

ATL – AI THREAT LENS: AN INTELLIGENT REAL-TIME CYBER THREAT DETECTION PLATFORM USING MACHINE LEARNING AND MULTI-ENGINE THREAT INTELLIGENCE

Prem kumar P ¹, Raghul Krishna J.V ², Reenish Ruban JP ³, Sheeba C ⁴

¹⁻³ Students, Department of Information Technology, Meenakshi College of Engineering, Tamil Nadu, India.

⁴ professor, Department of Information Technology, Meenakshi College of Engineering, Tamil Nadu, India.

Abstract – ATL – AI Threat Lens is an AI-assisted cybersecurity web application developed to detect malicious URLs, suspicious downloadable files, APK packages, and phishing indicators through a hybrid architecture that combines machine learning, third-party threat intelligence APIs, and a responsive web dashboard. The system integrates a React and TypeScript frontend, an Express.js middleware backend, and a Python-based machine learning engine. The platform analyses URLs using a trained ensemble phishing detection model comprising a Gradient Boosting classifier and a Random Forest classifier, and verifies files using multi-engine antivirus services including VirusTotal. The system produces a normalized composite threat score, severity classification across four risk levels, and engine-level evidence for improved user decision-making. By presenting technical threat intelligence in a simplified visual form through a live terminal interface and color-coded risk indicators, the application enables both technical and non-technical users to identify digital threats before interacting with unsafe resources. The platform is designed as a modular security solution that can be extended for enterprise, academic, and personal use. In addition to basic scanning, the system integrates five external APIs — VirusTotal, Google Safe Browsing, AbuseIPDB, IPQualityScore, and URLScan.io — to cross-validate machine learning predictions with globally sourced threat intelligence. The composite danger score is computed through a weighted aggregation of all available signal sources, offering a transparent and explainable threat assessment that transforms complex cybersecurity indicators into an actionable classification result.

Keywords: Phishing Detection, Machine Learning, URL Analysis, Threat Intelligence, VirusTotal, Cybersecurity, Ensemble Classifier, Server-Sent Events, Feature Extraction, Malware Classification.

1. INTRODUCTION

The rapid expansion of internet services has significantly increased the number and complexity of cyber threats encountered by users in daily digital activities. Malicious websites, phishing links, malware-infected files, and modified application packages are commonly employed by attackers to steal sensitive information, compromise systems, or gain unauthorized access to critical infrastructure. As online platforms become progressively more integrated into personal, academic, and business environments, the need for reliable, real-time threat detection mechanisms has become increasingly important. Cybercriminals continuously evolve their strategies to evade conventional defenses, registering new malicious domains, obfuscating malware payloads, and crafting increasingly convincing phishing pages that closely mimic legitimate services.

Conventional security tools generally focus on only one category of protection. Antivirus software primarily scans files after they have been downloaded to the user's device, while browser-based security tools often rely on predefined blacklists that are incapable of detecting newly created malicious domains or zero-day phishing pages. This creates a significant security gap wherein users must depend on multiple disconnected tools to verify whether a URL or file is safe. Such fragmentation reduces detection efficiency, increases user burden, and raises the probability of human error during security assessment. Furthermore, existing standalone tools do not provide an interpretable explanation of why a resource is considered dangerous, leaving users without actionable context.

ATL – AI Threat Lens was developed to address these limitations by providing a unified AI-driven cybersecurity platform capable of analyzing both URLs and files within a single integrated system.

The application combines machine learning algorithms with multiple external threat intelligence APIs to perform real-time, multi-layer security analysis. Instead of simply identifying a resource as safe or dangerous, the system generates a detailed composite threat score along with a technical explanation organized by source and engine, helping users understand the precise reasoning behind each classification. The architecture is designed to be modular and extensible, supporting future integration of additional intelligence services, deep learning models, and enterprise-grade deployment configurations.

The project integrates a modern frontend interface, a backend processing layer, and a dedicated machine learning engine to create a scalable and maintainable three-tier architecture. The frontend enables users to submit suspicious resources through a responsive dashboard, while the backend coordinates communication between the user interface and the analysis engine. The machine learning engine performs URL feature extraction, model inference, spam detection, and orchestration of all external API queries. The final output is returned either as a complete JSON response for file scans or as a live Server-Sent Events stream for URL scans, enabling real-time visibility into the analysis pipeline through a built-in terminal interface.

2. Body of Paper

2.1 SYSTEM OVERVIEW

ATL – AI Threat Lens is an intelligent cybersecurity platform developed to identify malicious digital resources such as phishing URLs, suspicious files, and modified application packages before they can affect the user's system. The system combines machine learning techniques with external threat intelligence services to perform multi-layered security analysis in real time. Unlike traditional security tools that focus only on file scanning or blacklist-based website detection, the proposed system provides a unified environment where different types of cyber threats can be examined within a single web-based interface. The application calculates a dynamic composite danger score from 0 to 100, classifies the severity of the threat into four distinct levels (SAFE, WARNING, DANGER, CRITICAL), and presents the result in a user-friendly visual format so that users can make informed decisions before accessing potentially harmful resources.

The system operates in two primary scanning modes: URL mode and File mode. In URL mode, the user submits a web address that is then processed through

eight sequential analysis steps, each executed by a dedicated module within the Python engine. These steps include URL validation, ML feature extraction, VirusTotal query, Google Safe Browsing lookup, AbuseIPDB IP reputation check, IPQualityScore analysis, URLScan.io submission, and composite threat classification. Each step emits a real-time progress event via Server-Sent Events, which the frontend renders in a live terminal interface, giving the user complete visibility into the analysis process as it occurs. In File mode, the user uploads a file of any size up to 15 gigabytes, which is streamed to disk and submitted to VirusTotal for multi-engine antivirus analysis. The resulting verdict is combined with file metadata to produce a threat classification and composite score.

2.2 Key Features

The ATL platform incorporates several features designed to enhance detection accuracy, user experience, and operational transparency. The system provides a dual-mode scanning interface that accepts both web URLs and uploaded files, eliminating the need for separate tools. Secure user authentication is implemented through a session-based login system with local storage persistence and a Remember Me option, ensuring that only authorized users can access the scanning dashboard. A dark-themed responsive interface built with React and TailwindCSS adapts to desktop and mobile viewports without functional compromise.

The live terminal interface represents one of the most distinctive features of the system. When a URL is submitted for analysis, the frontend opens a Server-Sent Events connection to the backend, which proxies the streaming output from the Python ML engine directly to the browser. Each analysis step — from ML feature extraction through to final threat classification — is displayed in real time, with color-coded status icons indicating whether a step is waiting, running, completed, or has encountered an error. This transparency mechanism transforms what is typically a black-box security scan into an observable, auditable process.

The composite danger score is displayed as a visual gauge with per-source breakdown bars, showing individually how much each analysis engine contributed to the overall threat assessment. The threat result panel provides a categorized classification (such as Phishing, Ransomware, Botnet/C2, or Legitimate), a detailed natural-language description of the threat type, engine-level evidence from all integrated APIs,

a SHA256 fingerprint for uploaded files, and a direct link to the URLScan.io report for submitted URLs. Browser push notifications are triggered automatically when a scan completes, ensuring users are informed even when the tab is not in focus. A profile management system allows users to save and update personal information, including a profile photograph, and a notification centre provides a persistent history of scan alerts.

2.3 SYSTEM ARCHITECTURE

The architecture of ATL – AI Threat Lens is designed as a three-tier model consisting of the Presentation Layer, the Application Layer, and the Intelligence Layer. Each tier is independently deployable and communicates with adjacent tiers exclusively through well-defined HTTP and REST interfaces, ensuring loose coupling, maintainability, and horizontal scalability.



Fig1: System Architecture diagram

3. LITERATURE SURVEY

The problem of automated phishing and malware detection has been extensively studied within the cybersecurity research community, with multiple approaches proposed across the spectrum of rule-based systems, machine learning classifiers, and deep learning architectures. A review of relevant literature provides the theoretical foundation upon which ATL – AI Threat Lens was designed and implemented.

Sahingoz et al. [1] conducted a comprehensive study on machine learning-based phishing URL detection and proposed a real-time system utilizing seven different classification algorithms applied to a dataset of over 73,000 URLs. Their work demonstrated that tree-based ensemble methods, particularly Random Forest, achieved superior accuracy compared to linear classifiers, establishing the empirical basis for the ensemble model architecture adopted in the proposed system. Their study also confirmed that URL-based features alone, without relying on third-party black-

lists, can achieve high detection accuracy, which motivates the feature engineering approach used in the ATL phishing model.

Mohammad et al. [2] proposed an intelligent phishing website detection system using an associative classification approach that analyzed seventeen URL and website features. Their research highlighted the importance of combining structural URL features such as domain length, subdomain depth, and the presence of IP addresses in the URL with behavioral features such as redirect counts and HTTPS usage. This multi-dimensional feature perspective directly informed the design of the fifty-feature extraction module implemented in the ATL system.

Gutierrez et al. [3] examined the integration of threat intelligence feeds from multiple external sources to improve the reliability of malware detection systems. Their work demonstrated that cross-validation of ML predictions with reputation-based APIs significantly reduced false positive rates and improved detection of zero-day threats. This finding provided the rationale for the multi-API integration strategy in ATL, where machine learning predictions are systematically validated against VirusTotal, Google Safe Browsing, AbuseIPDB, IPQualityScore, and URLScan.io.

Buber et al. [4] proposed an NLP-based approach to spam and phishing URL detection that utilized text-based features such as keyword density, entropy measures, and character sequence patterns. Their research demonstrated that entropy-based features are highly discriminative in identifying algorithmically generated domain names commonly used in phishing campaigns. The ATL feature extractor incorporates Shannon entropy calculations for the full URL, domain component, and path component, directly applying this finding in the feature engineering pipeline. Rajab et al. [5] analyzed the landscape of web-based malware distribution and found that newly registered domains account for a disproportionate share of malware hosting activity. Their study underscored the value of domain age as a threat signal, which is captured in the ATL system through the IPQualityScore API's domain age field and the suspicious TLD classification feature in the ML extractor. From the reviewed literature, it is evident that a system combining ensemble machine learning with real-time threat intelligence APIs, real-time result streaming, and an explainable composite scoring mechanism represents a meaningful advance over existing single-method or single-source detection tools.

4. EXISTING SYSTEM ANALYSIS

Traditional cybersecurity scanning tools are characterised by a narrow operational scope, each addressing a specific category of threat without integration across threat types. Standalone antivirus software such as Windows Defender and Malwarebytes operates primarily through signature databases and heuristic analysis applied to files already present on the user's system. While effective against known malware families, these tools cannot assess the risk of a URL before a file is downloaded, nor can they detect phishing pages that do not contain traditional malware but instead operate through social engineering. Browser-integrated security extensions such as Google Safe Browsing and Microsoft SmartScreen provide blacklist-based URL filtering but are constrained by the timeliness of their blacklist updates. Newly registered phishing domains that have not yet accumulated sufficient abuse reports may evade detection for hours or days after going live, creating a critical window of vulnerability. Furthermore, these tools do not provide users with any explanation of why a site has been blocked, reducing user trust and increasing the likelihood of users overriding the block.

VirusTotal, while providing multi-engine analysis for both URLs and files, is a raw intelligence aggregator and not a decision-support system. It presents raw engine verdicts without normalization, composite scoring, or natural-language threat categorization. Non-technical users may find it difficult to interpret the results and determine an appropriate course of action. There is no ML layer to assess URLs that have not yet been indexed by any of the antivirus engines, meaning newly created phishing pages may show as undetected.

The primary limitations of existing systems include the absence of a unified scanning interface for both URLs and files, the lack of machine learning-based zero-day detection for phishing URLs, the absence of a real-time observable analysis pipeline, the unavailability of a normalized composite score that integrates multiple intelligence sources, and the lack of natural-language threat categorization and description. The proposed ATL system directly addresses each of these limitations through its multi-layer architecture and intelligent threat classification framework.

5. PROPOSED SYSTEM

The proposed ATL – AI Threat Lens system addresses the limitations of existing cybersecurity tools

by integrating artificial intelligence with a comprehensive multi-API threat intelligence infrastructure, all accessible through a unified, modern web-based interface. The system is designed around three core principles: completeness, through the analysis of both URL-based and file-based threats within a single platform; explainability, through the provision of per-engine evidence and natural-language threat descriptions; and real-time observability, through the streaming of analysis steps via Server-Sent Events to a live terminal interface.

For URL analysis, the system deploys an ensemble machine learning model trained on 48,729 real-world URLs with fifty engineered features covering URL structure, domain composition, entropy metrics, keyword patterns, and brand impersonation indicators. The ensemble combines a Gradient Boosting classifier and a Random Forest classifier using a weighted voting scheme (2:1 ratio) to produce a phishing probability score from 0 to 100. A separate spam detection pipeline operates in parallel, using a pre-trained text classification model to assess whether the URL exhibits spam-like characteristics. These ML predictions are then enriched with intelligence from five external APIs: VirusTotal provides verdict counts from over seventy antivirus engines; Google Safe Browsing flags URLs matching known malware, social engineering, and unwanted software categories; AbuseIPDB reports the abuse confidence score and total report count for the resolved IP address; IPQualityScore provides a holistic risk score along with phishing, malware, spam, and domain age signals; and URLScan.io performs screenshot capture and behavioral analysis.

For file analysis, the system streams uploaded files of up to 15 gigabytes to disk without loading them into RAM, computing a SHA256 hash for cache lookup against the VirusTotal database. If the file is already known to VirusTotal, results are returned immediately from the cache. For unknown files smaller than 32 megabytes, the file is uploaded directly to the VirusTotal standard API endpoint. For files exceeding 32 megabytes, a special large-file upload URL is first obtained from VirusTotal, enabling secure transmission of very large executables, archives, and APK packages. The resulting engine verdict counts are then passed through the threat classifier to determine risk level and category.

6.1 Presentation Layer (Frontend)

The Presentation Layer is implemented using React 18 with TypeScript, styled through TailwindCSS version 3, and bundled by Vite 8. This layer is responsible for collecting user input through form controls, dispatching scan requests to the Application Layer, and rendering analysis results through interactive visual components. The routing system is managed by React Router 6 in Single Page Application mode, defining three primary routes: the login page at /login, the main dashboard at /, and a 404 catch-all. The QueryClient from TanStack React Query manages server state caching for non-streaming API calls. The TooltipProvider from Radix UI and the dual toast providers from Sonner and the custom Toaster component provide overlay and notification infrastructure throughout the application.

The main dashboard page, defined in client/pages/Index.tsx, contains the complete scanning interface and is the largest component in the application at approximately 1,300 lines of TypeScript. It manages a comprehensive local state including the active scan tab (URL or File), the current scan target string, the array of scan step states for the live terminal, the final ScanResponse result object, the user profile data object, profile editing state, the notification centre entries, and the active modal states. The component integrates the Navbar, Footer, ATLLogo, LiveTerminal, CompositeGauge, and ThreatResultPanel sub-components to compose the complete dashboard view.

The LiveTerminal component renders an auto-scrolling monospaced terminal panel that displays real-time analysis progress during URL scans. Each scan step is represented by an entry showing a status icon (waiting circle, animated spinner for running, green checkmark for done, red cross for error), the step label, and a detail message updated as each API returns its result. The CompositeGauge component renders the overall composite danger score as an animated horizontal bar, with per-source breakdown bars for each of the five intelligence engines, each labeled with its assigned weight in the scoring formula. The ThreatResultPanel component presents the final verdict with a themed background and border color corresponding to the risk level, along with the threat category badge, threat description text, composite gauge, engine results list, and supplementary file metadata.

6.2 Application Layer (Express.js Backend)

The Application Layer is implemented using Express.js version 5.1 with TypeScript and serves as the

coordination hub between the Presentation Layer and the Intelligence Layer. It is integrated with the Vite development server through a custom dev.mjs script that launches both processes concurrently on a single port (8080), enabling a unified development experience with hot module replacement for the frontend and tsx-based live reload for the backend. For production, the server is compiled to a standalone ESM bundle (node-build.mjs) using Vite's server build configuration and can be executed with Node.js without additional dependencies.

The Express server defines four primary API endpoints. The POST /api/scan/url/stream endpoint is the primary URL analysis endpoint, implementing a Server-Sent Events proxy that opens a streaming fetch connection to the Python ML engine's /analyze/url/stream endpoint and pipes the response chunks directly to the client browser using res.write(). SSE headers including Content-Type: text/event-stream, Cache-Control: no-cache, Connection: keep-alive, and X-Accel-Buffering: no are set immediately before the proxy connection is established, ensuring the browser begins receiving events without buffering. The POST /api/scan/url endpoint provides a fallback standard JSON scan for clients that cannot use SSE. The POST /api/scan/file endpoint uses the Multer middleware configured with disk storage in the OS temporary directory to accept multipart file uploads of up to 15 gigabytes, then forwards the file to the Python engine's /analyze/file endpoint via a multipart FormData request, and deletes the temporary file from disk in the finally block regardless of scan outcome. The GET /api/ml/health endpoint proxies a health check to the Python engine.

The shared/api.ts file defines TypeScript interfaces that are consumed by both the Express backend and the React frontend, establishing type-safe data contracts for all API request and response objects. Key interfaces include ScanResponse, which encapsulates the complete scan result with fields for mode, threat_label, risk_level, category, composite, ml, virustotal, google_safebrowsing, abuseipdb, ipqualityscore, and urlscan; CompositeScore, which defines the overall score and per-source breakdown structure; and discriminated union types ThreatLevel and RiskLevel that constrain threat classification values to their valid enumerated states.

6.3 Intelligence Layer (Python ML Engine)

The Intelligence Layer is implemented as a Flask 3.0 application running on port 5001, serving as the analytical core of the system. It is launched separately

from the Node.js backend using the dev:ml script defined in package.json, and communicates with the Express backend through internal HTTP requests. The engine exposes three primary endpoints: GET /health for status monitoring, POST /analyze/url and POST /analyze/url/stream for URL analysis (standard and streaming respectively), and POST /analyze/file for file analysis. The Flask CORS middleware is configured to accept cross-origin requests, enabling direct browser-to-engine communication in development scenarios. The engine is configured to accept file uploads of up to 15 gigabytes by setting the MAX_CONTENT_LENGTH configuration attribute.

On startup, the engine loads two pre-trained scikit-learn model objects from disk using joblib: the phishing detection ensemble stored in phishing_model_v3.pkl, which contains both the Gradient Boosting (gb) and Random Forest (rf) classifiers; and the spam detection pipeline stored in spam_model.pkl. These model files are deserialized once at startup and held in memory for the lifetime of the server process, avoiding repeated disk I/O overhead. The feature_names.pkl file stores the ordered list of feature column names used during model training, ensuring that prediction DataFrames are constructed with the correct column order.

7. ALGORITHMS AND AI MODEL

7.1 Feature Extraction Module

The feature extraction pipeline, implemented in python_engine/features.py, accepts a raw URL string and returns a dictionary of fifty numerical features that collectively characterize the structural, lexical, and semantic properties of the URL. This dictionary is then converted to a single-row pandas DataFrame and passed to the ensemble classifiers for prediction. The feature engineering design draws on established research in phishing URL analysis and encompasses five broad categories of signal.

The first category covers length and count-based features, including url_len (total URL character count), domain_len, path_len, query_len, fragment_len, dot_count, hyphen_count, underscore_count, slash_count, at_count, question_count, eq_count, amp_count, pct_count, digit_count, digit_ratio, letter_count, special_char_count, params_count, path_depth, and subdomain_count. These features capture the intuition that phishing URLs tend to be longer, contain more subdomains, and include more special characters than legitimate URLs. The second category covers binary structural flags including has_https, has_http, has_ip (IP address in domain),

has_at, has_double_slash, has_hex_encoding, has_port, has_fragment, is_trusted_tld, is_suspicious_tld, domain_is_alexa, is_shortener, has_punycode, tld_in_subdomain, domain_has_digits, domain_has_hyphen, http_in_path, and has_www.

The third category covers entropy-based features. The _entropy() function computes the Shannon information entropy of a string by calculating the character frequency distribution and applying the formula $H = -\sum(p_i \times \log_2(p_i))$. This is applied separately to the full URL (url_entropy), the domain component (domain_entropy), and the URL path (path_entropy). High entropy values in domain names are a strong indicator of algorithmically generated domains used in fast-flux phishing campaigns and botnet command-and-control infrastructure. The _consonant_ratio() function computes the proportion of consonant characters in the domain name, as legitimate domains tend to have pronounceable names with balanced vowel-consonant ratios, while randomly generated domains exhibit abnormally high consonant ratios.

The fourth category covers domain composition features including subdomain_len (total character length of all subdomain parts), longest_subdomain, domain_hyphen_count, and longest_digit_seq (the maximum run length of consecutive digits in the URL). The fifth category covers semantic and heuristic features. The suspicious_keyword_count feature counts occurrences of forty-five predefined suspicious keywords including login, verify, update, secure, account, bank, paypal, signin, confirm, reset, webscr, checkout, billing, password, credential, wallet, recovery, unlock, suspend, alert, urgent, free, bonus, gift, lucky, win, prize, click, download, install, and redirect within the lowercase URL string. The has_brand_impersonation feature detects the presence of fifteen major brand names (paypal, apple, amazon, google, microsoft, netflix, facebook, instagram, ebay, chase, wells Fargo, irs, usps, fedex, dhl) in the URL when the root domain does not belong to the legitimate brand. The phish_hints feature counts occurrences of eight high-signal phishing keyword indicators in the URL.

7.2 Ensemble Phishing Classification Model

The phishing detection model is an ensemble of two classifiers: a Gradient Boosting (GB) classifier and a Random Forest (RF) classifier, both trained on a labeled dataset of 48,729 URLs using the fifty features described above. The model was trained with a reported accuracy of 95 percent on held-out test data. The predict_url() function in api.py implements the prediction pipeline: it calls extract_features(url) to

obtain the feature dictionary, constructs a pandas DataFrame from this dictionary, invokes `gb.predict_proba(df)` and `rf.predict_proba(df)` to obtain class probability arrays from each classifier, and then computes the final phishing risk score using a weighted combination formula: $\text{risk} = ((2 \times \text{GB_prob} + \text{RF_prob}) / 3)[1] \times 100$, where `[1]` indexes the probability of the positive (phishing) class and the factor 2 assigns double weight to the Gradient Boosting classifier relative to the Random Forest.

This weighting scheme reflects the empirical observation that Gradient Boosting classifiers typically achieve higher precision on imbalanced datasets such as phishing URL collections, where the distribution of legitimate versus phishing samples may be unequal. The final risk score is rounded to one decimal place and ranges from 0 to 100. A threshold of 50 is applied to determine the binary label: URLs with risk scores at or above 50 are labeled Phishing while those below the threshold are labeled Legitimate. The risk score is then passed to the composite scoring and threat classification functions as the `ml_risk` parameter.

7.3 Spam Detection Pipeline

A separate spam detection pipeline operates in parallel with the phishing model during URL analysis. The `predict_spam()` function invokes the `spam_model.pkl` pipeline, passing the raw URL string as input text. The model returns a binary prediction (True for spam, False for legitimate) and a confidence probability derived from the `predict_proba()` output. The `is_spam` flag and spam confidence score are included in the ML result block of the scan response and are incorporated into the threat classifier's decision tree for spam and scam category assignments.

7.4 Composite Danger Score Computation

The `compute_composite_score()` function in `api.py` implements the multi-source threat score aggregation algorithm. For URL mode, the function combines up to five signal sources using a fixed weighted scheme: the ML model phishing risk contributes 30 percent, VirusTotal contributes 35 percent, AbuseIPDB contributes 15 percent, IPQualityScore contributes 15 percent, and Google Safe Browsing contributes 5 percent. Each source score is mapped to a 0-to-100 scale before aggregation. The VirusTotal score is computed as $(\text{malicious} + 0.5 \times \text{suspicious}) / \text{total} \times 100$, capped at 100. The AbuseIPDB abuse confidence score is directly used on its native 0-to-100 scale. The IPQualityScore risk score is similarly used directly.

Google Safe Browsing contributes a binary signal: 100 if the URL is flagged, 0 otherwise. The overall composite score is computed as the weighted average of all available signals, normalized by the sum of the weights of available sources. This design ensures that the composite score degrades gracefully when one or more APIs are unavailable, re-normalizing the weights across the remaining available sources. For file mode, the composite score is determined solely by the VirusTotal malicious ratio, as ML and IP-based signals are not meaningful for file content analysis.

7.5 Master Threat Classifier

The `classify_threat()` function implements a deterministic rule-based classifier that maps the combined signals from all analysis engines to a discrete threat category, risk level, and threat label. For URL mode, the classifier applies a cascading priority rule sequence: URLs with fifteen or more malicious detections on VirusTotal are classified as Exploit Critical at critical risk; ten or more detections trigger Botnet/C2 critical classification; seven or more trigger Ransomware critical; five or more, or fewer with a concurrent Google Safe Browsing flag, trigger Malware high; three or more trigger Virus high; a combination of two or more detections with an ML phishing label triggers Phishing high; one or more detections with an ML risk above 50 percent also triggers Phishing high. Subsequent rules handle Google Safe Browsing flags, IPQualityScore phishing or malware signals, spam indicators, suspicious defacement patterns, and pure ML-based classifications for URLs that score below detection thresholds on all API engines. The `risk_level` string (safe, medium, high, critical) is then mapped to the corresponding Threat-Level value (SAFE, WARNING, DANGER, CRITICAL) through a lookup dictionary.

For file mode, the classifier applies a separate rule sequence based on VirusTotal malicious engine counts and filename heuristics: fifteen or more malicious detections, or filenames containing ransom, crypt, lock, or wanna trigger Ransomware critical; ten or more trigger Virus critical; five or more trigger Malware high; three or more for APK or IPA files trigger Spyware high; one or more for files matching bot, rat, c2, cmd, or agent keywords trigger Botnet/C2 critical; a single malicious detection triggers Malware high; five or more suspicious detections trigger Exploit Critical; two or more suspicious trigger Suspicious medium; and zero detections result in a Benign safe classification. This dual-mode classifier enables ATL to produce contextually appropriate threat categories

for both web resource and file-based threats.

8. IMPLEMENTATION DETAILS

8.1 Technology Stack

The frontend technology stack comprises React 18.3 as the core UI library, TypeScript 5.9 for static type safety across all client-side code, React Router 6.30 for SPA routing, TailwindCSS 3.4 for utility-first styling, Vite 8.0 as the build tool and development server, TanStack React Query 5.84 for server state management, Framer Motion 12.23 for animation, Recharts 2.12 for data visualization, Three.js via `@react-three/fiber` and `@react-three/drei` for any 3D rendering, Radix UI for accessible headless UI primitives, Lucide React for the icon system, Sonner for toast notifications, and the custom `shadcn/ui` component library built on top of Radix UI. The `class-variance-authority` and `tailwind-merge` utilities are used for conditional class composition through the `cn()` helper function defined in `client/lib/utls.ts`.

The backend technology stack comprises Node.js 22, Express.js 5.1, TypeScript compiled via `tsx` for development and bundled by Vite for production, Multer 1.4 for multipart file upload handling with disk storage strategy, Zod 3.25 for runtime schema validation, `dotenv` 17.2 for environment variable management, and `cors` 2.8 for cross-origin request configuration. The production build is generated as a single ESM module (`node-build.mjs`) that bundles all server-side code including route handlers and serves the pre-built React SPA from the `dist/spa` directory. The package manager for the project is `pnpm` version 10.14.

The Intelligence Layer technology stack comprises Python 3.14, Flask 3.0 as the HTTP server framework, Flask-CORS for cross-origin support, `joblib` 1.3 for model serialization and deserialization, `scikit-learn` 1.3 for the ML classifiers and pipeline objects, `pandas` 2.0 for DataFrame construction, `NumPy` 1.24 for numerical array operations, and `requests` 2.31 for all outbound HTTP calls to external threat intelligence APIs. The ML models were serialized using `joblib` and stored as `.pkl` binary files within the `python_engine` directory, loaded once at engine startup. The engine is started using the `start.sh` shell script on Linux/macOS or `start.bat` on Windows, both of which invoke `python api.py` from within the `python_engine` directory.

8.2 Project File Structure

The project follows a monorepo structure rooted at the `atl-integrated` directory. The `client/` directory contains the entire React SPA codebase, organized into

`pages/` (route-level components including `Index.tsx`, `Login.tsx`, and `NotFound.tsx`), `components/` (reusable components including the `ATLLogo`, `Navbar`, `Footer`, and the complete `shadcn/ui` component library under `components/ui/`), `hooks/` (custom React hooks including `use-theme.ts`, `use-mobile.tsx`, and `use-toast.ts`), `lib/` (the `cn()` utility and test specifications), and `global.css` (TailwindCSS theme configuration including custom CSS variables for the ATL color scheme). The `server/` directory contains the Express application organized into `index.ts` (server factory function with all route registrations), `routes/scan.ts` (the three scan route handlers and the health check handler), `routes/demo.ts` (a demonstration endpoint), `dev.ts` (the development server entry point using `tsx`), and `node-build.ts` (the production build entry point). The `shared/` directory contains `api.ts` with all TypeScript interfaces shared between client and server. The `python_engine/` directory contains `api.py` (the complete Flask ML engine), `features.py` (the URL feature extraction module), `phishing_model_v3.pkl` and `spam_model.pkl` (serialized model files), `feature_names.pkl` (feature column order), `requirements.txt` (Python dependencies), `README_SETUP.md` (API key setup guide), and startup scripts for all platforms. Configuration files at the project root include `package.json`, `tsconfig.json`, `vite.config.ts`, `vite.config.server.ts`, `tailwind.config.ts`, `postcss.config.js`, `components.json`, `netlify.toml`, and `.env` with the `ML_ENGINE_URL` environment variable.

8.3 Authentication and SessionManagement

The authentication system is implemented as a client-side session mechanism using the browser's `localStorage` API. The Login page, defined in `client/pages/Login.tsx`, presents a form accepting an email address and password. On form submission, the system validates that both fields are non-empty and writes a flag (`atl-authenticated = true`) to `localStorage`. If the Remember Me checkbox is unchecked, the value `session` is stored instead of `true`, enabling the application to differentiate between persistent and session-scoped authentication states. The main dashboard performs a `localStorage` check on mount and redirects unauthenticated users to the login page using React Router's `Navigate` component. Sign-out is accessible from the profile menu in the `Navbar` and clears the authentication flag from `localStorage`.

8.4 Server-Sent Events Streaming Implementation

The real-time terminal interface is powered by a Server-Sent Events (SSE) architecture spanning all three tiers of the system. On the Python side, the `/analyze/url/stream` endpoint is implemented as a Flask generator function that yields SSE-formatted data strings after each processing step. Each event is a JSON object containing a type field (step or result), a step identifier, a human-readable message, and a status value. The generator yields events at eight points during the analysis: after URL validation, after ML feature extraction and prediction, after VirusTotal query, after Google Safe Browsing lookup, after AbuseIPDB IP resolution and check, after IPQualityScore query, after URLScan.io submission, and after threat classification. The final event of type result contains the complete ScanResponse payload that would normally be returned by the standard JSON endpoint.

On the Node.js side, the `handleUrlScanStream` handler sets the SSE response headers immediately on request receipt, then opens a fetch connection to the Python engine's streaming endpoint using the Fetch API with a five-minute AbortSignal timeout. The response body is consumed using a ReadableStream reader, and each chunk of bytes is decoded using a TextDecoder in streaming mode and written to the Express response using `res.write()`. The `res.flush()` method is called on each chunk to prevent buffering in reverse proxy environments such as Nginx (controlled by the `X-Accel-Buffering: no header`). On the React side, the frontend opens an EventSource connection to `/api/scan/url/stream`, parses each incoming JSON data payload, and updates the corresponding step in the StepState array using React's `useState` updater function, triggering a re-render that displays the updated terminal entry. When the final event of type result is received, the EventSource is closed and the full result is passed to the ThreatResultPanel for rendering.

9. SYSTEM DESIGN

9.1 System Workflow

The end-to-end workflow of the ATL system begins when an unauthenticated user navigates to the application root URL. The React Router intercepts the navigation request, checks for the authentication flag in `localStorage`, and redirects the user to the login page. Upon successful submission of the login form, the authentication flag is written to `localStorage` and the user is redirected to the main dashboard. The

dashboard renders the scan interface in URL mode by default, displaying the URL input field, file upload drop zone (inactive in URL mode), and the live terminal panel in its initial waiting state.

When the user enters a URL and initiates a scan, the React component calls the `scanUrl()` function, which dispatches a POST request to `/api/scan/url/stream`. The Express route handler sets SSE headers and opens a proxied streaming connection to the Python ML engine. The Python engine begins executing the eight-step analysis sequence: URL sanitization (automatic addition of the `https://` scheme if absent), regex-based URL validation, ML feature extraction using the fifty-feature extractor, ensemble prediction producing the phishing risk score and label, spam detection pipeline inference, parallel VirusTotal URL lookup by base64-encoded URL ID, Google Safe Browsing API query using the Threat Match endpoint, AbuseIPDB IP reputation check (with DNS resolution of the domain to its IP address), IPQualityScore URL analysis, URLScan.io public scan submission, and composite danger score computation followed by threat classification. Each step emits an SSE event upon completion, which traverses the proxy to the browser and updates the terminal display.

When the file scanning mode is selected, the user drags and drops or selects a file through the file input control. The React component reads the file object and dispatches a multipart FormData POST request to `/api/scan/file`. The Express handler receives the file through Multer, which writes it to the OS temporary directory. The handler then reads the file from disk, constructs a new FormData object with the file as a Blob, and issues a POST request to the Python engine's `/analyze/file` endpoint. The Python engine streams the file from the Flask request to a temporary file in 64-megabyte chunks, computes the SHA256 hash through chunked reading, performs the VirusTotal cache lookup and upload flow, executes the file threat classifier, computes the composite score, and returns the ScanResponse JSON. The Express handler forwards this response to the browser, and the React component updates the result panel accordingly. Both the Express-side and Python-side temporary files are deleted in their respective finally blocks after scan completion.

9.2 Data Flow Architecture

The data flow architecture can be characterized as a multi-fanout request pattern. A single URL scan request from the browser triggers up to seven outbound HTTP requests from the Python engine: one to the

VirusTotal URL analysis endpoint, one to the Google Safe Browsing Threat Match endpoint, one to the AbuseIPDB Check endpoint (following a DNS resolution), one to the IPQualityScore URL Reputation endpoint, and one to the URLScan.io scan submission endpoint. These requests are executed sequentially in the streaming generator to allow progress events to be emitted after each one. The sequential execution design was chosen over parallel execution (asyncio or threading) to preserve the ordered step-by-step visibility in the terminal interface, which is a key user experience feature of the system.

Each external API call is wrapped in a try-except block that returns a default failure response structure (with success: False and zero counts) if the request times out, the API returns an error code, or any other exception occurs. This defensive programming pattern ensures that the failure of any single external API does not abort the overall scan, and the composite score algorithm gracefully handles None values in the breakdown structure by re-normalizing the weighted average over only the available signal sources. API timeout values are configured conservatively: 10 seconds for AbuseIPDB, IPQualityScore, Google Safe Browsing, and URLScan.io; 10 seconds for VirusTotal URL lookups; and 30 minutes for VirusTotal large-file uploads.

10. RESULTS AND DISCUSSION

The ATL – AI Threat Lens system was evaluated across multiple scanning scenarios to assess detection accuracy, response latency, user interface responsiveness, and classification correctness. The evaluation encompassed URL scans of known phishing pages sourced from active phishing databases, known legitimate web services, and edge-case URLs exhibiting suspicious structural patterns without confirmed malicious intent. File scans included clean benign executables, known malware samples with documented VirusTotal histories, and benign APK packages.

For URL analysis, the ML ensemble demonstrated consistent classification behavior aligned with the reported 95 percent training accuracy. URLs containing multiple phishing keywords, high subdomain depth, non-trusted TLDs, and brand impersonation indicators received phishing risk scores above 70 percent. Legitimate top-level domains from the Alexa Top set received scores below 15 percent when accompanied by clean VirusTotal and AbuseIPDB results. The composite danger score was observed to increase significantly when VirusTotal engine counts exceeded zero, reflecting the 35 percent weight assigned to VirusTotal in the scoring formula. URLs flagged by

Google Safe Browsing received an immediate composite score increment of 5 percentage points, which, combined with other signals, consistently pushed them into the WARNING or DANGER risk band.

The Server-Sent Events streaming mechanism was observed to deliver the first terminal event within approximately 150 milliseconds of scan initiation under typical network conditions, confirming that the SSE proxy architecture introduces negligible latency overhead. Subsequent events arrived at irregular intervals corresponding to the actual API response times of each external service. Total URL scan completion time was observed in the range of 8 to 25 seconds depending on the responsiveness of the external APIs, with VirusTotal lookups being the dominant contributor to total scan duration. File scans for files smaller than 32 megabytes that were already present in the VirusTotal cache completed within 5 seconds, while new file uploads required 30 to 120 seconds for analysis completion.

The threat categorization system produced accurate and contextually appropriate labels across all tested scenarios. Ransomware samples with fifteen or more VirusTotal malicious detections were correctly classified as Ransomware critical. Known phishing pages with ML risk scores above 50 and one or more VirusTotal detections were classified as Phishing high. URLs with no VirusTotal detections, clean AbuseIPDB scores, and ML risk below 15 percent were classified as Legitimate safe. The natural-language threat descriptions rendered in the ThreatResultPanel were observed by test users to significantly improve their understanding of the threat and their confidence in the system's recommendation, validating the design principle of explainable AI-driven threat classification.

10.1 Comparative Analysis

Table 1 presents a comparative analysis of the ATL system against representative existing cybersecurity tools across key functional dimensions.

Feature	VirusTotal	Browser Extensions	ATL – AI Threat Lens
URL Scanning	Yes	Yes (blacklist)	Yes (ML + APIs)
File Scanning	Yes	No	Yes (VT + classifier)
Machine Learning	No	No	Yes (ensemble model)

Real-Time Streaming	No	No	Yes (SSE terminal)
Composite Danger Score	No	No	Yes (0–100 weighted)
Threat Description	No	No	Yes (per category)
Zero-Day URL Detection	Partial	No	Yes (ML-based)
Unified URL + File Interface	No	No	Yes (single dashboard)

Table 1: Comparative Analysis of ATL – AI Threat Lens against Existing Systems.

11. ADVANTAGES OF THE PROPOSED SYSTEM

The proposed ATL system offers several distinct advantages over conventional cybersecurity tools. The most significant advantage is the provision of a unified scanning interface for both URLs and files, eliminating the need for users to switch between multiple disconnected applications when assessing different types of digital resources. The integration of an ensemble machine learning model enables detection of zero-day phishing URLs that have not yet appeared in any antivirus blacklist, addressing a critical gap in coverage that purely signature-based tools cannot fill. The real-time SSE terminal interface provides a level of transparency that is absent from all existing consumer-grade threat scanning tools. Users can observe each analysis step as it executes, understand which APIs were queried and what each returned, and develop an informed appreciation of how the final threat verdict was reached. This observability transforms the system from a black-box oracle into an educational tool that builds user cybersecurity literacy over time. The natural-language threat category descriptions further enhance this educational value by explaining not just what threat was detected but what harm that category of threat is capable of causing. The weighted composite danger score provides a single normalized metric that can be compared across scans and over time, enabling users to develop a personal threat baseline and respond proportionately to different risk levels. The four-tier risk level classification with color-coded visual indicators (green for SAFE, amber for WARNING, red for DANGER, purple for CRITICAL) enables rapid threat triage that

requires no technical expertise. The modular three-tier architecture ensures that any individual component (the ML model, a specific API integration, or the frontend design) can be upgraded independently without requiring changes to the other tiers, providing long-term maintainability and extensibility. Support for very large file uploads up to 15 gigabytes, with streaming-based memory-efficient processing, makes the system capable of scanning enterprise-scale artifacts that would exceed the memory capacity of naive in-memory processing approaches.

12. FUTURE ENHANCEMENTS

The ATL system provides a strong foundation upon which several important enhancements can be built in future development iterations. The most immediate enhancement opportunity lies in the replacement of the current sequential API query pattern with an asynchronous parallel execution model using Python's asyncio library and the aiohttp framework. Parallel execution of the five external API calls would reduce total URL scan latency from the current 8-to-25 second range to approximately 3-to-8 seconds, corresponding to the response time of the slowest individual API rather than the sum of all API times. The SSE streaming architecture could be adapted to deliver progress events in arrival order rather than execution order.

A persistent backend database layer could be introduced to store historical scan results indexed by URL hash or file SHA256. This would enable cache-based instant results for previously analyzed resources, user-level scan history with trend analysis across time, aggregated threat intelligence dashboards showing the most commonly detected threat categories in the user's scan history, and model improvement feedback loops where misclassified results reported by users can be incorporated into future model retraining datasets. MongoDB or PostgreSQL would be suitable choices for this persistence layer.

The machine learning model could be upgraded from the current Gradient Boosting and Random Forest ensemble to a deep learning architecture such as a Bidirectional LSTM or BERT-based transformer applied to character-level URL tokenization. Deep learning models have demonstrated superior generalization on adversarially crafted phishing URLs that deliberately evade feature-engineered classifiers by exploiting the specific thresholds used in the feature extraction pipeline. The integration of WHOIS registration date data, DNS record freshness, SSL certificate validity period, and content-based features extracted from the rendered page DOM could further improve detection

accuracy for sophisticated phishing campaigns. Additional planned enhancements include the development of a native mobile application for iOS and Android that exposes the scan interface through a share-sheet extension, enabling users to scan any URL directly from their browser, email client, or messaging application without opening a separate app. Enterprise deployment configurations with role-based multi-user support, organizational scan history aggregation, SIEM integration via webhook callbacks, and API key management for downstream application integration would extend the system's value to security operations centre environments. Integration with additional threat intelligence feeds such as OpenPhish, PhishTank, Shodan, and Censys would broaden the system's coverage across the threat landscape.

13. CONCLUSION

ATL – AI Threat Lens represents a comprehensive and principled approach to accessible cybersecurity threat detection that successfully bridges the gap between advanced security tools and everyday users. The system addresses the fundamental limitations of existing security solutions through a unified, modular, three-tier architecture that combines an ensemble machine learning model trained on 48,729 URLs with fifty engineered features, five integrated real-time threat intelligence APIs, and a weighted composite danger scoring algorithm that synthesizes all available signals into a single normalized metric.

The implementation demonstrates that a full-stack cybersecurity platform built with modern web technologies — React, TypeScript, Express.js, TailwindCSS, Python, Flask, and scikit-learn — can deliver enterprise-grade threat analysis capabilities through a consumer-friendly interface. The Server-Sent Events streaming architecture enables an unprecedented level of analysis transparency by exposing the complete execution pipeline to the user in real time, transforming threat scanning from a black-box process into an observable, educational experience. The master threat classifier's dual-mode operation for URL and file threats and its twenty-category classification taxonomy provide actionable, contextually appropriate verdicts that support informed user decision-making.

The system's memory-efficient file processing architecture, which supports uploads of up to 15 gigabytes through chunked disk streaming and SHA256-based VirusTotal cache lookup, demonstrates that practical constraints of large-scale file scanning can be addressed within the constraints of a web-based architecture. The modular design of all three tiers ensures

that the ML model, API integrations, and frontend interface can each be independently upgraded as the threat landscape evolves, providing a sustainable platform for long-term cybersecurity capability development. The ATL system confirms that the combination of machine learning, multi-source threat intelligence, and thoughtful user experience design can significantly improve the accessibility and effectiveness of cybersecurity tools for users across all levels of technical expertise.

ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to the management and faculty members of their institution for their continuous support and encouragement throughout the development of this project. The authors are especially thankful to their project supervisor for valuable guidance, constructive suggestions, and consistent motivation, which played a crucial role in the successful completion of this work. The authors also extend their appreciation to the teams behind VirusTotal, Google Safe Browsing, AbuseIPDB, IPQualityScore, and URLScan.io for providing accessible threat intelligence APIs that made the multi-engine analysis architecture of this system possible.

REFERENCES

- [1] Sahingoz, O. K., Buber, E., Demir, O., Diri, B.: Machine Learning Based Phishing Detection from URLs. *Expert Systems with Applications*, Vol. 117, pp. 345–357, 2019.
- [2] Mohammad, R. M., Thabtah, F., McCluskey, L.: Intelligent Rule-based Phishing Websites Classification. *IET Information Security*, Vol. 8, No. 3, pp. 153–160, 2014.
- [3] Gutierrez, C. N., Kim, T., Della Corte, R., Avery, J., Goldwasser, D., Garg, S., Bagchi, S.: Learning from the Ones That Got Away: Detecting New Forms of Phishing Attacks. *IEEE Transactions on Dependable and Secure Computing*, 2018.
- [4] Buber, E., Diri, B., Sahingoz, O. K.: NLP Based Phishing Attack Detection from URLs. *International Conference on Intelligent Systems Design and Applications (ISDA)*, pp. 608–618, 2017.
- [5] Rajab, M. A., Ballard, L., Mavrommatis, P., Provos, N., Zhao, X.: The Nocebo Effect on the Web: An Analysis of Fake Anti-Virus Distribution. *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2010.
- [6] Chen, T., Guestrin, C.: XGBoost: A Scalable Tree

Boosting System. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794, 2016.

[7] Breiman, L.: Random Forests. Machine Learning, Vol. 45, No. 1, pp. 5–32, 2001.

[8] VirusTotal. VirusTotal API v3 Documentation. Available at: <https://developers.virustotal.com/reference/overview> (accessed April 2026).

[9] Google. Safe Browsing API Documentation v4. Available at: <https://developers.google.com/safe-browsing> (accessed April 2026).

[10] IPQualityScore. URL Reputation and Malicious URL Lookup API. Available at: <https://www.ipqualityscore.com/documentation> (accessed April 2026).