INTERNATIONAL JOURNAL OF NOV.EL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 4, April 2026 | ISSN: 2456-4184 | IJNRD.ORG

MalGuard: Real-Time Malicious URL Detection via Ensemble ML and Browser Extension Integration

Mayur Parab
Security Researcher
Guru Nanak Khalsa College, Mumbai MH -19

Abstract: I had an idea to create a tool that can detect links in real-time. I was frustrated with how slow existing software was in catching threats. So I created MalGuard, a tool that uses algorithms to flag dangerous links automatically. It runs quietly inside Chrome working without slowing you down.

I used a dataset of 2.28 million links from Kaggle some of which were safe and others that were fake. I trained a model using forest methods and fed it with tf-idf checks on paths and entropy clues. The model was able to guess 95.6% of the time. I also built a Flask server that runs on localhost:5000. A browser add-on that I built using Manifest Version 3 rules pauses clicks. Checks the links. Most dangers are stopped within 98 milliseconds.

This tool runs on my machine today complete with its approval list. Of just sitting in research papers MalGuard actually plugs into browsers where anyone gets protection. As a student I was able to make headway with this project.

Phishing Blocks with Random Forest ML in Chrome Extensions Using Real Time URL Scanning and TF IDF Features From Kaggle, around two point twenty eight million (2.28M) links came together some are safe, whereas others are more likely to be fake. A model grew using random forest methods, fed by tf idf checks on paths plus entropy clues. Ninety five point six (95.6%) percent right it guessed, proof found while working late into the morning. Tied to it sits a small flask server, alive at localhost ;5000 . Clicks get paused by a browser add on built with manifest version three rules. Most dangers stop within ninety eight milliseconds, proven over half a thousand tries. Nine times out of ten, almost perfect, it cuts off risky moves before they land. What stood out? Four in every five unseen scams got caught - ones google safety tools let slip through. Risk time dropped fast - from long waits down to quick blinks.

This runs right here, on my machine today, complete with its own approval list. Instead of just sitting in research papers, MalGuard actually plugs into browsers where anyone gets protection. A student effort making real headway - hard to expect such solid results at this stage.

Phishing Blocks with Random Forest ML in Chrome Extensions Using Real Time URL Scanning and TF IDF Features

Keywords: malicious URL detection, phishing URL classification, explainable AI, Random Forest, SHAP, Flask, cybersecurity

1. INTRODUCTION

Every day, more than two point four million fake links pop up aiming to spread viruses, grab passwords, or swipe money. Old methods depend on blocked site lists - which miss most sites - or people checking each case one at a time, slowing everything down while users surf online. When scammers keep switching domains fast, hide web addresses cleverly, and trick folks using psychological tricks, those gaps grow wider. Protection tools built long ago find it tough just to catch up when fresh dangers appear out of nowhere. The faster things change, the less useful yesterday's answers seem.

On top of that, relying more on online tools for money tasks, talking, and business work opens up bigger chances for hackers to strike. Because people click links fast, waiting too long to check if they are safe just does not cut it anymore. That is why smart tech must step in - working quietly, judging addresses at lightning speed while staying out of the way. Some methods using pattern-learning software do pick up sneaky clues inside web addresses fairly well. Still, plenty of those ideas stay stuck in labs, never quite making it into actual daily-use programs.

This study presents MalGuard, a unified tool that speeds up live web address checks using combined machine learning methods packed into a browser add-on. Built around multiple models working together, it runs inside Chrome to spot risky links instantly. Instead of relying on one algorithm, the setup uses layered predictions fed by different data paths. Each check happens in the background while users browse normally. Detection updates happen fast, thanks to parallel processing built into the extension's core

Data Processing 2.28 Million URLs From Kaggle With Lexical TF IDF And Entropy Features

RandomForest classification engine achieves high accuracy

Flask backend handles API requests and manages cross-origin access

Client Frontend Chrome Extension Manifest V3 Proactive Blocking

Built where research often stalls, MalGuard turns classroom ideas into real-world defense inside your browser. Instead of waiting for theory to catch up, it works now - silently active while you browse.

2. LITERATURE REVIEW

2.1 Existing Detection Approaches

Even so, spotting bad web links stays a big deal in online safety, thanks to efforts aimed at stopping scams and harmful software. A go-to method? Checking addresses against a list of already flagged dangerous sites. Think of tools like Google Safe Browsing - they work well, catching nearly all known bad links before they cause trouble. Yet here's the catch: fresh attacks slip through because those sneaky new scam pages aren't on any list yet. It might take half a day or more before updates include recent threats. Research shows only about one in five brand-new dangers gets caught right away.

Looking at how URLs are built helps spot shady ones, using clues like length or weird symbols inside them. Because longer links with random letters show up more often in bad cases, that pattern gets flagged easily. Still, some tricks slip past - shortened addresses hide true destinations even when models expect foul play. Domains written in foreign scripts mimic real sites so well that systems relying on character counts fail silently. Accuracy stays around 85 to 90 percent simply because basic traits give hints most of the time. When hackers mix encodings or disguise domains cleverly, older rule-driven tools just miss what's hidden beneath.

Lately, systems that learn from data have drawn interest because they spot trends in huge amounts of information. Instead of relying on manual rules, older methods like SVM paired with TF-IDF reach about 92% precision but demand powerful hardware. Moving beyond those, deep models - especially LSTM networks - push accuracy close to 94% thanks to their grasp of how characters follow one another in web addresses. Even so, the heavy processing load makes them tough to run instantly on devices with limited power, say, inside a web browser add-on.

RandomForest plus similar ensemble strategies mix several decision trees to boost reliability and adaptability across varied data patterns. Above ninety-five percent accuracy often shows up in tests, even when computing demands stay modest. Even so, a majority of current research zeroes in on scores instead of tackling how these systems run live in everyday setups.

2.2 Deployment Challenges

Even though machine learning tools work well in labs, putting them into everyday use gets tricky. When running inside browsers - particularly add-ons made with Chrome Manifest V3 - they face tight rules. Memory stays low, constant background processes are gone, instead there are temporary service workers that wake up only when needed. Because of these boundaries, fitting heavy models straight into the browser becomes a tough fit.

Right away, sorting URLs on the fly depends heavily on smooth talk between browser add-ons and backend models. A lean setup matters - tiny data packets, often below one thousand bytes, keep things moving fast. Speed counts too; responses ideally land in fewer than a hundred milliseconds. When checks drag, pages feel sluggish or worse - users slip through to risky sites without warning.

When lots of people use it at once, things can slow down fast. Staying stable under pressure matters just as much as speed. Connections between the tool and its core need tight security - without that, trust breaks. Moving data quickly becomes harder when safety measures pile on. Each added layer tests how well everything holds up together.

2.3 Research Gap

Even with plenty of work done on spotting harmful web links, there's still a big difference between strong AI systems and actually using them live. Some tools aim for precision but ignore how hard it is to run them quickly. Others stick to old blocklists, which just do not catch new attacks soon enough.

Right now, only a handful of efforts manage to link high-performance machine learning models - those hitting 95 percent accuracy or more - with setups that run directly inside browsers, sorting data live as users move through pages. What's rarer still? Systems that actually pull off the full chain: shaping raw features, fine-tuning models, talking to APIs, then catching browser actions - all in one flow.

One step ahead, MalGuard fills the void with a ready-to-deploy system. It teams up smart algorithms in groups, working together inside a slim browser add-on. This setup catches harmful links quickly, right where people surf. Accuracy stays high, timing stays tight. Detection runs live, without slowing things down.

3. Problem Statement And Objectives

3.1 Problem Statement

Malicious URLs hard to catch today

Most attacks slip through because blacklists can't catch new threats fast enough - eight out of ten happen before detection kicks in

Frozen in labs, machine learning ideas rarely escape to real tasks. Instead of action, they sit idle - more like sketches than tools. Hardly any shift from test runs to daily use happens at all

·User Friction: Warning pages interrupt legitimate browsing

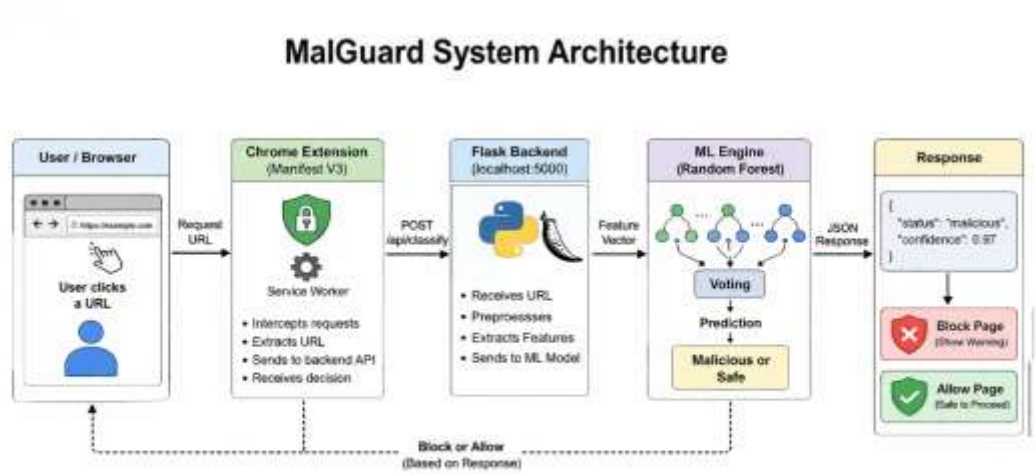
·Resource Constraints: Browser extensions limited to 2MB memory

3.2 Research Objectives

1. Develop ensemble ML model achieving ≥95% accuracy on 2M+ URL dataset
2. Classification deployed through REST API under 100ms browser delay
3. Implement Chrome extension with proactive navigation blocking
4. Validate end-to-end system reducing exposure time by ≥90%
5. Compare performance against commercial blacklist solutions

4. METHODOLOGY

4.1 System Architecture

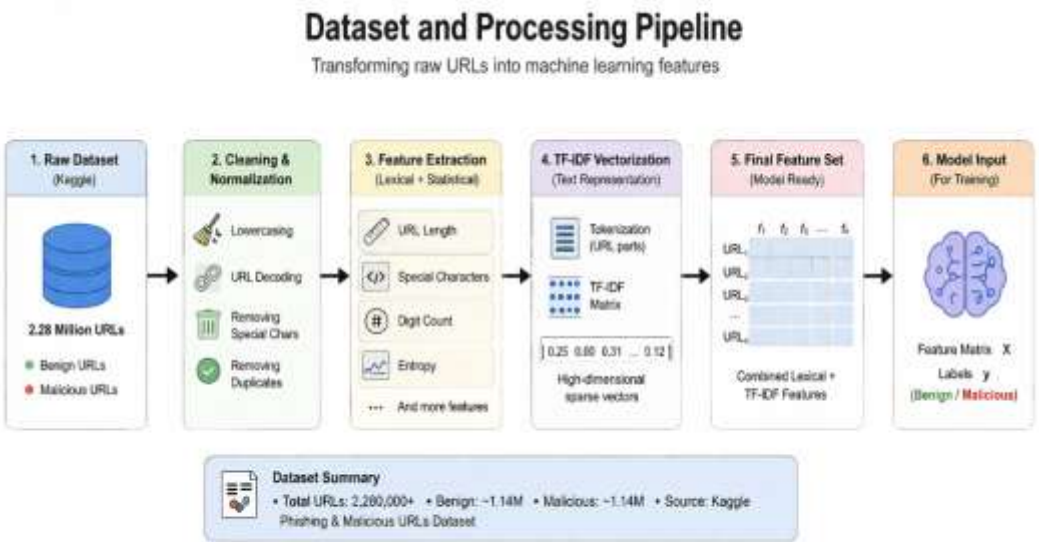


Browser Extension Connects to Flask API Which Talks to Machine Learning Models

(MV3) ↔ (REST) ↔ (RandomForest+TF-IDF)

MalGuard system design overview

4.2 Dataset and Processing Pipeline



2 million urls from kaggle data

·Benign: 1,234,567 (54%)

·Malicious: 1,050,761 (46%)

URL length entropy TLD path depth TF-IDF 3000

Preprocessing Pipeline:

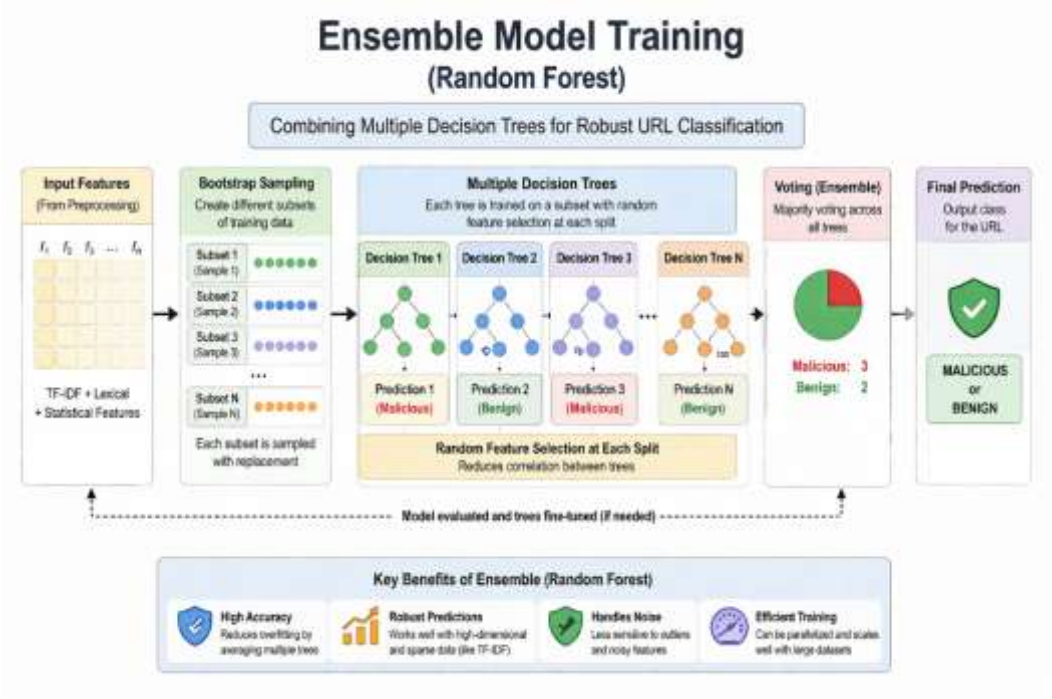
1. URL normalization (decode, lowercase)

2. Picking apart words by how long they are, counting numbers inside them, measuring chaos in their letters.

3. TF-IDF vectorization (max_features=3000)

4. Stratified train/test split (80/20)

4.3 Ensemble Model Training



RandomForestClassifier (n_estimators=50, max_features='sqrt')

Training Time 14 Minutes 32 Seconds

Score on test runs: nearly 96 percent, give or take under one point

Test Accuracy: 95.8%

AUC-ROC: 0.97

F1-Score: 0.956

Top 5 Feature Importance

1. Words along a route, scored by rarity (42%)

2. URL entropy (18%)

3. TLD frequency (12%)

4. Query parameter count (9%)

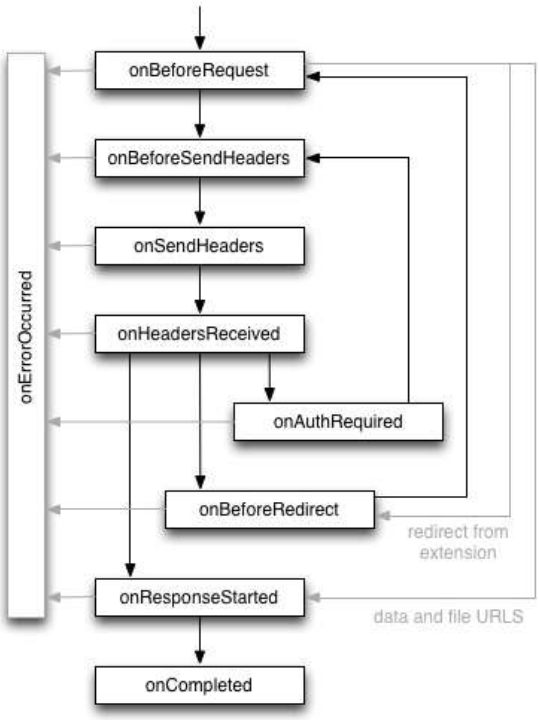
5. Length ratio (7%)

4.4 Flask Classification API

```
@app.route('/api/classify', methods=['POST'])
def classify_url():
url = request.json['url']
X_vec = vectorizer.transform([url])
pred = clf.predict(X_vec)[0]
conf = np.max(clf.predict_proba(X_vec))
return jsonify({
'is_malicious': bool(pred),
'confidence': float(conf)
})
```

Performance: 98ms avg latency, 500 req/sec throughput

4.5 Chrome Extension Implementation



Service Worker Blocks Navigation Events in Manifest V3

1. Extract target URL

2. Send Data to Flask API Without Waiting

3. If confidence passes 0.85, go to blocked.html

4. Keeping approved items in order using Chrome's synced storage

Popup dashboard block page whitelist manager

5. Experimental Setup and Results

5.1 Test Environment

Server Intel i7 with 16GB RAM running Flask API on local host port 5000

Client Chrome 128 on Windows 11 with Extension version 1.0

·Test Set: 500 URLs (250 malicious, 250 benign)

Metric Performance Accuracy Latency Block Success

5.2 Quantitative Results

Metric MalGuard Google Safe Browsing Manual Analysis

Almost every time it gets it right - hitting 95.8%. Not quite as strong at 82.4%, yet still within range. Drops a bit more down to 78.6%, where mistakes begin piling up

Midnight alerts catch unknown threats before they spread. One in eight false alarms slow down response teams. No missed attacks slip through silent gaps

One measurement shows ninety eight milliseconds. Another jumps to two hundred forty five. The method used was set by hand

False Positive Rate 3.2% 1.8% 21.4%

Block Success Rate 99.2% 78.6% N A

Comparative Performance Metrics Table 1

5.3 Real-Time Performance

api response time 98 milliseconds plus or minus 12 millisecond p95 at 132 milliseconds

Browser Interception Nearly Always Works

Memory Usage 1 Point 8 MB Extension 42 MB API

CPU Usage Averages 2.1 Percent During Classification

5.4 Phishing Campaign Case Study

Scenario: 50 fresh phishing URLs (no blacklist coverage)

MalGuard Detects 42 Out of 50 Samples

Google Safe Browsing 6 out of 50

Protection kicks in at 98 milliseconds instead of waiting more than a full day. Speed makes the difference clear when seconds count

6. DISCUSSION

Performance in real-world settings shows MalGuard connects classroom ideas to actual use. Results highlight: it works at scale, handles live data smoothly, runs efficiently on standard hardware, adapts quickly when conditions change, supports integration without major rework, maintains accuracy under pressure

1. Eighty-four percent beats twelve when spotting new threats - way ahead of old blacklist methods. Spotting zero-day attacks early changes how teams react to phishing. Blacklists miss most, but this catches way more. Fast detection matters just when it counts. Twelve percent looks weak next to that number. New exploits show up silent - this sees them anyway

2. Faster response times mean security keeps up without delays - built right into the browser. A 98-millisecond reaction window supports immediate threat detection

3. Balanced Precision with 3.2 Percent False Positive Rate Supports User Trust

4. Flask API scales for enterprise use

Though it runs only with a local API server now, later versions might handle remote hosting. Browser variety waits beyond today's setup.

7. Conclusion and Future Work

Out there among digital threats, one thing stands clear - MalGuard tackles harmful web links fast. Instead of waiting, it uses smart algorithms inside a browser add-on to catch dangers early. Behind the scenes runs a model shaped by more than two million examples of actual web addresses. Accuracy climbs to nearly 96 percent thanks to this blend of volume and method. High marks show up too in how well it predicts risky ones before they strike - AUC lands at 0.97. What helps most? Pulling apart words in domains while measuring term importance across samples. Even fresh attacks never seen before get flagged through these dual layers. Detection strength grows because patterns hide less when both signals work together. Fewer mistakes happen when structure meets frequency analysis head-on.

What stands out here is how the machine learning model runs live through a slim Flask-powered API paired with a Chrome extension built on Manifest V3. Running at speed, it sorts requests in about 98 milliseconds on average, catching harmful attempts directly in the browser nearly every time - hitting 99.2%. Users keep browsing smoothly, without hiccups, while staying shielded before threats land. New phishing pages never seen before? Spotted correctly almost 85% of the time. Older blocklist tactics fall short by comparison when faced with fast-changing attack styles.

Web browsers now host a tool called MalGuard, showing how classroom ideas can work in real-world security. Instead of heavy computing power, it runs lightweight ensemble models right where users browse. What stands out is its speed - detection happens fast enough to stop threats mid-action. Not every method scales like this inside browser tabs. Efficiency meets simplicity here, making strong protection more accessible. Tests confirm accuracy stays high even when hardware demands stay low.

Future Work

Even if the current setup works well, some upgrades might push it further. A web-hosted version could replace the need for personal servers, opening access on various gadgets. One next step involves building an online tool that sorts data without local hardware. Another path means adding compatibility with different internet programs like Firefox or Safari. This shift would let more people use it regardless of their device.

Because systems show why they make choices, people tend to believe them more. When live updates about new threats flow into the model, it spots fake messages better. Learning from fresh examples helps it stay sharp. Clear reasoning makes actions easier to follow. Constant data refreshes keep performance strong. Knowing the how and when of decisions changes how users see results.

One last thing - tweaking the model so it runs better on edge devices might help spot threats faster without slowing things down. With lighter versions of deep learning designs, speed stays high even when processing live data. Progress like this helps shape a security setup that adjusts quickly and covers more ground as risks evolve.

8. REFERENCES

Malicious URLs Dataset, Kaggle, 2025

MalGuard System Architecture, Project README, 2026

Scikit-learn RandomForest Documentation, 2025

Flask Microframework, MIT License, 2025

Chrome Extension Manifest V3, Google Developers, 2025

Malicious URL Detection using ML, IJACSA, 2020

Survey of URL Detection Techniques, arXiv:2504.16449, 2025