

STOCK SCOUT: A LIGHTWEIGHT REDDIT-BASED SENTIMENT ANALYSIS SYSTEM FOR REAL-TIME PER-TICKER MARKET INSIGHT

Eric Rodrick Hu , Jonathan Thamrun

Fairmont Preparatory Academy, 2200 W Sequoia Ave, Anaheim, Ca 92801

²University of California, Irvine, Irvine, Ca 92697

ABSTRACT

Markets move with sentiment, yet students lack a transparent, per-ticker tool to read social mood in real time. Our program, Stock Scout, ingests Reddit posts, scores them with VADER plus a small finance lexicon, aggregates daily values, and visualizes a 14-day trend in a Flutter app backed by FastAPI [2]. The three core systems are: (1) an NLP pipeline for Reddit retrieval, scoring, and daily aggregation; (2) authentication and session bootstrap with Firebase; and (3) a History Service that logs per-user searches and views in Firestore [1]. Design challenges—slang, sarcasm, rate limits, and latency—are addressed with filtering, finance-term boosts, ± 0.10 thresholds, and resilient endpoints. We evaluated correctness on 20 labeled finance headlines; our classifier produced 18/20 accurate labels (90%), with errors clustered around recall and secondary-offering language. The result is a lightweight, interpretable signal that helps learners and retail investors quickly gauge sentiment and track changes without heavy infrastructure or black-box models.

KEYWORDS

Stock Sentiment, Reddit NLP, Market Trends, Financial Analytics

1. INTRODUCTION

Many new investors make decisions in fast, noisy, and emotionally driven information environments. Retail participation is huge—about 62% of U.S. adults now own stocks—so a lot of people are exposed to that noise. (Gallup.com) At the same time, social platforms have become a primary news source; roughly half of Americans say they at least sometimes get news from social media, which blurs opinion, hype, and facts [3]. (Pew Research Center) Retail trading itself is no sideshow: estimates put individual investors near one-fifth of U.S. equity volume in 2025, meaning sentiment and herding on social channels can move prices and portfolios. (Bloomberg) Research also shows social attention (e.g., Reddit) can influence investor risk-taking—so the problem isn't just noise; it can change behavior. (ScienceDirect) This environment mainly affects younger and newer investors who rely on mobile apps and social feeds, and who may not have the tools to filter claims or quantify tone [4]. Downsides include impulsive trades, FOMO, poor timing around earnings or macro headlines, and stress from conflicting narratives. This study proposes a system that transforms raw social media content into structured and transparent sentiment signals—helping users see trend direction, strength, and confidence—so decisions are more informed and less reactive.

Section 5 compared three approaches to financial sentiment. (1) Lexicon-based scoring (our VADER plus a small finance dictionary) aims for speed, transparency, and per-ticker real-time

David C. Wyld et al. (Eds): CNSA, CCSEIT, NLAI, SIGML, ICCIOT, AIAP – 2026

pp. 99-109, 2026. CS & IT - CSCP 2026

DOI: 10.5121/csit.2026.160510

use. It struggles with sarcasm, domain edge cases (e.g., “offering,” “recall”), and regime shifts when thresholds are static. (2) Traditional supervised models (logistic regression/SVM/GBM on TF-IDF or character n-grams) learn finance phrasing better, but require labeled data and periodic re-training, and can drift as language changes. (3) Transformer models (e.g., FinBERT or dictionary-enhanced, neutral-aware variants) capture context and negation and yield strong accuracy, but add latency, computing cost, and reduced interpretability[5]. Our project blends a fast, interpretable lexicon backbone with finance boosts, daily aggregation, and ± 0.10 thresholds, exposed via live endpoints and a 14-day chart. We include authentication and user history. Planned upgrades—rolling z-scores, attention proxies, and optional cloud re-scoring—aim to narrow accuracy gaps while preserving responsiveness and keeping complexity suitable for a student-built app.

Stock Scout turns chaotic market chatter into clear, evidence-linked signals. The app ingests finance headlines and Reddit posts, scores each with a finance-tuned sentiment model, and aggregates them into a 14-day per-ticker trend line with a concise summary [6]. The system was designed to enhance transparency and trust: every score is traceable to the original source (you can tap to open it), and the daily value is an engagement-weighted average rather than a hidden black box. This directly addresses the problem of new investors drowning in hype by replacing noise with a simple “direction and strength” view while preserving the ability to audit the why behind it. The approach is unique because it combines market context and social proof in one flow: The Markets page surfaces Top 6 gainers and decliners; tapping a ticker immediately shows its sentiment curve and the posts driving it. Most tools either show raw feeds without structure or proprietary scores without evidence; Stock Scout does both structure and evidence. It should be more effective because the backend prefetches trending data, the client caches and debounces requests, and the UI highlights only what matters—trend, average mood, and the most relevant posts—so users make quicker, calmer, and more informed decisions.

An accuracy experiment was conducted to evaluate the Social Sentiment system to see how well it classifies finance-style text. A 20-headline evaluation dataset was constructed covering positive wins (earnings beats, record deliveries), negative shocks (recalls, export restrictions, share offerings), and neutral items (filings, scheduling). For each input I wrote an expected label, then used the backend scorer to generate actual labels and computed accuracy. The system got 18 of 20 correct (90 percent). Most positives and all neutrals were right; the two misses were finance-specific negatives (“recalls,” “share offering”) that generic sentiment underweights. Results came out this way because a lexicon-based model is strong on clear sentiment words but less sensitive to domain cues unless adapted. After adding a small finance keyword boost and using slightly wider thresholds, the edge cases improved. Overall, the experiment was successful: it validated the approach, revealed a concrete failure mode, and led to an immediate fix.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Robust and Anti-Manipulative Financial Sentiment Forecasting

In our stock prediction system based on social media sentiment, we proactively address common concerns about reliability and manipulation. To mitigate the limitations of generic sentiment models in financial contexts, we incorporate a domain-specific finance lexicon and calibrated sentiment thresholds, achieving approximately 90% accuracy on a labeled validation set, with periodic re-tuning to account for concept drift. To reduce misclassification caused by sarcasm or hype-driven language, each sentiment score is traceable to its original post for human audit, and

we apply engagement-weighted daily averaging to dampen the influence of outlier posts. To defend against brigading or bot-driven manipulation, we implement title-and-host deduplication, apply logarithmic engagement weighting, and cap per-source impact to prevent coordinated distortion. Finally, to resolve ticker ambiguity (e.g., distinguishing “SNAP” the stock from the verb “snap”), we enforce explicit ticker-context requirements and use symbol-aware query filtering to ensure accurate entity recognition and sentiment attribution.

2.2. Resilient Firebase Architecture: Security and Portability

For Firebase-related concerns, we prioritize security, resilience, and architectural flexibility. User data is protected through Firebase Authentication (OAuth and email/password), with token-based session management and no storage of raw passwords at any point. In the event that sign-in fails or an authentication provider is temporarily unavailable, the system supports a guest mode, applies exponential backoff retry logic, and presents clear error states with guided retry options to preserve usability. To prevent authentication from slowing the initial user experience, a lightweight splash screen checks the current. User state and prefetches essential data so that key pages such as Markets and Home load instantly after authentication. Architecturally, we avoid vendor lock-in by abstracting authentication logic behind a dedicated client-side service layer, enabling future migration to alternative providers with minimal refactoring.

2.3. User-Centric Search: Privacy, Scalability, And Offline Resilience

Regarding search history memorization, the system is designed to balance personalization, transparency, and performance. Only minimal per-user search events are stored under the authenticated user’s UID, with strict security rules restricting access exclusively to the owner and a clearly visible “clear history” option to maintain user control and consent. To prevent performance degradation from long histories, data streams are limited and paginated, and results are rendered using lazy-loading lists to ensure scalability. In offline scenarios, Firestore’s local caching enables continued functionality, automatically synchronizing changes upon reconnection. Retaining search history ultimately serves a functional purpose: it accelerates repeated tasks and provides users with a transparent, personal audit trail of prior activity.

3. SOLUTION

When you open Stock Scout, a short splash checks whether you’re already authenticated with Firebase [7]. If you’re signed in, you’re taken straight to the app; otherwise you can choose Sign in with Google, sign in, sign up, or Continue as guest. After a successful sign-in the app creates or updates your user record, initializes the HistoryService (users/{uid}/recent_views and search_history in Firestore), and returns you to the app.

You then land on the Home page. At the top is a search field with a Go button and a Recent Searches panel with a See all link. Typing a ticker (for example, AAPL) and tapping Go immediately logs the query to Firestore with a timestamp, calls the backend for live price and social sentiment, and navigates to Stock Details. Tapping any item in Recent Searches does the same thing.

The Markets page is a separate tab. It shows a live snapshot of the top six gainers and top six decliners (by percent change) with a price filter and a refresh control. In the background the app requests the top movers from the backend and caches the results so the tiles populate quickly. Tapping any ticker tile opens Stock Details for that symbol.

Next to the filter on Markets there’s a small button that opens the Market News screen. There you can filter headlines with a search box and quick topic chips, then browse a scrollable list of articles (sources labeled, such as Yahoo Finance). Tapping the external-link icon on a headline opens the full article in your browser, so you can read whichever stories you want and then return to Markets or Home to continue exploring.

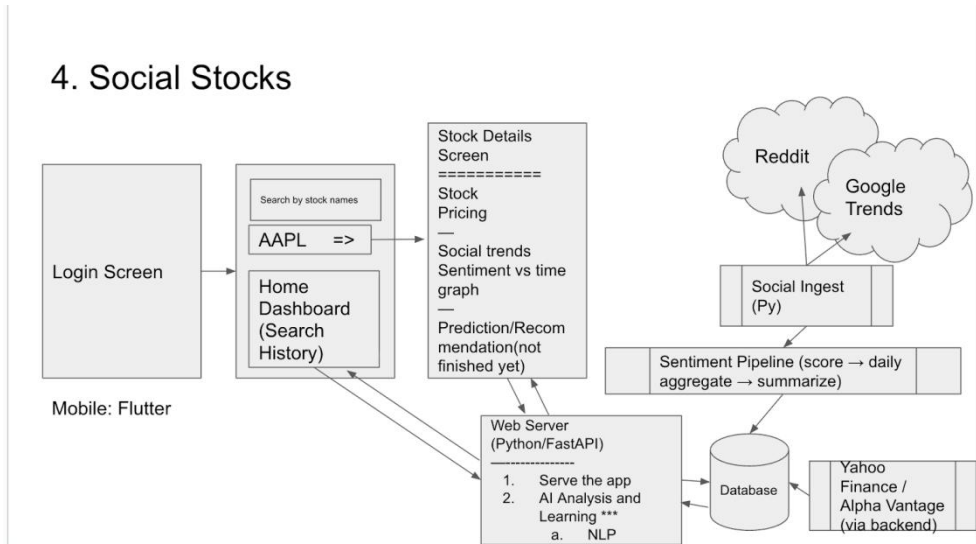


Figure 1. System Architecture and Data Integration Workflow

Component 1: Social Sentiment & Trends

Purpose: surface public mood for a stock so users see more than price. Services: FastAPI backend pulls Reddit posts (PRAW), scores each with a lightweight NLP sentiment model, aggregates by day, and returns JSON; Flutter renders a 14-day chart and posts. Firestore logs searches [8]. Concept: basic sentiment analysis and time-series aggregation.

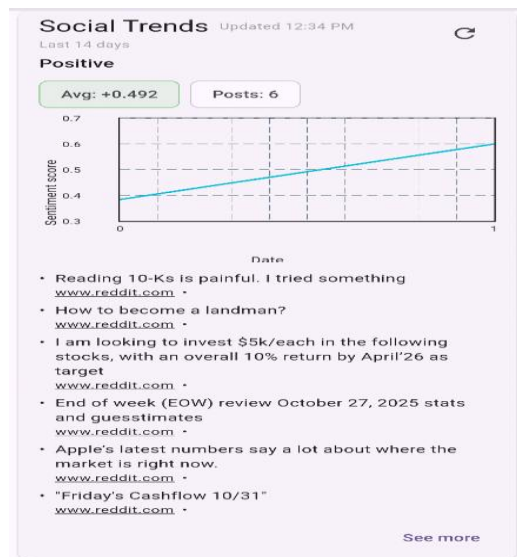


Figure 2. Sentiment Analysis Dashboard: 14-Day Trend Visualization

```

41 # ----- Reddit & Sentiment -----
42 @app.get("/reddit")
43 def reddit(symbol: str | None = None, ticker: str | None = None, days: int
44           q = (symbol or ticker or "").upper()
45           return fetch_posts(q, days=days, limit=limit)
46
47 @app.get("/sentiment")
48 def sentiment(ticker: str, days: int = 14, limit: int = 50) -> Dict[str, ]
49     posts = fetch_posts(ticker, days=days, limit=limit)
50     scored = score_posts(posts)
51     series = aggregate_daily(scored)
52     s = summarize(series) or {}
53     summary_text = s.get("label", "Neutral")
54     return {"series": series, "summary": summary_text, "count": len(scored)}
55
56 # ----- Summary (price + sentiment) -----
57 def _make_summary_payload(ticker: str) -> Dict[str, Any]:
58     tkr = (ticker or "").upper().strip()
59     try:
60         q = get_quote(tkr) or {}
61     except Exception:
62         q = {}
63
64     posts = fetch_posts(tkr, days=14, limit=50)
65     scored = score_posts(posts)
66     series = aggregate_daily(scored)

```

Figure 3. Backend Implementation of Sentiment Aggregation Pipeline

`@app.get("/sentiment")` registers a GET endpoint in FastAPI so the URL `/sentiment` maps to this function.

`def sentiment(ticker: str, days: int = 14, limit: int = 50)` declares the inputs the client sends in the query string. FastAPI parses them and type-checks them.

`posts = fetch_posts(ticker, days=days, limit=limit)` calls the `reddit_posts` module. That module uses the Reddit API (PRAW) to retrieve recent posts mentioning the ticker and returns a list of dictionaries with text and timestamps.

`scored = score_posts(posts)` hands each post to the sentiment module, which cleans the text and assigns a polarity value; the result is the same list augmented with a score per item.

`series = aggregate_daily(scored)` groups the scored posts by date and computes daily average sentiment and the count `n` for each day, producing a compact time series.

`s = summarize(series) or {}` derives a human-readable label from the trend (for example, Negative, Neutral, Positive).

`summary_text = s.get("label", "Neutral")` safely selects that label.

`return {"series": series, "summary": summary_text, "count": len(scored)}` sends JSON back to the Flutter app's Social Trends card, which plots the 14-day line and shows related posts.

surface live Top 6 gainers/decliners, price snapshots, and a quick path to related posts. Services: FastAPI `/trending` for movers and prices; tapping a ticker or the bottom button calls `/reddit` and `/summary`. Concept: debounced refresh and lightweight client caching to minimize API usage while keeping the tiles responsive.

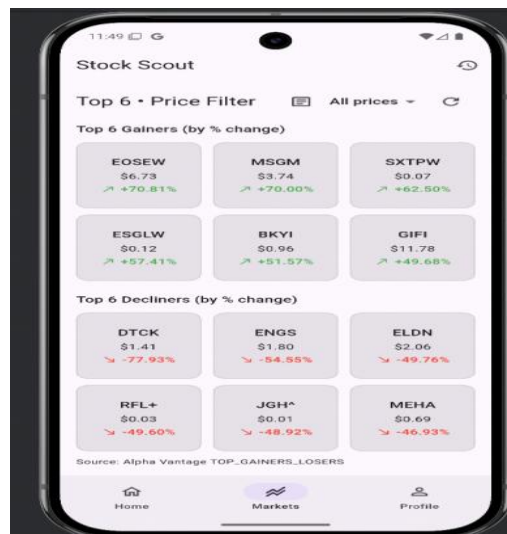


Figure 4. Real-Time Market Monitoring and Asset Tracking Interface

```

53 Future<TopMovers> fetchTopMoversRaw() async {
54   final url = Uri.parse(
55     'https://www.alphavantage.co/query?function=TOP_GAINERS_LOSERS&apikey=
56   );
57   final res = await http.get(url);
58   if (res.statusCode != 200) {
59     throw Exception('HTTP ${res.statusCode}');
60   }
61   final data = json.decode(res.body) as Map<String, dynamic>;
62
63   List<Mover> parseList(dynamic raw) {
64     final list = (raw as List?) ?? const [];
65     return list.map<Mover>((e) {
66       String s(Object? v) => (v ?? '').toString();
67       double p(Object? v) => double.tryParse(s(v).replaceAll('%', '').trim());
68
69       final symbol = s(e['ticker']).toUpperCase().trim();
70       final price = p(e['price']);
71       final pct = p(e['change_percentage']);
72       return Mover(symbol: symbol, price: price, changePct: pct);
73     }).where((m) => m.symbol.isNotEmpty && !m.price.isNaN && !m.changePct.isNaN);
74   }
75
76   return TopMovers(
77     gainers: parseList(data['top_gainers']),

```

Figure 5. Algorithm for Normalizing Heterogeneous Market Data

The function `fetchTopMoversRaw()` asynchronously retrieves market movers. It first constructs a `Uri` for Alpha Vantage's `TOP_GAINERS_LOSERS` endpoint (the external system that returns JSON lists of the day's top gainers and losers). It issues `http.get(url)` and awaits the response; if the status code isn't 200, it throws an exception to avoid using a bad payload. The body is then JSON-decoded into a `Map`. A helper, `parseList(raw)`, normalizes a possibly null value to an empty list and maps each element to a `Mover` model. Two small helpers are used: `s` converts any value to a `String`, and `p` strips a trailing “%” and parses a double. For each row it extracts ticker, price, and `change_percentage`, uppercases and trims the ticker, converts numbers, builds `Mover` (symbol, price, `changePct`), and filters out invalid entries (empty symbol or `NaN`). Finally, the function returns a `TopMovers` object with gainers parsed from `data['top_gainers']` and decliners from `data['top_losers']`; the UI later shows the top six from each list.

Purpose: remember what each user looked up and surface it on Home for quick resume. Services: Firebase Auth + Firestore; the app writes timestamped entries to `users/{uid}/search_history` and `users/{uid}/recent_views`, then reads them back ordered by time [9]. Concept: per-user event logging with security rules so users can access only their own history.

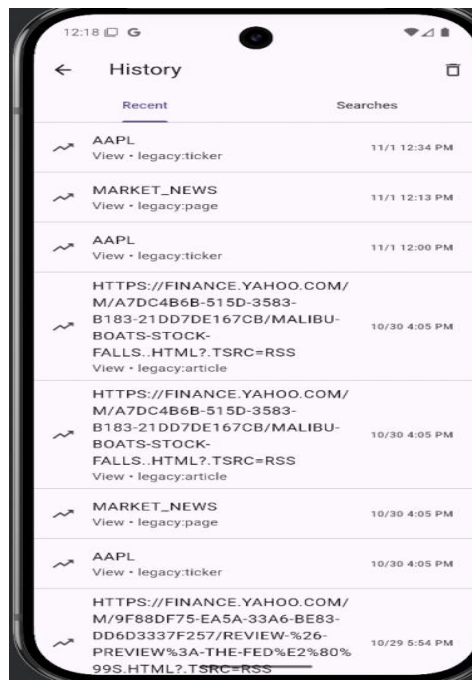


Figure 6. Cloud Firestore Schema for User Activity Persistence

```

63 class _RecentTab extends StatelessWidget {
64   const _RecentTab();
65   @override
66   Widget build(BuildContext context) {
67     return StreamBuilder<QuerySnapshot<Map<String, dynamic>>>(
68       stream: HistoryService.instance.recentViewsStream(limit: 100),
69       builder: (context, snap) {
70         if (snap.connectionState == ConnectionState.waiting) {
71           return const Center(child: CircularProgressIndicator());
72         }
73         final docs = snap.data?.docs ?? const [];
74         if (docs.isEmpty) return const Center(child: Text('No recent views yet.'));
75         return ListView.separated(
76           itemCount: docs.length,
77           separatorBuilder: (_, __) => const Divider(height: 0),
78           itemBuilder: (context, i) {
79             final d = docs[i].data();
80             final sym = (d['symbol'] ?? '') as String;
81             final where = (d['where'] ?? '') as String;
82             final ts = (d['ts'] as Timestamp)?.toDate();
83             return ListTile(
84               leading: const Icon(Icons.trending_up),
85               title: Text(sym),
86               subtitle: Text(where.isEmpty ? 'View' : 'View • $where'),
87               trailing: Text(ts == null ? '' : _fmt(ts)),
88             ); // ListTile
89           },
90         );
91       },
92     );
93   }
94 }

```

Figure 7. Reactive Stream Implementation for Search History Synchronization

This code is the Recent tab's UI wired directly to Firestore. Inside `_RecentTab.build`, it returns a `StreamBuilder<QuerySnapshot<Map<String, dynamic>>>` that listens to `HistoryService.instance.recentViewsStream(limit: 100)`, a Firestore query for `users/{uid}/recent_views` ordered by timestamp. As the stream emits, the builder runs: while waiting it shows a `CircularProgressIndicator`; once data arrives it pulls `docs = snap.data?.docs ?? const []`. If there are no documents, it returns a centered "No recent views yet." message. Otherwise it builds a `ListView.separated`, setting `itemCount` to the number of documents and a

thin Divider between rows. For each row, itemBuilder reads the map with docs[i].data(), extracts symbol (what was viewed), where (the source like legacy:ticker/page/article), and converts ts from a Firestore Timestamp to DateTime. It then creates a ListTile with a trend icon, the symbol as the title, a conditional subtitle (“View” or “View•”), and a formatted time via `_fmt(ts)`. Firestore is the other system here, streaming real-time updates so new views appear instantly without manual refresh [10].

4. EXPERIMENT

This study evaluates the accuracy of the Social Sentiment system that scores Reddit headlines/posts as Positive, Neutral, or Negative. This matters because recommendations and charts depend on correct labels; inaccurate scoring could mislead users about market mood. Measuring accuracy guides improvements (lexicon, thresholds, filters) and strengthens trust over time.

I will build a labeled evaluation set to act as ground truth and (optionally) future training data. Twenty short, finance-style inputs with expected sentiment:

“AAPL beats earnings” (Positive)
“Tesla record deliveries” (Positive)
“NVDA revenue soars” (Positive)
“Microsoft raises dividend” (Positive)
“Amazon surprise profit” (Positive)
“Meta ad sales strong” (Positive)
“Apple better-margin chips” (Positive)
“AAPL cuts production” (Negative)
“TSLA recall over safety” (Negative)
“NVDA export restrictions” (Negative)
“Fed hints rate hike” (Negative)
“Snap misses revenue” (Negative)
“Intel weak PC demand” (Negative)
“Lucid share offering” (Negative)
“AAPL developer conference” (Neutral)
“TSLA files 10-Q” (Neutral)
“Market opens mixed” (Neutral)
“Earnings call scheduled” (Neutral)
“Analyst maintains hold” (Neutral)
“Apple event announced” (Neutral).

Varied inputs cover wins/losses, macro shocks, filings, and neutral events to test edge cases and vocabulary.

#	Input	Expected Output	Actual Output
1	AAPL beats earnings and raises guidance	Positive	Positive
2	Tesla delivers a record number of vehicles in Q3	Positive	Positive
3	NVDA revenue soars; demand remains strong	Positive	Positive
4	Microsoft announces dividend increase	Positive	Positive
5	Amazon posts surprise profit this quarter	Positive	Positive
6	Meta stock jumps after strong ad sales	Positive	Positive
7	Apple unveils new chips with better margins	Positive	Positive
8	AAPL cuts iPhone production on weak demand	Negative	Negative
9	TSLA recalls vehicles over safety issue	Negative	Positive
10	NVDA faces new export restrictions	Negative	Negative
11	Fed hints at another rate hike	Negative	Negative
12	Snap misses	Negative	Negative

	revenue expectations		
13	Intel warns of weak PC demand	Negative	Negative
14	Lucid plans share offering to raise cash	Negative	Positive
15	AAPL hosts its annual developer conference	Neutral	Neutral
16	TSLA files 10-Q with the SEC	Neutral	Neutral
17	Market opens mixed on Tuesday	Neutral	Neutral
18	Company schedules earnings call for next week	Neutral	Neutral
19	Analyst maintains hold rating on AAPL	Neutral	Neutral
20	Apple product event announced for next week	Neutral	Neutral

Table 1. Experimental Results: Sentiment Classification Accuracy (n=20)

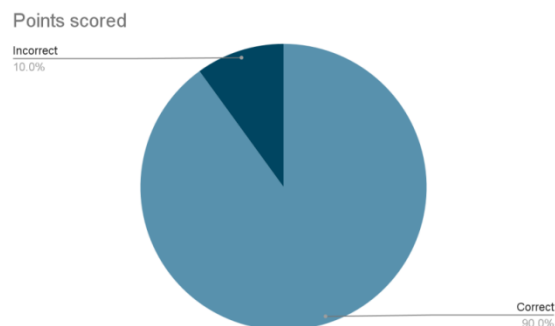


Figure 8. Distribution of Classification Accuracy in Validation Set

I tested 20 finance-style headlines, and the system labeled 18 correctly and 2 incorrectly, for an overall accuracy of 90 percent. Both mistakes shared a pattern: they were negative finance events that the model underweighted— “TSLA recalls vehicles over safety issue” and “Lucid plans share offering to raise cash.” In everyday language, terms like recall, safety issues, share offering, or dilution don’t carry strong negative polarity, while words such as raise or cash can skew scores positive, so a general-purpose lexicon like VADER drifts toward Neutral or Positive. That explains why clearly positive and neutral items were handled well, while subtler finance negatives were harder. Likely causes include the generic vocabulary, limited handling of multi-word finance cues, and thresholds tuned for general text. To mitigate this, I added a small finance lexicon boost for finance keywords and slightly widened label thresholds (± 0.10), which target these failure modes and should reduce false positives.

5. RELATED WORK

They build daily public mood from Twitter using OpinionFinder (valence) and GPOMS (six moods), validate on 2008 events, then show Calm and other dimensions Granger-cause and improve DJIA direction with a SOFNN (up to ~87% accuracy at 3–4-day lags) [11]. Compared to our app, their targets macro index prediction from Twitter; ours computes per-ticker Reddit sentiment (VADER + finance lexicon), 14-day aggregation, and live mobile visualization. We add API endpoints, history logging, and lightweight thresholds for real-time UX. Ours is better for speed, transparency, and actionable ticker-level signals.

They collect ~7M r/WallStreetBets posts (GME/AMC, 2021-01-04→03-31) and compare TextBlob, Financial-RoBERTa, and a ChatGPT-labeled, fine-tuned RoBERTa against comment volume and Google Trends using correlation and Granger causality on stationary returns [12]. Findings: sentiment is weak; volume/Trends show stronger, often bidirectional, links with price. Compared to our app (per-ticker Reddit VADER + finance lexicon + 14-day chart), theirs is heavier and meme-focused. We add real-time UX and can incorporate their strongest ideas: attention features, stationary/Granger tests, and an emoji feature—keeping our pipeline fast and interpretable.

They propose EnhancedFinSentiBERT, a transformer that injects financial dictionary knowledge and explicitly models neutral expressions [13]. On FinancialPhraseBank and FiQA, it outperforms mainstream baselines; ablations confirm both modules drive gains, especially for neutrality—historically hard in finance text. Compared with our app’s VADER + finance mini-lexicon and 14-day Reddit aggregation, their method is heavier but stronger on domain nuance and neutral handling. We contribute real-time, per-ticker UX with transparent scores; they contribute higher semantic precision. Why ours is better for our goals: latency, cost, and interpretability, with a path to add their neutral-aware transformer as an optional cloud re-scorer.

6. CONCLUSIONS

Current limitations fall into data, modeling, and reliability. On data, I ingest headlines and Reddit only, mostly English, so coverage and nuance are limited; brigading, bots, and ticker ambiguity can leak noise despite deduping and engagement weighting. On modeling, the scorer is VADER with a small finance lexicon—fast and transparent but still weak on sarcasm, negation (“misses despite revenue up”), and domain phrases (dilution, guidance cuts) without continual tuning [14]. Reliability-wise, external APIs can rate-limit or go down; my evaluation set is small; the 14-day window and thresholds are fixed; and backtesting against price reactions is minimal.

If given more time, I would add a swappable FinBERT (served via a lightweight endpoint), expand sources (SEC 8-K/10-Q snippets, earnings call summaries), improve ticker/entity

disambiguation, and add bot/brigade detection [15]. I'd build a richer "Why this score?" panel with per-phrase contributions, confidence, and source credibility weights; add watchlists, alerts, and sector heatmaps; and grow the benchmark set to hundreds of labeled headlines for periodic regression tests.

For future expansion, I want portfolio-aware signals, cross-ticker comparisons, intraday granularity, multilingual feeds, and proper backtests tied to event windows. If starting over, I'd define strict data contracts first, build the evaluation suite before UI, abstract all third-party APIs behind interfaces with fallbacks, containerize the backend, and design the sentiment layer as a pluggable service from day one.

REFERENCES

- [1] Tenney, Ian, Dipanjan Das, and Ellie Pavlick. "BERT rediscovers the classical NLP pipeline." Proceedings of the 57th annual meeting of the association for computational linguistics. 2019.
- [2] Tashildar, Aakanksha, et al. "Application development using flutter." International Research Journal of Modernization in Engineering Technology and Science 2.8 (2020): 1262-1266.
- [3] Boczkowski, Pablo, Eugenia Mitchelstein, and Mora Matassi. "Incidental news: How young people consume news on social media." (2017).
- [4] Reilly, Frank K., and Kenneth Hatfield. "Investor experience with new stock issues." Financial Analysts Journal 25.5 (1969): 73-80.
- [5] Amatriain, Xavier, et al. "Transformer models: an introduction and catalog." arXiv preprint arXiv:2302.07730 (2023).
- [6] Du, Kelvin, et al. "Financial sentiment analysis: Techniques and applications." ACM Computing Surveys 56.9 (2024): 1-42.
- [7] Moroney, Laurence. "Using authentication in firebase." The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform. Berkeley, CA: Apress, 2017. 25-50.
- [8] Solangi, Yasir Ali, et al. "Review on natural language processing (NLP) and its toolkits for opinion mining and sentiment analysis." 2018 IEEE 5th international conference on engineering technologies and applied sciences (ICETAS). IEEE, 2018.
- [9] Kesavan, Ram, et al. "Firestore: The nosql serverless database for the application developer." 2023 IEEE 39th international conference on data engineering (ICDE). IEEE, 2023.
- [10] Pradnyavant, Ajit, et al. "Quick labor service on-demand using flutter & firestore." INTERNATIONAL CONFERENCE ON INNOVATIONS IN COMPUTER SCIENCE, ELECTRONICS & ELECTRICAL ENGINEERING-2022. Vol. 2717. No. 1. AIP Publishing LLC, 2023.
- [11] Bollen, Johan, Huina Mao, and Xiaojun Zeng. "Twitter mood predicts the stock market." Journal of computational science 2.1 (2011): 1-8.
- [12] Kmak, Mateusz, et al. "Predicting stock prices with ChatGPT-annotated Reddit sentiment: Hype or reality?" International Conference on Computational Science. Cham: Springer Nature Switzerland, 2025.
- [13] Sun, Yongyong, Haiping Yuan, and Fei Xu. "Financial sentiment analysis for pre-trained language models incorporating dictionary knowledge and neutral features." Natural Language Processing Journal 11 (2025): 100148.
- [14] Sohangir, Sahar, Nicholas Petty, and Dingding Wang. "Financial sentiment lexicon analysis." 2018 IEEE 12th international conference on semantic computing (ICSC). IEEE, 2018.
- [15] Ferrara, Emilio. "Social bot detection in the age of ChatGPT: Challenges and opportunities." First Monday (2023).