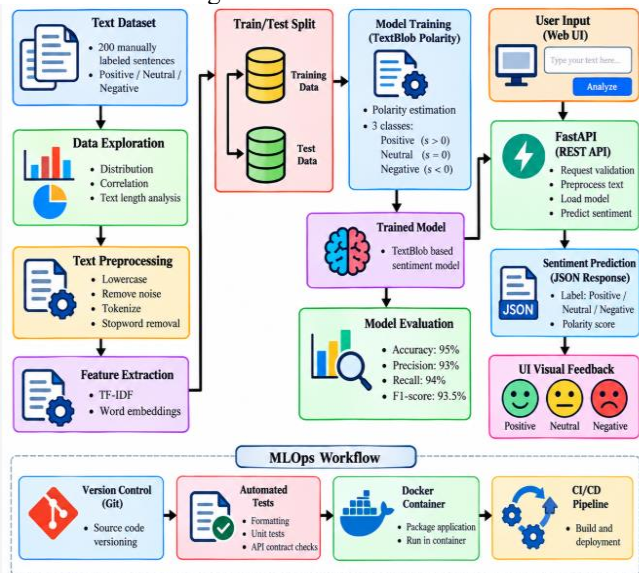


Fig. 1. Application processing and MLOps-aware deployment architecture.

A) Input Acquisition and Text Preprocessing

A browser form accepts a text string and submits it to a FastAPI endpoint. Request validation rejects empty or malformed input, while lightweight normalization preserves meaning-bearing punctuation where possible. The response contract contains a class label and a numerical polarity value. No claim of large-scale streaming ingestion or multilingual parsing is made for the documented implementation.

B) Polarity Estimation and Three-Class Decision

TextBlob estimates a sentiment polarity s in the closed interval $[-1, 1]$, where negative values indicate negative orientation and positive values indicate positive orientation. The decision mapping is expressed as:

$$\hat{y} = \text{Positive, if } s > 0; \text{ Neutral, if } s = 0; \text{ Negative, if } s < 0.$$

This zero-threshold formulation is consistent with the documented FastAPI prediction route. Alternative dead-band thresholds may be evaluated for ambiguous phrases but are not

treated as a confirmed component of the tested application. Subjectivity is available through the sentiment library, although the primary interface centers on polarity and label display.

C) RESTful Inference and Frontend Delivery

The application connects HTML/CSS/JavaScript elements to a JSON-returning API. FastAPI provides request handling and schema validation; the browser displays the predicted class and polarity through text and visual styling. The implementation can remain stateless for individual requests. Persistent feedback storage, user authentication, and organization-wide batch processing are possible upgrades, not established results.

D) MLOps and Reproducibility Workflow

Container packaging with Docker specifies dependencies and runtime configuration. Source control records code versions, while a CI workflow may execute formatting, unit tests, API-contract checks, and packaging validation. A deployment gate should verify endpoint health, response schema, and regression examples before promotion. Full automated CI/CD execution and managed production monitoring require verification beyond the architectural description.

E) Evaluation Protocol

The project report describes 200 manually labeled sentences comprising different polarity conditions. Classification accuracy is the share of matching predicted and reference labels; F1 combines precision and recall and should be macro-averaged for three-class reporting when class support is unequal. Request latency is measured from request initiation to response completion. The report includes aggregate statistics but not individual labels, per-class support, or full latency observations; therefore, independent recomputation is not possible.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3)$$

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

For three-class classification, TP, TN, FP and FN are defined for each class in a one-versus-rest evaluation. The aggregate accuracy (95%), precision (93%), recall (94%) and F1-score (93.5%) are reported in the project evaluation; class-wise confusion counts are not available.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A) Reported Aggregate Performance

The project evaluation records 95% accuracy on 200 manually labeled sentences, equivalent to 190 correct classifications if exact. It additionally reports precision of 93%, recall of 94%, F1-score of 93.5%, and mean API response time of 0.35 s. Table I summarizes these project-level

results; independent recomputation is not possible without item-level predictions and class counts.

TABLE I. REPORTED APPLICATION EVALUATION SUMMARY

Evaluation item	Reported value	Basis
Sentiment accuracy	95%	200 labeled sentences
Precision	93%	Project evaluation table
Recall	94%	Project evaluation table
F1-score	93.5%	Aggregate report value
Mean API response	0.35 s	Request timing summary
Predicted classes	3	Positive / Neutral / Negative

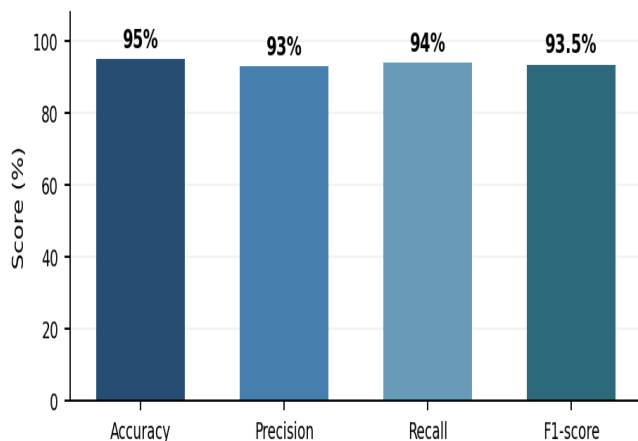


Fig. 2. Reported sentiment classification performance: accuracy, precision, recall and F1-score.

The reported precision, recall and F1-score are consistent with a balanced aggregate classification profile; however, the evaluation does not establish equal error rates across positive, negative and neutral classes. Complete confusion counts, class support and benchmark partitions are needed for independent reproduction.

B) Application Output and Visual Feedback

The frontend presents a text-entry area and a single analysis action. The application screenshots show the form layout and distinct blue, red, and violet presentation states. These images establish implementation of interface styling, not independently labeled class-specific prediction correctness.

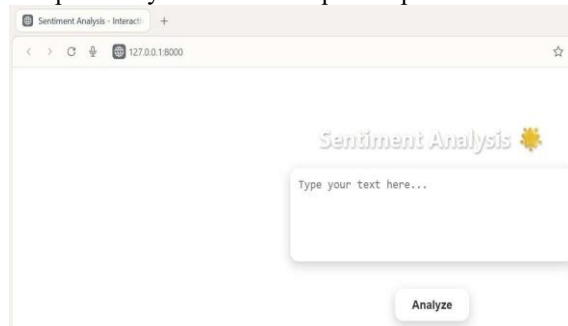


Fig. 3. Sentiment application input interface.

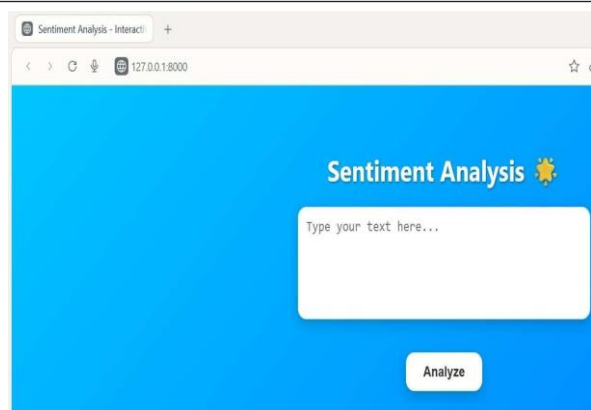


Fig. 4. Blue interface state of the sentiment application.

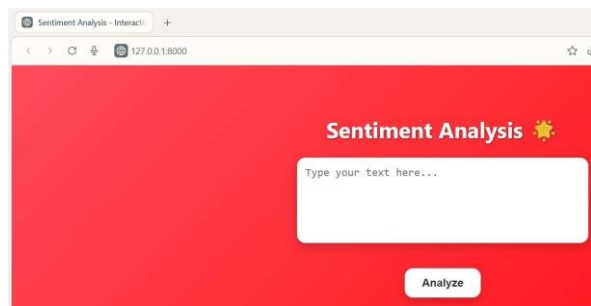


Fig. 5. Red interface state of the sentiment application.

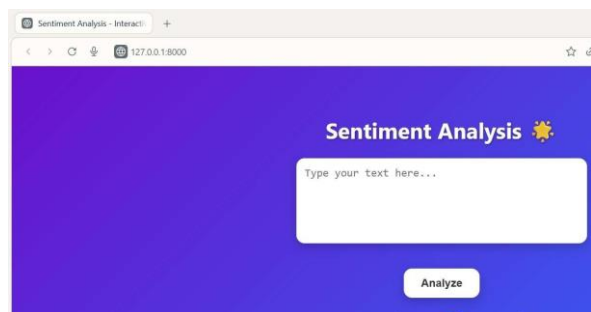


Fig. 6. Violet interface state of the sentiment application.

The three visual states provide immediate perceptual differentiation when combined with textual output. Color alone is not an accessible status indicator; deployment should expose the predicted class in machine-readable text and maintain sufficient contrast for all users. Request-specific polarity values should remain visible alongside the color treatment.

C) Sample Inference Cases

The project report gives short representative input/output cases and response-time examples. These examples illustrate service behavior and are not a substitute for the complete 200-sentence benchmark. Table II reproduces a compact selection of the reported cases.

TABLE II. DOCUMENTED SENTIMENT INFERENCE EXAMPLES

Input phrase	Polarity	Label	Time (s)
The app is amazing!	+0.82	Positive	0.30

This is the worst service ever.	-0.91	Negative	0.40
The design is okay, nothing special.	0.00	Neutral	0.35

D) Reliability, Limitations and Operational Interpretation

FastAPI service boundaries enable client/server decoupling and support endpoint tests. The report provides examples of local hosting, Docker packaging, and operational planning. Nevertheless, throughput under concurrency, environment portability across independent machines, pipeline pass rates, automated rollback, and continuous production monitoring are not supported by full machine-readable logs.

An extended evaluation should collect per-class support and confusion matrices on independent review corpora; latency percentiles and concurrency curves should be derived from recorded traces. A system-level regression suite should include empty strings, punctuation-only inputs, Unicode text, input-length boundaries, mixed sentiment, ambiguous negations, and repeated API requests. These criteria would distinguish reproducible reliability from demonstration-only success.

For real-world adoption, the model should disclose its English-language and lexicon-dependent limits, minimize retained user text, and monitor data drift. Adding a transformer model is an engineering option to benchmark, not an established improvement of the reported implementation. The complete MLOps lifecycle also requires observable evidence of reproducible builds, versioned data, gated releases, and runtime monitoring.

E) Threats to Experimental Validity

Manual assignment of reference labels introduces annotation uncertainty. A well-designed sentiment evaluation should specify whether multiple annotators independently labeled each sentence, how disagreements were reconciled, and whether neutral opinions were represented at sufficient frequency. If the 200 records primarily contain unequivocally positive and negative examples, the reported accuracy may overstate performance on mixed evaluations. Source-specific separation is also relevant: a test set consisting of product reviews can differ substantially from social-media comments or technical support messages.

A second validity concern is the use of illustrative sentiment examples as evidence of generalization. Short sentences with strong lexical cues are expected to be comparatively easy for lexicon-based polarity methods. Challenging cases include negation across clauses, ironic praise, negated intensifiers, quotations, slang, emojis, domain-specific jargon, and complaints containing a combination of positive and negative aspects. The evaluation should distinguish such groups and report counts, rather than extrapolating overall accuracy from a few successful requests.

Response-time measurements depend on whether they include network transport, frontend rendering, first-request initialization, request validation, and model inference. A single arithmetic mean can conceal transient latency spikes, so the measurement plan should record median, 95th-percentile, and

maximum response time alongside the hardware and operating-system configuration. Warm and cold invocation should be identified separately. No percentile claim can be inferred from the mean value alone.

MLOps maturity must also be assessed from executable evidence. A Dockerfile shows a specification for container builds; build verification additionally requires a pinned dependency set, successful image construction, vulnerability review, health checks, and repeated startup success. A Git repository or CI/CD workflow description does not prove that automated deployment is active. The service should record build identifiers and test outcomes for each candidate release. These criteria give a reproducible boundary to operational claims.

F) API Contract and Engineering Test Design

API contracts are particularly important because sentiment labels are delivered to a presentation layer. The request schema should define an accepted textual field, maximum input size, encoding, and behavior for missing values. The response schema should encode the predicted class using a fixed vocabulary and serialize polarity numerically, ideally preserving adequate decimal precision to avoid rounding-induced misinterpretation near zero. An explicit error envelope enables the frontend to distinguish invalid input from server failure and from a genuinely neutral prediction.

Regression testing should use fixed sentences whose expected outputs are recorded together with the TextBlob and dependency versions. Because lexical resources can change between package releases, reproducing the application requires preserving version metadata. Tests should verify that an invalid payload cannot silently produce a neutral classification and that a server exception does not cause the user interface to retain a previously calculated result as current evidence.

Property-oriented checks can enforce invariants even in the absence of a full labeled corpus. The polarity score must remain within its contractual numerical interval, the returned label must belong to the defined three-class set, and deterministic repeated calls under unchanged dependencies must produce identical results. Concurrent requests should not leak input or outputs between sessions. For a stateless service, isolation can be validated by interleaving independent requests and checking response correspondence.

User interface acceptance tests should examine keyboard focus, text-area accessibility, readable contrast, noncolor sentiment indicators, and behavior during slow network responses. A visible loading state prevents repeated submissions while a response is pending. Error messages should be comprehensible and persistent enough for assistive technologies. These engineering requirements are aligned with reliable service delivery but require actual automated or human testing before being claimed as verified features.

G) Operational Monitoring and Model Change Control

A minimal runtime observability layer can measure request counts, status-code distributions, throughput, error rate, and latency, without storing raw customer input. Sentiment-distribution monitoring may track shifts in the proportion of

positive, neutral, and negative predictions over time. Such a shift is not proof of model degradation: it can reflect a genuine change in the underlying text population. Reliable drift investigation therefore compares incoming text characteristics against reference populations and uses newly labeled observations where available.

For model-version governance, changes to polarity thresholds, TextBlob resource versions, API validation rules, or the frontend label mapping should be treated as coupled releases. A production rollout should preserve rollback artifacts and a reproducible validation manifest. Threshold modifications ought to be calibrated on a development set and evaluated on a separate test set, because repeated selection against the same 200 labels can inflate apparent accuracy. Such separation is essential if a modified threshold is reported as a methodological advance.

An advanced transformer alternative may reduce certain lexical limitations, but it increases resource consumption and may introduce calibration and fairness concerns. A disciplined comparison would evaluate a TextBlob baseline and a fine-tuned contextual model on identical data and class definitions, using both quality metrics and operational costs. CPU inference time, model size, memory consumption, and deployment complexity are meaningful alongside classification accuracy. No alternative-model scores are assigned here because the project does not provide a directly comparable trained benchmark.

H) Data Governance and Reproducible Evaluation

Text reviews may contain personal information even when the application appears to handle ordinary opinions. Storage minimization, input retention limits, permission management, and controlled logging should therefore accompany operational deployment. Testing can use deidentified examples where possible, and benchmark releases should include documentation of label provenance and any transformation of the original text. Personal identifiers should not appear in observability dashboards or exception messages.

A reproducible experimental package should contain the 200 reference sentences or an authorized shareable subset, class annotations, a deterministic inference script, dependency versions, test hardware specifications, and a clear account of the metric computation. A three-class confusion matrix and class-support vector would permit independent calculation of macro-F1, precision, and recall. Without these artifacts, the project-level aggregate outcomes remain useful descriptive results but cannot support fine-grained claims about error distribution or robustness.

A final evaluation should additionally separate functional usability from predictive validity. The presence of a working interface, successful HTTP responses, and predictable text formatting demonstrates integration. Accuracy and F1 quantify agreement with a reference labeling procedure. Continuous-integration pass rates, container portability, and resource utilization evaluate operational reliability. Treating these as different assessment dimensions avoids implying that a good classifier score alone establishes production readiness.

I) Interpreting Class Boundaries and Polarity Magnitude

The sign-based mapping is straightforward but the magnitude of polarity should not be confused with calibrated probability. A value such as +0.85 indicates a strongly positive lexical score under the employed tool, not an 85% chance that a human reader would assign the positive label. This distinction matters when integrating sentiment signals into operational decisions. Downstream consumers should not threshold the polarity score as a probability without separate calibration experiments and an explicitly defined ground-truth objective.

Neutrality is similarly ambiguous. A zero score may arise from genuinely neutral wording, absence of recognized sentiment tokens, or cancellation of positive and negative expressions within a sentence. To evaluate this distinction, a future annotated sample could separate objectively neutral descriptions from mixed-polarity passages. Reporting false-neutral rates would help identify when the three-class decision rule suppresses actionable mixed opinions. A fixed tolerance interval around zero is one potential design, but its bounds must be chosen on independent development data and verified on unseen samples.

Polarity estimates can also vary with input length. A longer review may mention a favorable product feature and an unfavorable shipping experience, giving an aggregate score near neutrality even though both opinions are meaningful. Aspect-based methods could preserve the two targets while a document-level label cannot. Such an extension changes the prediction task itself; reported three-class accuracy would no longer be directly comparable unless both systems were evaluated using consistent targets.

J) Deployment Risk and Acceptance Criteria

Operational readiness can be framed through explicit, measurable acceptance criteria. For example, request-schema validation should successfully reject missing text while valid calls return a parseable polarity value and label. A deployment pipeline should reproduce the same responses after a clean build, and a new release should not introduce changes to stored regression examples without review. These criteria are preferable to generic claims of reliability because they map to executable checks and objectively inspectable artifacts.

Availability and resilience require failure-mode testing. If the NLP dependency cannot initialize, the API should return a controlled service error rather than a fabricated neutral label. If a browser loses connectivity, it should display a request failure instead of preserving a previous sentiment result as the latest one. If the incoming text exceeds the configured length, consistent validation prevents performance degradation and potential resource exhaustion. These failure controls are particularly important when the sentiment service is embedded in a customer-support workflow.

Finally, release documentation should distinguish planned engineering mechanisms from verified execution evidence. Source control, dependency pinning, image-building instructions, reproducible initialization, and integration testing form a progressive maturity path. Continuous model monitoring and scheduled retraining would require an

additional labeled-data lifecycle, automated evaluation criteria, and operational governance. For a lexicon-driven system without a trained project-specific model, package and resource versioning may be more immediate priorities than automatic retraining.

K) Error Taxonomy and Targeted Regression Sets

A useful extension to the experimental design is an explicit error taxonomy, allowing the reported overall accuracy to be interpreted in operational terms. False positive sentiment may result from favorable adjectives in negative contexts, while false negative sentiment can arise when criticism is expressed indirectly. False-neutral outputs may be caused by a polarity score of exactly zero after opposing lexical cues cancel. Annotation guidelines should specify whether neutral means neither positive nor negative, balanced opinion, or insufficient evidence, since these definitions lead to different model targets.

Targeted regression cases can be arranged into mutually documented categories: direct praise or complaint, local and long-range negation, comparative constructions, conditional statements, mixed sentiment, sarcasm, intensifiers, nonstandard spellings, and domain-specific vocabulary. Each category should contain sufficient independent instances to permit rate estimates. A trace record may retain a hashed sample identifier, expected label, observed label, polarity, tool version, and request duration. This structure allows errors to be analyzed without embedding confidential customer text into routine application logs.

Threshold tuning should be undertaken only after the evaluation protocol is frozen. Using the same 200 instances to choose a threshold and then to claim the final score would contaminate the assessment. Instead, an annotated development subset can support threshold selection while a separate locked test set estimates generalization. Stratified reporting ensures that classes with few examples are not hidden by a high weighted average. Bootstrap intervals or repeated independent test sets may be considered for uncertainty quantification once the item-level records are available.

L) Service Observability and Integration Architecture

An API-centered architecture should expose operational signals at both transport and application levels. Transport-level monitoring records HTTP status, payload size, endpoint latency, and retry rate. Application-level monitoring tracks empty-input rejections, the distribution of generated sentiment labels, and processing exceptions. These metrics have different interpretations: a rising response latency may indicate a resource bottleneck, while a shift in label proportions may reflect evolving customer opinions rather than a software fault. An incident investigation should therefore correlate operational observations before triggering corrective actions.

The frontend should receive stable machine-readable responses and display them without reconstructing business logic. Centralizing the polarity-to-label mapping within the API prevents differences between browser clients or future mobile applications. Backward-compatible schema changes can be checked against saved request and response examples.

Client applications can implement request timeouts, controlled retries, and clear error indications without duplicating the sentiment engine. This separation supports maintainability even when the underlying NLP library changes.

Dependency reproducibility is important for lexical NLP libraries because tokenizer behavior, linguistic corpora, package versions, and compatibility fixes may change outcomes. A container build should pin dependencies and supply a verified resource initialization procedure. Rebuilding from the same manifest can then be compared with an archived reference response set. This type of invariant-based regression check is especially suitable when the application is not training a new statistical model during each release.

M) Future Comparative Benchmarking

The strongest next research stage would compare the lightweight TextBlob implementation against a classical supervised classifier and a compact pretrained transformer using identical annotated texts. Training, development, and held-out test partitions should be fixed before model selection. Besides class-wise precision, recall, macro-F1, and accuracy, the analysis should include wall-clock request latency, memory footprint, package size, cold-start behavior, and CPU utilization. These characteristics are particularly relevant to resource-constrained organizational deployments.

Interpretability should also be compared on comparable terms. TextBlob provides a continuous polarity score and a relatively accessible lexical basis, but it does not automatically furnish a human-verified rationale for every phrase. A supervised linear model can expose influential n-grams, whereas contextual transformers may require post-hoc feature attribution, adding computation and interpretation caveats. Explanation quality must be evaluated separately from classifier correctness. The comparison would reveal whether additional model complexity materially improves the application under its realistic target workload.

Such a benchmark should retain a distinction between classification performance and MLOps quality. Different NLP engines could share the same FastAPI contracts, frontend and release checks. This makes it possible to compare inference modules without confounding their effect with different deployment paths. If a new model achieves higher predictive scores but incurs unacceptable memory or latency costs, the original lightweight implementation may still be preferable for short interactive reviews. Evidence-based engineering decisions should jointly consider task accuracy, operational constraints, reproducibility, and the sensitivity of the processed text.

V. CONCLUSION

A lightweight sentiment analysis application has been described using TextBlob polarity estimation, a FastAPI inference layer, and browser-based visual feedback. The architecture supports three-class opinion classification and defines a clear progression toward reproducible packaging and deployment controls. The project report states 95% accuracy, 93.5% F1-score, and 0.35 s mean response time for its application evaluation, while the original screenshots confirm presentation-layer states. The work establishes a technically

accessible service pattern, with further research needed for independently reproducible class-level testing, contextual language robustness, and fully verified continuous delivery and monitoring.

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*. Morgan & Claypool, 2012.
- [3] C. J. Hutto and E. Gilbert, "VADER: A parsimonious rule-based model for sentiment analysis of social media text," *Proc. ICWSM*, 2014.
- [4] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly, 2009.
- [5] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment classification using machine learning techniques," *Proc. EMNLP*, pp. 79–86, 2002.
- [6] T. Mikolov et al., "Distributed representations of words and phrases and their compositionality," *Proc. NeurIPS*, 2013.
- [7] J. Devlin et al., "BERT: Pre-training of deep bidirectional transformers for language understanding," *Proc. NAACL-HLT*, pp. 4171–4186, 2019.
- [8] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," *arXiv:1907.11692*, 2019.
- [9] S. Loria, "TextBlob: Simplified text processing," Python package documentation, <https://textblob.readthedocs.io/>.
- [10] A. Joulin et al., "Bag of tricks for efficient text classification," *Proc. EACL*, pp. 427–431, 2017.
- [11] Z. Yang et al., "XLNet: Generalized autoregressive pretraining for language understanding," *Proc. NeurIPS*, 2019.
- [12] V. Sanh et al., "DistilBERT, a distilled version of BERT," *arXiv:1910.01108*, 2019.
- [13] A. Vaswani et al., "Attention is all you need," *Proc. NeurIPS*, pp. 5998–6008, 2017.
- [14] D. Sculley et al., "Hidden technical debt in machine learning systems," *Proc. NeurIPS*, pp. 2503–2511, 2015.
- [15] M. Baylor et al., "TFX: A TensorFlow-based production-scale machine learning platform," *Proc. KDD*, pp. 1387–1395, 2017.
- [16] E. Breck et al., "The ML test score: A rubric for ML production readiness," *Proc. IEEE Big Data*, 2017.
- [17] L. Buitinck et al., "API design for machine learning software: Experiences from the scikit-learn project," *ECML PKDD Workshop*, 2013.
- [18] P. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (MLOps): Overview, definition, and architecture," *IEEE Access*, vol. 11, pp. 62966–62979, 2023.
- [19] D. Tamburri, "Sustainable MLOps: Trends and challenges," *Proc. IEEE/ACM ICSE Workshops*, 2020.
- [20] M. Zaharia et al., "Accelerating the machine learning lifecycle with MLflow," *IEEE Data Engineering Bulletin*, vol. 41, no. 4, 2018.
- [21] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux Journal*, no. 239, 2014.
- [22] FastAPI, "FastAPI framework documentation," <https://fastapi.tiangolo.com/>.
- [23] Docker Inc., "Docker documentation," <https://docs.docker.com/>.
- [24] GitHub, "GitHub Actions documentation," <https://docs.github.com/actions>.
- [25] M. Fowler, "Continuous integration," *MartinFowler.com*, 2006.
- [26] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," *Proc. ICML*, pp. 233–240, 2006.
- [27] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.
- [28] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [29] A. McCallum and K. Nigam, "A comparison of event models for naive Bayes text classification," *AAAI Workshop on Learning for Text Categorization*, 1998.
- [30] A. S. McCallum, "MALLET: A machine learning for language toolkit," *University of Massachusetts Amherst*, 2002.