

SmartView: A Mobile Application for Google Play Review Sentiment Analysis Using Traditional Machine Learning and BiLSTM

Madhuri Sanjiv Bhadane¹, Devshree Sopan Aavhad¹, Kavita Rajendra Nikam¹,

Janhavi Kalpesh Jadhav¹, Ms. P. S. Pawar²

Students, Department of Information Technology¹⁻⁴

Project Guide, Department of Information Technology⁵

MVP's Rajarshi Shahu Maharaj Polytechnic, Nashik, Maharashtra

Abstract: Mobile applications receive large volumes of user reviews that provide feedback about usability, performance, reliability, and feature expectations. Manually examining these reviews is time-consuming. This paper proposes SmartView, a mobile application for analyzing publicly accessible Google Play reviews using sentiment classification and visual analytics. The proposed workflow collects accessible reviews for a selected application, preprocesses review text, and compares traditional machine-learning models with a Bidirectional Long Short-Term Memory (BiLSTM) network. The evaluation plan uses accuracy, precision, recall, macro-F1 score, confusion matrices, and inference time. A Flutter interface presents sentiment distributions and review-level predictions. As implementation and experiments are not yet complete, this manuscript presents the proposed method and evaluation plan; it does not claim measured model performance.

Keywords: SmartView, Sentiment Analysis, Google Play Reviews, Machine Learning, BiLSTM, Text Classification, Flutter, Review Analytics

I. INTRODUCTION

Mobile applications are frequently updated, and user reviews provide direct feedback about application quality. Reviews may describe crashes, slow performance, confusing interfaces, missing features, or positive experiences. Large collections are difficult to inspect manually. Sentiment analysis uses natural-language processing and machine learning to estimate the polarity expressed in text, often represented as positive, negative, or neutral.

SmartView is proposed as a mobile-first review analysis tool. A user selects a publicly listed Google Play application, retrieves available reviews through a compatible data source, and views sentiment summaries and individual predictions. The research compares traditional machine-learning methods with BiLSTM under a consistent evaluation protocol.

The central problem addressed by SmartView is not simply assigning a positive or negative label to a single sentence. A useful review-analysis tool must also make large collections understandable, show how predictions were produced, and communicate the limits of the available data. A high average star rating can hide recurring complaints, while a small number of extreme reviews may not represent the broader user experience. Review text therefore provides complementary evidence that can be summarized alongside rating distributions and review counts.

The proposed application is designed for mobile access so that a user can enter an application name or Play Store URL and receive a compact analytical summary. The backend performs review retrieval and inference rather than placing model-training workloads on the phone. This separation keeps the mobile client lightweight and allows the model pipeline to be updated centrally. SmartView is intended as a decision-support tool for developers, students, and product teams; it does not replace manual investigation of individual reviews.



II. LITERATURE REVIEW

Pang and Lee surveyed opinion mining and sentiment analysis, outlining approaches for classifying subjective language and summarizing opinions [1]. Traditional text-classification pipelines commonly represent documents using term frequency-inverse document frequency (TF-IDF) and train classifiers such as Logistic Regression, linear Support Vector Machines (SVM), or Multinomial Naive Bayes. These methods provide efficient and reproducible baselines. Hochreiter and Schmidhuber introduced Long Short-Term Memory (LSTM) networks to address limitations in learning long-range dependencies in recurrent neural networks [2]. Bidirectional recurrent models process a sequence in forward and backward directions and can capture contextual evidence from both sides of a word. BiLSTM is a useful comparison candidate, but its performance depends on dataset quality, class balance, preprocessing, and hyperparameter selection.

III. PROPOSED SYSTEM

The proposed system contains application identification, review collection, text preprocessing, sentiment prediction, and analytics presentation. The user enters an application name or Play Store URL. The backend resolves the application identifier and requests available reviews through a selected data-access method. Review retrieval must follow source terms and technical limits. Google's official Play Developer API is intended for reviews of apps managed by the authenticated developer account and does not itself provide arbitrary third-party review access.

Traditional models use TF-IDF features with Multinomial Naive Bayes, Logistic Regression, and Linear SVM. The BiLSTM pipeline tokenizes text, converts tokens to sequences, pads them to a fixed length, and learns contextual representations. Both approaches should use the same labelled dataset split.

The planned mobile client uses Flutter. A Python API exposes endpoints for app lookup, review retrieval, prediction, and aggregate analytics. The system should report retrieval errors clearly rather than presenting fabricated or stale data as live results.

The system follows a client-server design. The Flutter client presents the search form, filters, sentiment cards, review examples, and charts. A Python REST API validates requests, resolves the target application identifier, invokes the source adapter, calls the selected model, and returns structured results. The preprocessing service applies the same normalization rules used during model development. A model registry or configuration setting identifies the active classifier and its version so that predictions can be interpreted consistently.

The review pipeline should retain essential metadata such as the application package identifier, retrieval timestamp, source status, and model version. Cache entries can reduce repeated requests and improve responsiveness, but cached results must be labelled with their retrieval time. Persistent storage should avoid collecting unnecessary personal information. If individual review text is retained, the application should document the purpose, retention period, access controls, and applicable source terms.

Review-source constraint: Google's official Play Developer API requires authorized access and is intended for reviews associated with apps managed through the developer account [3]. For third-party public apps, the system must use a permitted data source or a clearly documented dataset; an unofficial scraper may be rate-limited or stop working [4]. The application must display source status and retrieval time and must not imply that every review is always available.

IV. EXISTING SYSTEM

Users commonly inspect reviews manually or rely on an average star rating. Manual inspection becomes inefficient as the number of reviews grows, while star ratings alone do not explain the reasons behind satisfaction or dissatisfaction. Some sentiment tools also lack transparent model comparisons and reproducible evaluation. SmartView proposes review-level classification, aggregate visualization, and an explicit model comparison.

In a manual workflow, a user opens the store listing, reads a limited sample of reviews, and forms an impression based on the visible comments and star rating. This approach is straightforward but becomes slow when an application has



thousands of reviews or when feedback must be compared across releases. Star ratings also do not explain which feature caused dissatisfaction, and rating distributions can be affected by review volume and user self-selection. Some existing analytics tools provide rating summaries or keyword counts, but these may not offer a transparent comparison between traditional machine-learning baselines and a neural sequence model. SmartView's proposed contribution is the combination of a mobile-first interface, a replaceable review-source adapter, side-by-side model evaluation, and explicit communication of data and prediction limitations. These are design goals and should not be interpreted as a claim that every existing product lacks such functionality.

V. SYSTEM ARCHITECTURE

The system uses a Flutter mobile application, Python REST API, review-source adapter, preprocessing component, two model pipelines, and a storage/cache layer. The app sends a package identifier to the API. The adapter retrieves accessible reviews; preprocessing prepares text; models predict sentiment; and the API returns predictions and aggregate statistics to the dashboard.

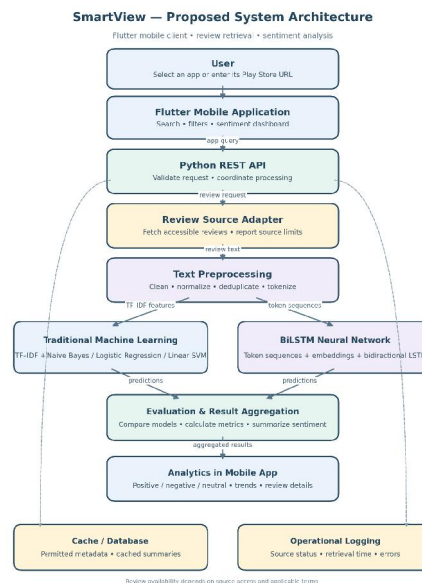


Fig. 1. Proposed SmartView system architecture

Figure 1 represents the logical data flow rather than a deployment diagram for a completed production service. The user submits a search request through the Flutter interface. The API validates the request and forwards it to the review-source adapter. Retrieved text is normalized and passed to either the TF-IDF-based traditional classifier or the token-sequence BiLSTM. Both pipelines return review-level labels and confidence information where supported. The aggregation layer computes counts and percentages, then sends the results to the mobile analytics screen. The cache stores permitted summaries and retrieval metadata; it does not replace the source of record.

VI. WORKING PRINCIPLE

1) The user searches by app name or pastes a Play Store URL. 2) SmartView resolves the package identifier and checks source availability. 3) The backend retrieves an allowed batch of reviews and records source and retrieval time. 4) Text is normalized and duplicates are handled. 5) The selected model predicts positive, negative, or neutral sentiment. 6) The dashboard shows review counts, sentiment distribution, examples, and evaluation metrics. 7) Users can filter results and revisit cached analyses.



VII. SYSTEM MODULES

1) Application Search: accepts an app name or URL. 2) Review Collection: retrieves accessible reviews and reports source limitations. 3) Text Preprocessing: normalizes text and prepares model input. 4) Traditional ML Classifier: trains and evaluates TF-IDF baselines. 5) BiLSTM Classifier: trains a sequence model. 6) Analytics Dashboard: visualizes sentiment and metrics. 7) Evaluation and Export: produces confusion matrices and result tables. 8) History and Cache: stores permitted metadata and analysis summaries.

VIII. EXPERIMENTAL METHODOLOGY AND EVALUATION PLAN

This paper presents a proposed evaluation protocol. No training or test experiment was executed for this draft; therefore, dataset size, measured scores, and runtime are not available. For a reproducible study, select a labelled review dataset and document its source, collection date, number of records, label definitions, language coverage, class distribution, and filtering rules. Publicly retrieved live reviews should be treated separately from the labelled evaluation set.

Use a stratified training/validation/test split with no duplicate or near-duplicate review leakage across partitions. Compare Multinomial Naive Bayes, Logistic Regression, and Linear SVM using TF-IDF with a BiLSTM trained on token sequences. Use identical test examples and report accuracy, macro-precision, macro-recall, macro-F1, confusion matrix, inference latency, and model size. Select hyperparameters on validation data only and fix random seeds where applicable.

Expected outcome (qualitative): TF-IDF-based traditional classifiers are expected to provide efficient baselines. A BiLSTM may capture sequence context when sufficient labelled data are available, but it may require more training time and tuning. These are hypotheses to be tested, not measured conclusions.

Table I. Results to be completed after running the evaluation

Model	Accuracy	Macro-F1	Inference time	Status
Multinomial Naive Bayes + TF-IDF	Not measured	Not measured	Not measured	Pending evaluation
Logistic Regression + TF-IDF	Not measured	Not measured	Not measured	Pending evaluation
Linear SVM + TF-IDF	Not measured	Not measured	Not measured	Pending evaluation
BiLSTM	Not measured	Not measured	Not measured	Pending evaluation

No numerical results are reported because the models have not been trained and evaluated for this manuscript. Insert measured test-set values only after completing the protocol described above.

Review data can contain spelling errors, abbreviations, emojis, multiple languages, sarcasm, and mixed positive and negative opinions. Preprocessing must be applied consistently to training and inference data. Duplicate reviews and near-duplicates should be identified before splitting the dataset, otherwise similar text may appear in both training and test partitions and inflate the apparent score. Where labels are derived from star ratings, the mapping from ratings to positive, neutral, and negative classes must be stated because star ratings are only an imperfect proxy for textual sentiment.

The system should return a clear error when the source is unavailable, the target application cannot be resolved, or a request exceeds an allowed limit. It should not silently substitute old cached data for fresh results. Review text and application metadata should be transmitted over HTTPS, API requests should be validated, and secrets should remain on the server rather than inside the mobile application. Logging should focus on operational metadata and avoid unnecessary storage of personal data.



IX. ADVANTAGES AND LIMITATIONS

IX-A. DATA QUALITY, PRIVACY, AND RELIABILITY

IX-B. ANALYTICS AND EVALUATION METRICS

Expected advantages include faster review triage, consolidated sentiment summaries, visibility into recurring praise and complaints, and reproducible model comparison. Future extensions may include aspect-based sentiment analysis, multilingual reviews, topic discovery, and trends across app versions.

Limitations include rate limits or changes in unofficial review extraction, sampling bias, language variation, sarcasm, mixed sentiment, and classification errors. Predictions are estimates, not ground truth, and should be interpreted with these limitations.

The mobile dashboard can display total reviews analyzed, positive/neutral/negative counts, sentiment percentages, recent review examples, and filters by date or rating when the source provides the required metadata. Each chart should show its denominator and retrieval time so that users do not confuse a sampled batch with the full lifetime review history. A confidence threshold may be used to flag uncertain predictions for manual inspection, but the threshold must be selected using validation data rather than chosen after looking at the test set.

Accuracy measures the proportion of all predictions that are correct, but it can be misleading when one sentiment class dominates. Macro-precision, macro-recall, and macro-F1 give each class equal weight and should be reported together with a confusion matrix. Inference latency and model size help explain practical trade-offs for mobile-facing use. Results should include the dataset source, number of examples per class, split strategy, random seed, tokenization rules, maximum sequence length, optimizer, epochs, batch size, and the criteria used for model selection.

X. CONCLUSION

Research status: this manuscript describes the proposed system and evaluation protocol. No empirical evaluation has been conducted for this draft.

This paper proposes SmartView, a mobile application for analyzing publicly accessible Google Play reviews through sentiment classification and visual analytics. The research compares traditional TF-IDF-based classifiers with BiLSTM under a shared evaluation protocol. Since implementation and experiments are pending, no numerical performance claims are made. Future work will implement the pipeline, evaluate it on a documented labelled dataset, and report reproducible results.

SmartView's intended value lies in turning scattered user comments into a structured overview while retaining access to individual reviews for context. Future work should evaluate the system on a documented labelled dataset, examine errors by language and sentiment class, and investigate aspect-level categories such as performance, usability, reliability, pricing, and feature requests. A production release should also test API reliability, retrieval limits, accessibility, and the consistency of results across repeated requests.

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, doi: 10.1561/1500000011.
- [2] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.
- [3] Google for Developers, "Google Play Developer API: reviews.list." Available: <https://developers.google.com/android-publisher/api-ref/rest/v3/reviews/list>
- [4] JoMingyu, "google-play-scraper," open-source software repository. Available: <https://github.com/JoMingyu/google-play-scraper>. Accessed: 11 Oct. 2026.
- [5] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

