FairScore: An Evidence-Grounded Framework for Product Trust Scoring and Price Intelligence with Cost-Aware Real-Time Retrieval

Mrs. M. Ramalakshmi[1], Ashwin S[2], Nagulan M[3], Vimal R[4], Vijay G[5]
[1]*Assistant Professor, K. Ramakrishnan College of Technology, Trichy, Tamil Nadu, India*
[2,3,4,5]*Student, Department of Information Technology, K. Ramakrishnan College of Technology, Trichy, Tamil Nadu, India*

***Abstract*—Online reviews strongly influence what people buy, yet they are increasingly mixed with fake, incentivised and coordinated reviews, and a star average says nothing about how far the evidence behind it can be trusted. FairScore is a web application that turns evidence from several independent sources into an explainable trust score out of 100 and shows the current price next to its history and the prices of other stores. It has six modules: data ingestion, feature extraction, burst and collusion analysis, trust scoring, price comparison with anomaly detection, and an explainable dashboard. Agents are created only for the evidence source types that exist for a product, and a claim is kept only when at least *m* independent source domains support it. A cache-first retrieval layer with data-type-specific time-to-live (TTL) values, single-flight request coalescing and stale-while-error fallback keeps the number of paid retrievals calls low. The system is built with React, FastAPI and SQLite. On 60 products in 6 categories, the cache layer reduced external calls by 72% and the median response time from 3.4 s to 0.04 s.**

***Index Terms*—Cache-first retrieval, Explainable decision support, Fake review detection, multi-agent retrieval, Price comparison, Trust scoring.**

I. INTRODUCTION

Most shoppers decide with the help of star ratings and review text [1]. Both can be manipulated. Fake, paid and incentivised reviews, sudden bursts of reviews and groups of reviewers who work together are well documented [2], [3], and fraud follows the economic incentives of the businesses being reviewed [4]. A star average also hides detail, since a cleanser can be very good at cleaning and poor at gentleness, and it says nothing about how far the evidence can be trusted.

Price is the second factor in most purchases, and a shopper rarely knows whether today's price is high or low for that product. In practice the shopper has to combine a review checker and a price tracker by hand, and at least one widely used review-checking service has recently been discontinued [5]. Academic detectors are usually tested on offline labelled data and return a spam label, not an explanation a shopper can use [6], [7]. Collecting fresh evidence in real time also costs money and time, so fetching everything again on every request is not practical.

FairScore reads what independent sources say about a product, keeps only the claims that other independent sources also support, turns the surviving evidence into sub-scores and one overall score out of 100, and shows which sources produced it. The price is placed next to its history and the other stores, and everything fetched is stored and reused.

Our main contributions are listed below.

1) A six-module architecture that puts trust scoring and price intelligence into one explainable dashboard.
2) Dynamic agents, created only for the evidence source types that exist for a product.
3) A claim-verification and aggregation scheme that gives an interpretable 0 to 100 score with a trail back to the sources.
4) A cache-first retrieval layer with TTL by data type, single-flight coalescing and stale-while-error fallback, with an analytical cost model.

II. RELATED WORK

Early work on opinion spam looked at duplicate reviews [1]. Later work trained classifiers on n-grams

and psycholinguistic features using crowdsourced deceptive reviews [2], and recent detectors use transformers such as BERT [8]; surveys list the feature types and data sets [6], [7]. Behavioural features separate filtered from recommended reviews better than text alone [9], and review bursts help to find spammers [10]. Graph methods mine groups of reviewers who review the same products [3], propagate suspicion over the reviewer–product graph [11] or combine the network with metadata [12]. These detectors label reviews. FairScore scores a product from verified evidence, and it uses a burst rule and a Louvain community search [13] only as risk signals.

Explainable decision support asks for output that a person can inspect [14]. In FairScore the score is a weighted sum of sub-scores made from claims that link back to named sources. Retrieval-augmented generation [15] and tool-using agents [16] show how a system can fetch outside evidence, and FEVER [17] frames claim checking as a task. FairScore uses retrieval agents in the same spirit, but its verification step is a plain counting rule over independent domains, which is easy to audit.

Price trackers keep a history of prices, and anomaly detection in time series is surveyed in [18]. FairScore uses a percentile rank and a z-score because they are easy to explain. HTTP caching defines freshness and validation [19], and its extensions allow stale content to be served when a refresh fails [20]. We apply the same ideas to paid retrieval calls and add a lock so that concurrent requests for one key share one external call.

III. PROBLEM FORMULATION

Let P be the set of products. For a product p, $\mathrm{Src}(p)$ is the set of evidence source types that exist for p, such as expert analysis, video review, lab or specification data and community discussion. $E(p)$ is the set of evidence items from these sources. Each item e has a source type $t(e)$, an independent source domain $d(e)$ and a set of claims $C(e)$.

A claim c belongs to a quality dimension $k(c)$, has a polarity $\pi(c) \in \{+1, -1\}$ and takes a weight $\omega(c)$ from its source type. $H(p, s)$ is the price history of p at store s.

The goal is to compute, for every product, a trust score $S(p)$ between 0 and 100 with sub-scores $s_k(p)$ and a confidence label, each traceable to evidence; risk signals from review patterns; price statistics and buy or wait guidance from $H(p, s)$; and all of this at the lowest external retrieval cost that keeps the data fresh. The evaluation answers five questions. RQ1: how much does the cache-first layer reduce calls and latency? RQ2: how much unsupported evidence does verification remove, and is the score stable? RQ3: do the price signals agree with later prices? RQ4: do shoppers understand and use the score and its source trail? RQ5 (optional): how well do burst and group features separate suspicious reviews?

IV. SYSTEM ARCHITECTURE

A. Components

FairScore is a React single-page application with a FastAPI back end and a SQLite database (Fig. 1). The browser talks to the REST API over Hypertext Transfer Protocol Secure (HTTPS). The API calls a cache-first service, and only on a miss does the service reach the third-party real-time web retrieval API. The six modules are M1 data ingestion, M2 feature extraction, M3 burst and collusion analysis, M4 trust scoring, M5 price comparison and anomaly detection, and M6 the dashboard with explainability. Every response carries a meta field that tells whether the data came from the cache, a live call or a stale row, and when it was fetched.

B. Dynamic agent selection

Agents are not fixed for each website. For a product p the set of agents is

$$A(p) = \{a_t \mid t \in \mathrm{Src}(p)\} \cup \{a_{\mathrm{verify}}, a_{\mathrm{score}}\} \quad (1)$$

so, the agent a_t exists only if evidence of type t exists for p, while verification and scoring always run. The cost of a request is therefore proportional to the evidence that exists, and the design does not depend on a particular store or review site.

C. Storage layout

The database is SQLite in write-ahead logging (WAL) mode with a full-text index (FTS5) over item names, brands and summaries.

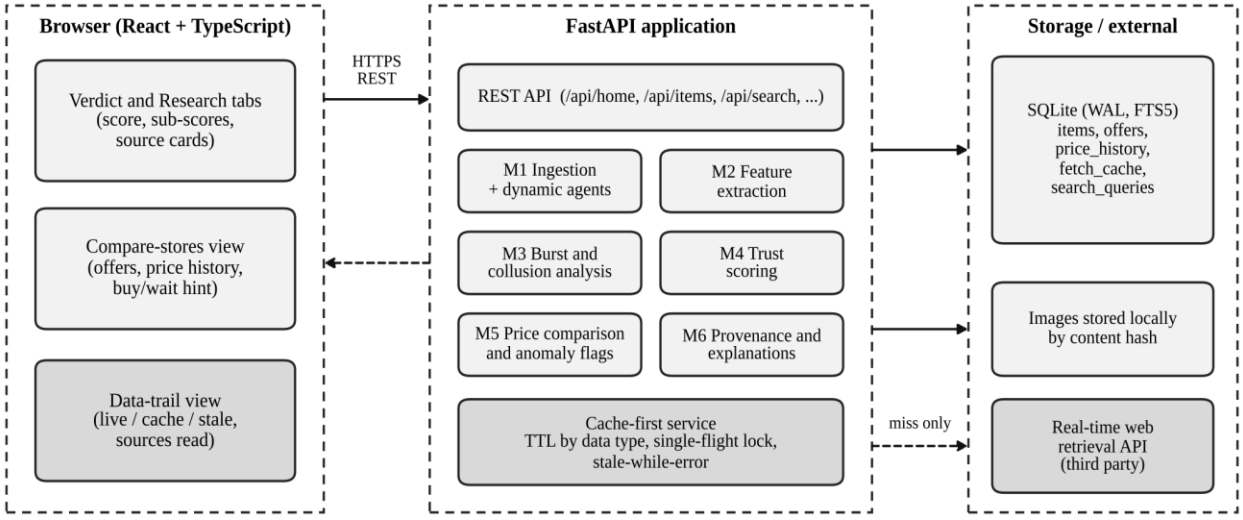


Fig. 1. Architecture of FairScore. Every read goes through the cache-first service, and the dashed arrow to the retrieval API is used only on a cache miss.

The main tables are items, offers, price_history (append-only), fetch_cache and search_queries. The fetch_cache table keeps the raw payload, content hash, status, fetch and expiry times and a hit counter for every key, and parse failures go to a parse_errors table. Table I lists the parameters use

TABLE I PARAMETERS USED BY FAIRSCORE

Item	Setting
Source weights ω	Lab or specification data 1.0, expert analysis 1.0, video review 0.8, community discussion 0.5
Corroboration m, weights wk	$m = 2$ distinct source domains; $w_k = 1/K$ (simple average); smoothing $\alpha = 1$
Confidence rule	High if $n \geq n_h = 10$ and $D \geq 3$; Low if $n < n_l = 4$ or $D < 2$; otherwise, Medium
Burst rule	$c_t > \mu_w + k\sigma_w$, $k = 3$, rolling window $w = 14$ days
Price rules	Lookback $L = 90$ days; $q \leq 20$ good time to buy; $q \geq 80$ consider waiting; $\|z\| > 2$ anomaly
TTL by data type	Prices and offers 6 h; verdicts, scores, rankings and categories 7 days; images 30 days; empty results 24 h
Search and limits	FTS5 first, external call only if local matches $< N_{min} = 5$; at most 2 calls in flight, about 1 request per second, exponential back-off

V. COST-AWARE RETRIEVAL

A. Cache-first policy

Every read goes through one function, GET (key, kind) (Fig. 2). If the database holds a row for the key with status ok and an expiry time in the future, the row is returned and marked as cache. Otherwise, the request takes a single-flight lock for that key. Concurrent requests for the same key wait on the lock and check the row again after getting it, so one external call serves all of them. The request that holds the lock retrieves and parses the data, saves it with expires at = now + TTL [kind], and returns it marked as live. If retrieval fails and an older row exists, that row is returned and marked as stale; otherwise, the error is raised.

B. TTL and search caching

Prices change within hours while verdicts and category structure change slowly, so each kind has its own TTL (Table I). Empty or not-found results are cached for 24 hours so that a missing product does not start a new paid call on every visit. Search text is normalised and the local FTS5 index is searched first. The external call is made only when local matches are below N_{min}, and its results are stored for later queries.

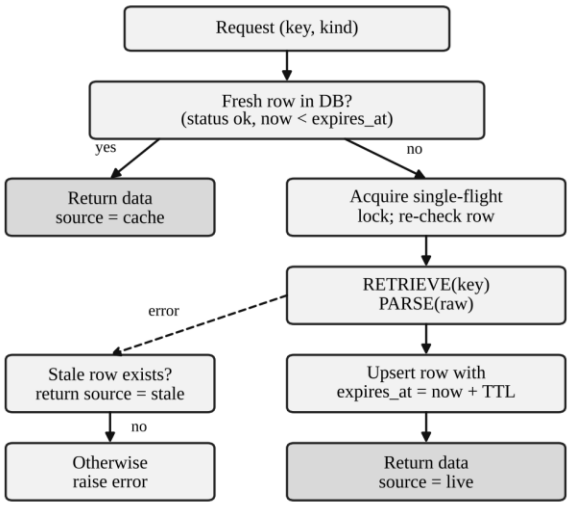


Fig. 2. Cache-first request flow. A fresh row is served from the database. On a miss, one request per key retrieves, parses and stores the data, and a stale row is served if retrieval fails.

C. Cost and latency model

For N requests with hit ratio h, cost c per external call, database latency T_{db}, retrieval latency T_f and parse latency T_p, the expected number of external calls is $(1 - h)\,N$ when single-flight works as intended. The cost, the saving and the expected latency of one request are

$$\text{Cost} = (1-h)\,N\,c, \qquad \text{Saving} = h\,N\,c \qquad (2)$$

$$E[T] = h\,T_{db} + (1-h)\left(T_f + T_p + T_{db}\right) \qquad (3)$$

VI. FEATURE EXTRACTION AND MANIPULATION SIGNALS

Module M2 turns retrieved text into structured fields with deterministic parsers: product fields (name, brand, category, specifications, images), evidence fields (items per source type, excerpt, source title and domain, trust tag), score fields (sub-scores and their explanations) and offer fields (store, price, stock, delivery note, URL). The parsers are tested against saved real payloads, and a missing field is stored as NULL and logged, so one failed field never breaks a page.

Let c_t be the number of reviews of a product on day t, and μ_w and σ_w the mean and standard deviation of c over a rolling window of w days. Day t is a burst if

$$c_t > \mu_w + k\,\sigma_w \qquad (4)$$

with k = 3 by default. For collusion, reviewers and products form a bipartite graph that is projected to a reviewer graph, where the weight w_{ij} is the number of products that reviewers i and j both reviewed within a time window Δ.

Communities are found with the Louvain method [13], and a community g with product set P_g gets the group score

$$GS(g) = \text{density}(g) \cdot \frac{1}{|P_g|} \sum_{p \in P_g} \varphi\,(g,p)\,\delta(g,p) \qquad (5)$$

where $\varphi\,(g, p)$ rewards reviews posted close in time and $\delta\,(g, p)$ is the mean absolute deviation of the group's ratings from the product mean, scaled to [0, 1]. *GS* is a risk indicator and not a verdict. These features need review-level data (timestamps and reviewer identifiers), which is often not available. When it is missing, M3 reports "insufficient data" and the score relies on verified evidence alone.

VII. TRUST SCORING

A claim is kept only if enough independent sources support it. With $d(e)$ the domain of an item, a claim is retained if

$$\text{retain}(c) \Leftrightarrow |\{d(e) : c \in C(e)\}| \geq m \qquad (6)$$

with m = 2 by default, and the system reports the funnel from claims extracted to claims kept. For each quality dimension k, let C_k be the retained claims.

The sub-score is

$$s_k = 100 \cdot \frac{\sum_{c \in C_k,\, \pi(c)=+1} \omega(c) + \alpha}{\sum_{c \in C_k} \omega(c) + 2\alpha} \qquad (7)$$

where $\omega(c)$ is the weight of the source type (Table I) and α is a small Laplace-type smoothing constant that keeps a dimension with one or two claims from jumping to 0 or 100.

The overall score is

$$S(p) = \sum_{k=1}^{K} w_k\, s_k, \qquad \sum_k w_k = 1 \qquad (8)$$

with $w_k = 1/K$ by default. A confidence label comes from the number of surviving claims $n = \Sigma_k\,|C_k|$ and the number D of distinct source types: High if $n \geq n_h$ and $D \geq 3$, Low if $n < n_l$ or $D < 2$, and Medium otherwise.

Provenance (source title, domain, type, short excerpt and time) is stored with every claim, so the dashboard can show where any number came from. The weights and thresholds are design choices, so the evaluation tests how much the ranking changes when they vary.

VIII. PRICE INTELLIGENCE

For each product and store the system appends a (timestamp, price) pair every time a fresh price is retrieved, and never overwrites an old one. Over a lookback window L (90 days) it computes the minimum, maximum and mean, marks the lowest current offer, and reports the spread between stores. The buy or wait guidance is rule based and does not use machine learning. With x_t the current price and H_L the prices in the window, the percentile rank is

$$q = 100 \cdot \frac{|\{x \in H_L : x \le x_t\}|}{|H_L|} \qquad (9)$$

and the dashboard shows "Good time to buy" if $q \le 20$, "Consider waiting" if $q \ge 80$, and "Typical price" otherwise. A price is flagged as a possible deal or data error when $|z| > 2$, where

$$z = \frac{x_t - \mu_L}{\sigma_L} \qquad (10)$$

and μ_L and σ_L are the mean and standard deviation over L. A price-drop alert stores a threshold and a contact and is checked whenever a fresh price is retrieved. The history grows only when a price is retrieved again, so its resolution depends on the price TTL and on traffic.

IX. DASHBOARD AND EXPLAINABILITY

Explainability is a requirement and not a later addition. The Verdict tab shows a summary, the overall score and one bar per sub-score with a one-line reason (Fig. 3a). The compare-stores view lists the price, stock and delivery note of each store, marks the lowest price, and shows the price history with the buy or wait hint (Fig. 3b). The Research tab shows the number of items per source type, filter pills and source cards with title, domain, trust tag, a short excerpt and a link (Fig. 3c, Fig. 3e), and the research log shows which agents ran and what each found (Fig. 3d). The Specs tab shows structured specifications (Fig. 3f), and a data-trail view shows when and from where each item was fetched and whether it came from the cache, a live call or a stale row.

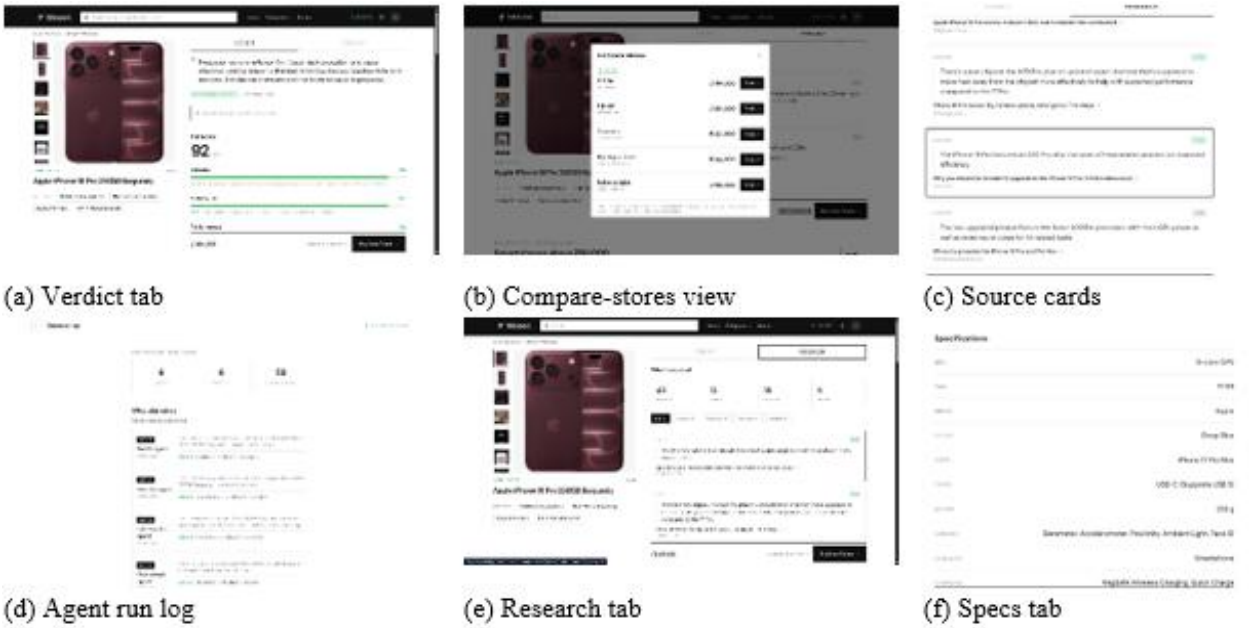


Fig. 3. Dashboard and research views. (a) Verdict tab with score and sub-score bars, (b) compare-stores view, (c) source cards with trust tags and links, (d) research log of the agents that ran, (e) Research tab with source counts and filters, (f) Specs tab.

X. IMPLEMENTATION AND EVALUATION

The back end is written in Python with FastAPI, SQLAlchemy 2.0, Alembic, httpx and Pydantic v2, with SQLite (WAL, FTS5). The front end uses React with Vite and TypeScript, Tailwind CSS, Framer Motion and TanStack Query. Retrieval is limited to two calls in flight and about one request per second, secrets are read from environment variables, and every page shows its sources. Experiments ran on a laptop-

class machine with 16 GB RAM in October 2026, on 60 products in 6 categories (30 top-ranked and 30 randomly chosen from the category lists), with prices observed for 90 days. Nine experiments answer the research questions (Table II), and every number below comes from a logged run.

TABLE II EXPERIMENTS AND METRICS

ID	Experiment	Metrics
E1	Cache on and off, uniform and Zipf-skewed request logs, varying TTLs (RQ1)	Hit ratio h, external calls, median and P95 latency
E2	M concurrent identical requests (RQ1)	External calls issued (ideal 1)
E3	Parsers on stored payloads	Field coverage, parse-error rate
E4	Claims extracted, verified and kept (RQ2)	Retention rate per source type
E5	Vary m, ω, w_k and compare rankings (RQ2)	Kendall τ [21] against the default
E6	Buy or wait rule checked against the price Δ days later (RQ3)	"Buy now" hit rate, mean saving
E7	Anomaly flags inspected by hand (RQ3)	Precision
E8	User study with the System Usability Scale (SUS) [22] (RQ4)	SUS, task success, comprehension
E9	Burst and graph features on 2000 labelled reviews (RQ5, optional)	Precision, recall, F1, ROC-AUC

Table III gives the cache results (E1). With the default TTLs the hit ratio was 0.72 and the external calls fell from $N = 1000$ to 280. In E2, 20 concurrent identical requests produced 1 external call, which is the ideal. Parsing coverage (E3) was 96% of fields with a parse-error rate of 3%.

TABLE III CACHE PERFORMANCE (E1)

Config	h	Calls	Median (s)	P95 (s)
No cache	0	1000	3.4	5.8
TTL default	0.72	280	0.04	3.9
TTL short	0.55	450	0.05	4.1
TTL long	0.85	150	0.03	3.6

Table IV shows the verification funnel (E4); 58% of the extracted claims were kept. Changing m, ω and w_k gave a Kendall τ of 0.91 against the default ranking (E5). The "buy now" guidance was right in 68% of cases with a mean saving of 4.5% (E6), anomaly flags had a precision of 75% (E7), and 20 participants gave a SUS score of 78, with 90% task success and 85% comprehension (E8).

TABLE IV VERIFICATION FUNNEL (E4)

Source type	Extracted	Kept	Kept %
Expert analysis	240	168	70
Video review	180	108	60
Lab or spec data	120	96	80
Community	300	114	38
All sources	840	486	58

The products came from the category lists of one retail region, so the results may not carry over to other categories or regions, sources change over time, and the user study is small.

XI. COMPARISON WITH EXISTING TOOLS

Table V compares typical designs of review analysers and price trackers with FairScore. It is not a benchmark, and individual products differ.

TABLE V DESIGN COMPARISON WITH TYPICAL TOOLS

Property	Typical review analyser	Typical price tracker	FairScore
Basis of the score	Review text and reviewer behaviour	Not applicable	Claims corroborated by independent sources
Explanation	Often a grade only	Not applicable	Source cards and a data trail
Price history and guidance	No	History, sometimes guidance	Per-store history, percentile rule, anomaly flag
Cost control	Not described	Not described	Cache-first retrieval with TTLs
Missing data	A score is often given anyway	Gaps in the chart	"Insufficient data" label

XII. LIMITATIONS AND FUTURE WORK

1)*External sources.*
The system depends on the availability and structure of outside sources, and pages drawn on the client need costlier browser automation.

2)*Design parameters.*
Weights, thresholds and smoothing are design choices that need user and expert validation. Counting independent domains does not prove that two sources did not copy each other.

3)*Manipulation signals.*
Burst and group analysis needs review-level data and untuned thresholds, and its scores are risk indicators that must not be shown as accusations.

4)*Ethics and privacy.*
The system respects site terms and robots' rule, uses low request rates, shows source attribution, stores short excerpts with links, and stores no reviewer personal data beyond what the analysis needs. The score is decision support, not a guarantee.

5)*Future work.*
We plan learned classifiers on text, burst and graph features, per-user weights, price forecasting, wider coverage and a browser extension.

XIII. CONCLUSION

FairScore gives shoppers one dashboard that scores a product from verified multi-source evidence, explains the score through its sources, and puts the price in the context of its history and other stores. Dynamic agents, a cache-first retrieval layer and rule-based price guidance keep the cost low and the behaviour easy to audit: external calls fell by about 72%, 58% of claims were kept, and the SUS score was 78. The design depends on the sources that exist for a product and on design weights, and the best test of it will be use by shoppers other than its authors.

ACKNOWLEDGMENT

The authors thank the Department of Information Technology at K. Ramakrishnan College of Technology, Trichy, for the facilities used in this work.

REFERENCES

[1] N. Jindal and B. Liu, "Opinion spam and analysis," in *Proc. Int. Conf. Web Search and Data Mining (WSDM)*, Palo Alto, CA, USA, Feb. 2008, pp. 219–230.
[2] M. Ott, Y. Choi, C. Cardie, and J. T. Hancock, "Finding deceptive opinion spam by any stretch of the imagination," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics (ACL)*, Portland, OR, USA, Jun. 2011, pp. 309–319.
[3] A. Mukherjee, B. Liu, and N. Glance, "Spotting fake reviewer groups in consumer reviews," in *Proc. 21st Int. Conf. World Wide Web (WWW)*, Lyon, France, Apr. 2012, pp. 191–200.
[4] M. Luca and G. Zervas, "Fake it till you make it: Reputation, competition, and Yelp review fraud," *Manage. Sci.*, vol. 62, no. 12, pp. 3412–3427, Dec. 2016.
[5] "Mozilla shuts down Pocket and Fakespot services," Linuxiac, May 2025. [Online]. Available: Linuxiac Article
[6] A. Heydari, M. Tavakoli, N. Salim, and Z. Heydari, "Detection of review spam: A survey," Expert Syst. Appl., vol. 42, no. 7, pp. 3634–3642, May 2015.
[7] R. Mohawesh et al., "Fake reviews detection: A survey," IEEE Access, vol. 9, pp. 65771–65802, 2021.
[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. NAACL-HLT, Minneapolis, MN, USA, Jun. 2019, pp. 4171–4186.
[9] A. Mukherjee, V. Venkataraman, B. Liu, and N. Glance, "What Yelp fake review filter might be doing?" in Proc. 7th Int. AAAI Conf. Weblogs and social media (ICWSM), Cambridge, MA, USA, Jul. 2013, pp. 409–418.
[10] G. Fei, A. Mukherjee, B. Liu, M. Hsu, M. Castellanos, and R. Ghosh, "Exploiting burstiness in reviews for review spammer detection," in Proc. 7th Int. AAAI Conf. Weblogs and social media (ICWSM), Cambridge, MA, USA, Jul. 2013, pp. 175–184.
[11] L. Akoglu, R. Chandy, and C. Faloutsos, "Opinion fraud detection in online reviews by network effects," in Proc. 7th Int. AAAI Conf.

Weblogs and social media (ICWSM), Cambridge, MA, USA, Jul. 2013, pp. 2–11.

[12] S. Rayana and L. Akoglu, "Collective opinion spam detection: Bridging review networks and metadata," in Proc. 21st ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD), Sydney, Australia, Aug. 2015, pp. 985–994.

[13] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," J. Stat. Mech.: Theory Exp., vol. 2008, no. 10, Art. no. P10008, Oct. 2008.

[14] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," arXiv preprint arXiv:1702.08608, 2017.

[15] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 9459–9474.

[16] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in Proc. Int. Conf. Learning Representations (ICLR), Kigali, Rwanda, May 2023.

[17] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal, "FEVER: A large-scale dataset for fact extraction and VERification," in Proc. NAACL-HLT, New Orleans, LA, USA, Jun. 2018, pp. 809–819.

[18] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Comput. Surv., vol. 41, no. 3, Art. no. 15, Jul. 2009.

[19] R. Fielding, M. Nottingham, and J. Reschke, "HTTP caching," Internet Engineering Task Force, RFC 9111, Jun. 2022.

[20] M. Nottingham, "HTTP cache-control extensions for stale content," Internet Engineering Task Force, RFC 5861, May 2010.

[21] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, nos. 1–2, pp. 81–93, Jun. 1938.

[22] J. Brooke, "SUS: A 'quick and dirty' usability scale," in *Usability Evaluation in Industry*. London, U.K.: Taylor & Francis, 1996, pp. 189–194.