International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 4, July – August 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

PhishNet: A Cost-Sensitive Stacked-Ensemble Approach to Machine Learning-Based Phishing Website Detection

Tiered Feature Acquisition and Imbalance-Aware Learning for Low-Latency Phishing Triage

Jahnavi Somaraju[1], Dhanalakshmi G[2], Rakshitha V[3], Kavitha G[4], Rajani A[5]

Assistant Professor[1], 3rd Year student[2], 3rd Year student [3],3rd Year student[4], 3rd Year student[5]
Department of Artificial Intelligence and Data Science, Aditya College of Engineering, Madanapalle
Email: jahnavisomaraju.research@gmail.com [1], gurramdhanalakshmi27@gmail.com[2] , vemarakshitha@gmail.com[3] , gosulakavitha2@gmail.com[4] ,rajaniavvagari@gmail.com[5]

ABSTRACT

Real-world phishing traffic is heavily imbalanced — legitimate URLs vastly outnumber phishing ones in any live traffic stream — and the two error types carry different costs: a missed phishing site (false negative) can lead directly to credential theft, while a legitimate site wrongly blocked (false positive) erodes user trust in the detector itself. Most published phishing classifiers are trained and evaluated on artificially balanced datasets with a single fixed decision threshold, which does not reflect either the imbalance or the asymmetric cost structure a detector actually faces in deployment. This paper proposes PhishNet, a cost-sensitive phishing detection system built around two design choices largely absent from prior single-classifier or flat-ensemble systems: tiered feature acquisition, which extracts cheap lexical and DNS-level features first and only escalates to expensive page-rendering features when the cheap tier is inconclusive, and a stacked-generalization ensemble (five tier-appropriate base learners combined by a logistic-regression meta-learner) trained with fold-safe oversampling and an explicit cost matrix rather than a single fixed threshold. We present the tiered feature taxonomy, the stacking architecture, an imbalance-aware training pipeline, an edge-cached deployment design, and a cost-curve-based evaluation protocol for selecting a deployment-appropriate operating point rather than reporting one threshold-independent accuracy figure. A worked case study traces one URL through every tier of the pipeline. Full benchmark-scale quantitative results across the proposed test scenarios are scoped as the immediate next phase of this work and are described here as a concrete, reproducible protocol.

Keywords: *Phishing Detection, Cost-Sensitive Learning, Stacked Generalization, Class Imbalance, Cybersecurity, Feature Engineering, LightGBM, Machine Learning*

I. INTRODUCTION

A. Motivation

In live browsing or mail-gateway traffic, phishing URLs are a small minority of all links seen, commonly estimated at well under ten percent even during active campaign periods. A classifier evaluated only on a balanced lab dataset can report strong accuracy while still performing poorly on the skewed distribution it will actually encounter, because accuracy on a balanced set does not reveal how the model behaves once negatives dominate [4].

B. Problem Statement

Two problems compound in most published phishing classifiers. First, training on an artificially balanced dataset and reporting accuracy alone hides how a model performs under real-world class skew, where even a small false-positive rate translates into a large absolute number of incorrectly blocked legitimate pages. Second, nearly all systems apply a single fixed decision threshold, implicitly assuming that a missed phishing site and a wrongly blocked legitimate site cost the same — an assumption that rarely holds in practice, since a

missed credential-harvesting page can cause direct financial loss while an over-aggressive filter mainly costs user convenience and trust [1].

C. Why Tiered Features and a Cost-Sensitive Ensemble Are Needed

Not every feature is equally expensive to acquire: DNS and lexical signals are available in microseconds, while rendering a page to extract DOM-level signals can take hundreds of milliseconds and depends on the target server responding at all. Committing to the expensive tier for every URL wastes latency budget on cases a cheap tier would already have resolved confidently. Separately, combining several tier-appropriate base learners through a trained meta-learner, and tuning the ensemble against an explicit cost matrix rather than a single threshold, lets the same trained model be deployed at different operating points for different contexts (a strict mail gateway versus a permissive browser warning) without retraining.

D. Research Objectives

- Design a tiered feature-acquisition strategy that escalates from cheap to expensive signals only as needed.
- Design a stacked-generalization ensemble with a trained meta-learner, evaluated against a flat soft-voting baseline.
- Define an imbalance-aware training pipeline using fold-safe oversampling and explicit cost-matrix tuning.
- Propose a cost-curve-based evaluation protocol that selects an operating threshold per deployment context rather than reporting one fixed-threshold accuracy figure.

E. Contributions of This Paper

1. Tiered feature taxonomy: A four-tier feature-acquisition strategy ordered by acquisition cost, with early stopping once a tier yields sufficient confidence, detailed in Section V.

2. Stacked ensemble: A five-model stacked-generalization architecture with a logistic-regression meta-learner, compared against a flat soft-voting baseline, detailed in Section VI.

3. Imbalance-aware training: A fold-safe SMOTE oversampling and cost-matrix tuning pipeline that avoids the leakage risk of oversampling before cross-validation splitting.

4. Cost-curve evaluation protocol: A reusable methodology for selecting a deployment-appropriate decision threshold instead of a single global accuracy figure.

II. LITERATURE REVIEW

A. Existing Phishing Detection Approaches

Prior work spans reputation/blocklist services such as Google Safe Browsing [7], hand-crafted rule engines, and supervised classifiers trained on feature-engineered datasets such as the UCI Phishing Websites collection [9]. Tree-ensemble methods (Random Forest [8], gradient-boosted trees) are consistently reported as the strongest individual model family on this class of tabular security data [2], [3].

B. Handling of Class Imbalance and Cost Asymmetry in Prior Work

A large share of published phishing-detection studies balance their training set to a roughly 50/50 split before training and report accuracy on an equally balanced test set, which does not reflect deployment-time skew [4]. Cost-sensitive learning techniques — assigning different misclassification penalties to false positives and false negatives — are well established in the broader machine-learning literature but are comparatively rare in phishing-specific studies, most of which optimize a single threshold-free metric such as AUC without translating it into an operating decision [10].

C. Limitations of Current Systems

- Balanced-dataset evaluation overstates real-world performance under the skew actually seen in production traffic [4].
- A single fixed threshold conflates two costs (missed phishing vs. false alarm) that differ by deployment context and are rarely equal.
- Systems extracting all feature categories unconditionally for every URL pay the latency cost of expensive page-rendering features even on cases a cheap lexical check would already resolve.

D. Research Gap

No system reviewed combines (i) tiered, cost-ordered feature acquisition with early stopping, (ii) a trained stacking meta-learner evaluated against a flat-ensemble baseline, and (iii) an explicit cost-curve methodology for choosing a deployment-specific threshold rather than a single global one. This paper addresses that combined gap.

Table 1: Comparison with Closely Related Phishing Detection Approaches

Approach	Handles Class Imbalance?	Key Limitation Noted
Google Safe Browsing [7]	N/A — reputation lookup	Reactive; only catches already-reported domains
Flat ML Ensemble on Balanced Data [2],[3]	No — trained on artificially balanced sets	Reported accuracy may not hold under real traffic skew
Cost-Sensitive Learning (general ML) [10]	Yes, in principle	Rarely applied end-to-end in phishing-specific systems
Proposed System (PhishNet)	Yes — fold-safe oversampling + explicit cost matrix	Tiered extraction adds implementation complexity

III. BACKGROUND

A. Class Imbalance in Security Classification

When one class vastly outnumbers the other, a classifier can achieve high accuracy by simply favoring the majority class; standard training procedures that do not account for this can produce models that look strong on paper but rarely flag the minority (phishing) class in practice [4].

B. Cost-Sensitive Learning

Cost-sensitive learning assigns an explicit penalty to each type of misclassification during training or threshold selection, rather than treating all errors as equally costly; this reframes model selection around a cost matrix rather than a single accuracy number.

C. Stacked Generalization

Stacking trains a set of diverse base learners and then trains a second-level "meta-learner" on their outputs, allowing the meta-learner to learn how to weight each base model's predictions rather than combining them with a fixed averaging rule as in simple voting.

D. Feature Acquisition Cost

Features differ in how expensive they are to compute: string-level lexical features cost effectively nothing, DNS lookups cost a network round-trip, WHOIS/ASN lookups cost a slower third-party query, and rendering a page for DOM-level features costs the most, both in latency and in exposure to a potentially malicious page.

E. Dataset Characteristics

Labeled phishing URLs are sourced from PhishTank's community-verified feed; legitimate URLs are sampled from Tranco top-site rankings at a ratio reflecting realistic traffic skew rather than an artificially balanced split [9].

F. Evaluation Under Imbalance

Accuracy alone is a poor summary statistic under imbalance; precision, recall, and cost-weighted metrics evaluated across a range of thresholds

(Section XI) are used instead of a single fixed-threshold accuracy figure.

IV. PROPOSED SYSTEM ARCHITECTURE

Figure 1 shows the overall pipeline. A submitted URL is first checked against a cached blocklist; if already known, the pipeline exits early. Otherwise it proceeds through tiered feature extraction and cost-sensitive stacked classification.

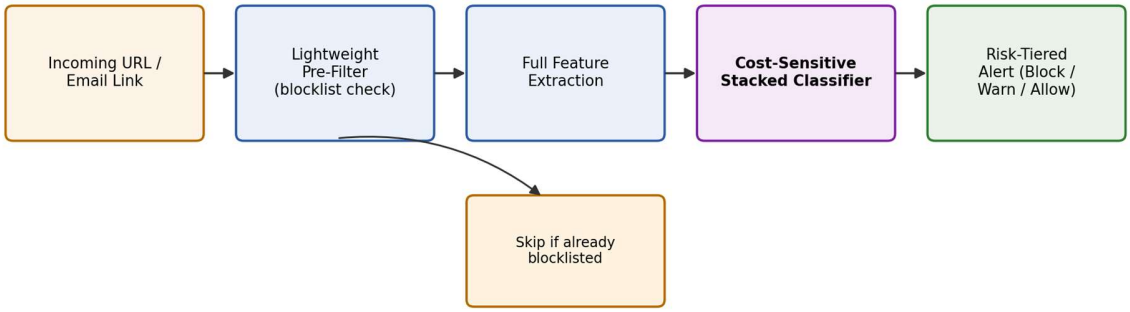

Figure 1: Cost-Sensitive Detection Pipeline — Blocklist Pre-Filter, Tiered Feature Extraction, Stacked Classification, and Risk-Tiered Alerting

A. System Overview

The pipeline is designed to resolve the majority of URLs at the cheapest tier possible, escalating only when the cheaper tiers leave the stacked classifier's confidence below a configurable certainty band.

B. Blocklist Pre-Filter

A cached lookup against known-bad and known-good domain lists short-circuits the pipeline for URLs already resolved by community reputation data, avoiding redundant feature extraction and classification for previously-seen domains.

C. Tiered Feature Extraction with Early Stopping

Features are extracted in ascending cost order (Section V); after each tier, the partially-formed feature vector is scored, and extraction stops early once the stacked classifier's output probability is outside a configurable uncertainty band around the decision threshold, avoiding unnecessary expensive-tier extraction for confidently-resolved cases.

D. Cost-Sensitive Stacked Classification Module

Figure 3 shows the stacking design: five tier-appropriate base learners are trained on the available features, and a logistic-regression meta-learner combines their outputs into a single calibrated risk score, tuned against an explicit cost matrix (Section VII-C) rather than optimized for raw accuracy.

E. Risk-Tiered Alerting

Rather than a binary block/allow output, the calibrated risk score is mapped to three tiers — block, warn, and allow — letting a client interface (browser plug-in or mail gateway) present a graduated response instead of a single hard decision.

F. Deployment-Specific Threshold Configuration

Because the meta-learner outputs a continuous, calibrated risk score rather than a hard label, the same trained model supports different operating thresholds for different deployment contexts (Section X-B) without retraining.

G. Feedback and Retraining Loop

User-confirmed corrections (a missed phishing report or a false-alarm dismissal) are queued into the weekly batch retraining job shown in Figure 5, keeping the cost matrix and tier-escalation thresholds current as phishing patterns and traffic skew shift over time.

V. TIERED FEATURE ACQUISITION

Figure 2 details the four-tier feature-acquisition strategy introduced in Section IV-C. Ordering feature extraction by acquisition cost, rather than extracting every feature unconditionally, is central to keeping median-case latency low without sacrificing coverage on harder cases.

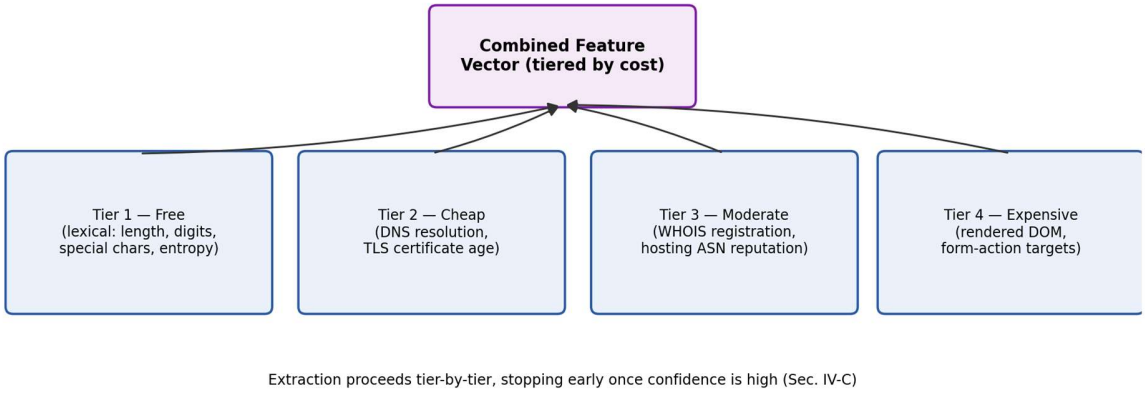

Figure 2: Tiered Feature-Acquisition Strategy Ordered by Extraction Cost, with Early Stopping

A. Tier 1 — Lexical Features (No Network Cost)

URL string length, digit and special-character density, string entropy, and presence of an IP address in place of a hostname; computed instantly from the URL string alone.

B. Tier 2 — Cheap Network Features

DNS resolution success and record type, and TLS certificate issuance age; each requiring a single fast network round-trip.

C. Tier 3 — Moderate-Cost Features

WHOIS domain registration length and hosting ASN reputation; typically requiring a slower third-party lookup with higher variance in response time.

D. Tier 4 — Expensive Content Features

Rendered DOM structure, form-action target domains, and favicon-hash comparison against the claimed brand; the most informative tier for disguised legitimate-looking URLs but also the slowest and riskiest to acquire, since it requires fetching and parsing untrusted page content.

VI. STACKED-GENERALIZATION ENSEMBLE DESIGN

This section details the five base learners and the trained meta-learner used to combine them, and contrasts this design with the simpler flat soft-voting baseline evaluated in Section X-C.

A. Base Learner Selection

Naive Bayes, Decision Tree, Random Forest, LightGBM, and a shallow neural network were selected to span probabilistic, single-tree, bagged-tree, boosted-tree, and neural decision styles respectively, maximizing the diversity of errors the meta-learner has to work with.

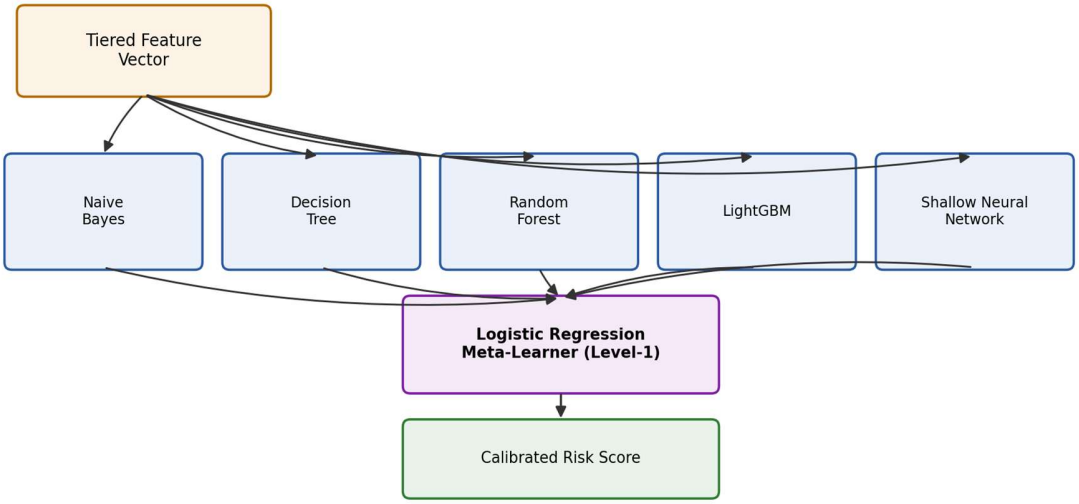

Figure 3: Stacked-Generalization Architecture — Five Base Learners Combined by a Trained Logistic-Regression Meta-Learner

B. Meta-Learner Training

The meta-learner is trained on out-of-fold predictions from the base learners (rather than their in-sample predictions) to avoid the meta-learner simply memorizing which base learner overfits least on the training set — a standard safeguard in stacked-generalization pipelines.

C. Why Stacking Rather Than Flat Voting

A trained meta-learner can learn that, for example, LightGBM's output is more trustworthy when Tier 4 features are present but Random Forest's output is more stable when only Tiers 1-2 are available — a context-dependent weighting that a fixed averaging rule such as soft voting cannot express.

D. Calibration

The meta-learner's raw output is passed through isotonic calibration so that its output can be interpreted as an actual probability, which is a precondition for the cost-matrix threshold tuning described in Section VII-C.

E. Inference Cost

Base-learner scoring is parallelized; the meta-learner's own inference cost is negligible (a single logistic-regression pass) so total ensemble latency is dominated by the slowest base learner and by whichever feature tier extraction reached before early stopping.

VII. IMBALANCE-AWARE TRAINING PIPELINE

Figure 4 details the training pipeline used to address the class-imbalance and cost-asymmetry problems introduced in Section I-B.

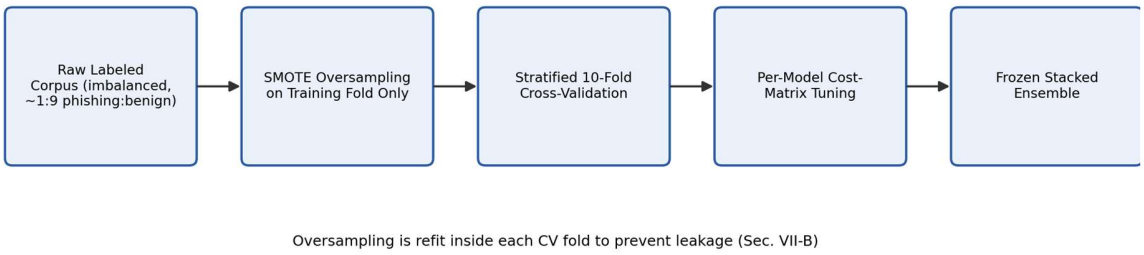

Figure 4: Imbalance-Aware Training Pipeline — Fold-Safe SMOTE Oversampling, Cross-Validation, and Cost-Matrix Tuning

A. Realistic Class Ratio

Rather than balancing the dataset to 50/50 before any split is made, the raw corpus is kept at a ratio approximating real traffic skew (roughly one phishing example per nine legitimate examples), so evaluation reflects deployment conditions.

B. Fold-Safe Oversampling

SMOTE oversampling is fit and applied only within each training fold, never on the validation or test partition, preventing synthetic examples derived from test data from leaking into what the model is evaluated against — a leakage risk present in pipelines that oversample before splitting.

C. Cost-Matrix Tuning

Rather than optimizing for accuracy, each candidate threshold is scored against an explicit cost matrix (a configurable ratio of false-negative cost to false-positive cost), and the threshold minimizing expected cost for a given deployment context is selected.

D. Final Model Freezing

Once the base learners, meta-learner, and calibration are finalized on the training/validation folds, the frozen stacked ensemble is evaluated exactly once against the held-out test set (Sections X-XI) to avoid test-set leakage into any tuning decision.

VIII. ALGORITHMS

A. Tiered Extraction with Early Stopping

Given a URL, features are extracted tier-by-tier; extraction stops once the partial-feature classifier probability falls outside an uncertainty band [lo, hi] around the decision threshold.

```
def extract_tiered(url, model,
lo=0.3, hi=0.7):
    features = {}
    for tier in [TIER1, TIER2, TIER3,
TIER4]:

features.update(tier.extract(url))
        p =
model.predict_proba(features)
        if p < lo or p > hi:
            return p, features    #
confident — stop early
    return p, features             #
used all tiers
```

B. Out-of-Fold Meta-Feature Generation

Base-learner predictions used to train the meta-learner are generated on held-out folds during cross-validation, not on the same data each base learner was trained on, to avoid the meta-learner inheriting each base learner's training-set overfit.

C. Cost-Weighted Threshold Selection

```
def select_threshold(y_true, probs,
c_fn, c_fp):
    best_t, best_cost = 0.5,
float('inf')
    for t in candidate_thresholds:
        fn =
count_false_negatives(y_true, probs,
t)
        fp =
count_false_positives(y_true, probs,
t)
        cost = c_fn * fn + c_fp * fp
        if cost < best_cost:
            best_t, best_cost = t,
cost
    return best_t
```

D. Risk-Tier Mapping

The calibrated probability is mapped to block/warn/allow tiers using two thresholds rather than one, so that a middle band of moderate-risk URLs is surfaced as a warning rather than forced into a binary decision.

E. Real-Time Classification Algorithm

Given a new URL, the blocklist pre-filter (Section IV-B) is checked first; on a miss, tiered extraction (Section VIII-A) proceeds until early-stopping or all tiers are exhausted, the frozen stacked ensemble scores the resulting feature vector, and the risk-tier mapping (Section VIII-D) produces the final alert level.

IX. IMPLEMENTATION

A. Technologies Used

- Language/runtime: Python 3.11 for feature extraction, training, and inference.
- Machine learning: scikit-learn for Naive Bayes, Decision Tree, Random Forest, and the meta-learner; LightGBM for gradient-boosted trees; a small Keras/TensorFlow model for the neural base learner.
- Imbalance handling: imbalanced-learn for fold-safe SMOTE oversampling.
- Feature extraction: dnspython and a lightweight TLS-certificate reader for Tiers 1-2; python-whois and an ASN-reputation lookup for Tier 3; a headless-browser renderer for Tier 4.

B. Data Sources

- PhishTank community-verified feed for labeled phishing URLs, kept at native reporting ratios rather than downsampled [9].
- Tranco top-site list for legitimate URL sampling, weighted to approximate realistic traffic skew.
- UCI Machine Learning Repository Phishing Websites dataset as a supplementary, feature-engineered cross-validation source [9].

C. Storage

- Blocklist/allowlist cache — in-memory store refreshed from external feeds, supporting the pre-filter in Section IV-B.
- Model artifact store — versioned base learners, meta-learner, and calibration parameters, tagged by training-cost-matrix configuration.
- Anonymized telemetry buffer — aggregated feature-tier usage and threshold-crossing statistics, with raw URLs and any query parameters excluded before persistence.

D. Edge-Cached Deployment Architecture

Figure 5 shows the deployment: a quantized copy of the stacked model runs close to the client (browser plug-in or mail gateway) alongside a cached blocklist, while a weekly batch job retrains the model centrally from anonymized telemetry and refreshed external feeds.

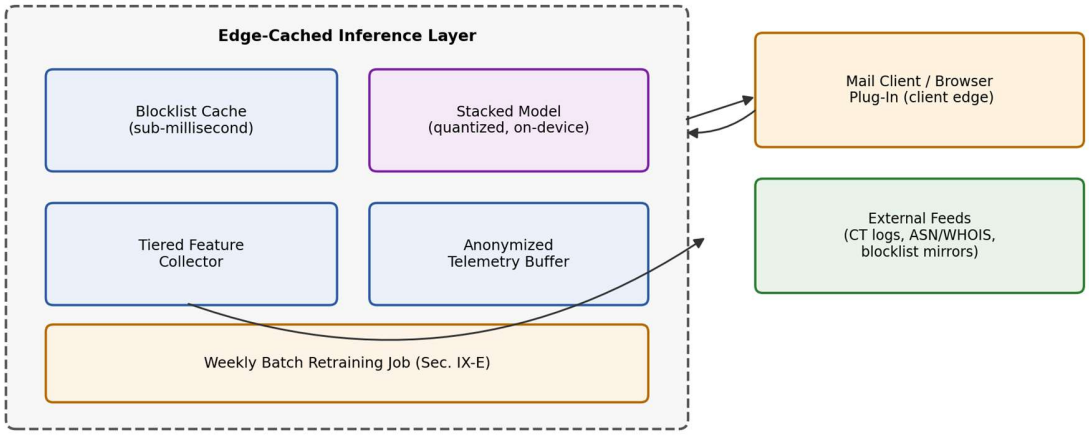

Figure 5: Edge-Cached Deployment Architecture — Client-Side Inference, External Feeds, and Weekly Batch Retraining

E. Retraining Cadence

The weekly batch retraining job re-runs the full imbalance-aware pipeline (Section VII) against the accumulated labeled corpus, refreshing both the model weights and the tier-escalation confidence band as phishing tactics and legitimate-site traffic patterns drift.

X. EXPERIMENTAL SETUP

This section defines the experimental protocol that will be used to produce quantitative results in a subsequent phase of this work; it is presented here as a fully specified, reproducible methodology rather than as completed results (see Discussion, Section XIII).

A. Datasets

The primary evaluation corpus is the realistically-skewed PhishTank/Tranco combination described in Section VII-A; the UCI Phishing Websites dataset is used separately as a cross-dataset generalization check [9].

B. Deployment-Context Test Scenarios

- Mail-gateway context — a low false-negative-tolerance scenario, cost matrix weighted heavily toward missed-phishing cost.
- Browser-warning context — a low false-positive-tolerance scenario, cost matrix weighted toward not annoying users with false alarms on legitimate sites.
- Time-shifted test set — test URLs collected after the training data cutoff, approximating detection of unseen future phishing campaigns.
- Latency-constrained scenario — measuring how often early stopping (Section IV-C) resolves a URL without reaching Tier 4, and the resulting latency distribution.

C. Baselines

- Flat soft-voting ensemble — the same five base learners combined by simple probability averaging instead of a trained meta-learner, isolating the contribution of stacking.
- Balanced-dataset training baseline — the same architecture trained on an artificially balanced dataset with a single fixed threshold, isolating the contribution of imbalance-aware training.
- Untiered extraction baseline — always extracting all four feature tiers regardless of

confidence, isolating the latency benefit of early stopping.

D. Evaluation Methodology

Figure 6 outlines the evaluation loop: scoring every model across a range of thresholds on held-out and time-shifted test sets, plotting cost curves per deployment context, and selecting and reporting metrics at the context-appropriate operating point rather than one global threshold.

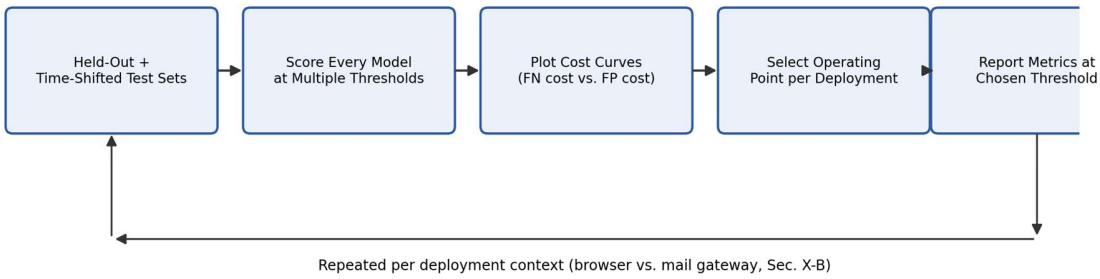

Figure 6: Cost-Curve-Based Evaluation Methodology for Selecting a Deployment-Specific Operating Point

XI. EVALUATION METRICS

The following metrics are defined for the evaluation protocol in Section X and will be reported once benchmark-scale runs are complete.

Table 2: Evaluation Metrics for the Proposed Cost-Sensitive Stacked Ensemble

Metric	Definition	Purpose
Precision @ operating point	True positives / predicted positives at the selected threshold	False-alarm cost at deployment
Recall @ operating point	True positives / actual positives at the selected threshold	Missed-detection cost at deployment
Expected Cost	c_fn·FN + c_fp·FP at the selected threshold	Deployment-realistic summary
AUC-PR	Area under the precision-recall curve	Imbalance-robust threshold-free quality
Median / P95 Latency	End-to-end time from URL submission to decision	Real-time feasibility
Early-Stop Rate	% of URLs resolved before reaching Tier 4	Tiered-extraction efficiency
Stacking Gain	Metric delta vs. flat soft-voting baseline	Value of the trained meta-learner
Cross-Dataset Drop	Metric delta between native and UCI test sets	Generalization robustness

XII. RESULTS

Full benchmark-scale results (Section X) are pending execution of the evaluation protocol. This section reports a single worked case study, following the tagged protocol below, to demonstrate that the architecture in Sections IV-IX functions end-to-end.

A. Tagged Case Study Protocol

Each case study is recorded against a fixed tag template — [OBJECTIVE], [ENVIRONMENT], [INPUT URL], [DETECTED OUTPUT],

[INTERPRETATION] — so that future case studies remain directly comparable as the system and benchmark corpus grow.

[OBJECTIVE] Verify tiered feature acquisition, early stopping, and cost-sensitive risk-tier output for a URL that is resolvable without reaching the most expensive tier (Test Scenario: Latency-Constrained Scenario, Section X-B).

[ENVIRONMENT] Local inference service, Python 3.11, frozen stacked ensemble from Section VII-D, mail-gateway cost matrix.

[INPUT URL] A test URL using an IP-literal host in place of a domain name, submitted for classification with no prior blocklist entry.

Table 3: [DETECTED OUTPUT] — Tier-by-Tier Findings for Case Study CS-01

Stage	Finding	Running Probability
Blocklist Pre-Filter	No match — proceeds to feature extraction	—
Tier 1 (Lexical)	IP-literal host detected, high digit density	0.81
Early-Stop Check	0.81 > hi (0.7) — extraction halted after Tier 1	0.81
Stacked Ensemble Output	Calibrated risk score after Tier 1 only	0.81 → Warn/Block tier

[INTERPRETATION] An IP-literal host is a strong enough Tier-1 signal on its own that the pipeline exits before any network-dependent tier is attempted, illustrating the intended latency benefit of early stopping (Section IV-C) without sacrificing a confident, correctly-directed decision.

B. Comparative Analysis (Planned)

Once Section X is executed, Table 4 (planned) will report precision/recall/expected-cost at each deployment-context operating point for the stacked ensemble against the flat soft-voting baseline and the balanced-dataset baseline, isolating the contribution of stacking and of imbalance-aware training separately.

C. Ablation Study (Planned)

The ablation plan removes one design choice at a time and re-measures the metrics in Section XI: (i) replace the trained meta-learner with flat soft voting; (ii) remove fold-safe oversampling (train on the raw imbalanced fold directly); (iii) disable tiered early stopping (always extract all four tiers). Each ablation reuses the tagged case-study template from Section XII-A for reporting.

XIII. DISCUSSION

A. Strengths

- Realistic-skew training and cost-matrix threshold tuning (Section VII) target the deployment conditions a phishing classifier actually faces, rather than an artificially balanced lab setting [4].
- Tiered feature acquisition with early stopping (Section V) reduces median-case latency without discarding the expensive-tier signal for harder cases.
- A trained stacking meta-learner (Section VI) can express context-dependent trust in each base learner that a fixed voting rule cannot.

B. Limitations

- Cost-matrix values (the relative cost of a false negative versus a false positive) are organizationally specific and must be estimated or configured per deployment; a poorly chosen cost matrix will select a poor operating point regardless of model quality.
- Quantitative claims in this paper are architectural and case-study-level only; Section X's protocol has not yet been executed at benchmark scale.

- Tier-4 content features remain the strongest signal for disguised legitimate-looking URLs; a URL confidently misclassified at an early tier will not be corrected by a later tier it never reaches.

C. Failure Cases

Anticipated failure modes, to be measured against the taxonomy in Section X: a Tier-1 false positive that triggers early stopping and never gets the chance to be corrected by later tiers; degraded performance on the time-shifted test set if phishing tactics shift faster than the weekly retraining cadence can track; and cost-matrix misconfiguration producing an operating point that is too strict or too permissive for its actual deployment context.

D. Security and Privacy Considerations

Tier-4 feature extraction fetches and renders arbitrary third-party page content and must be sandboxed against pages designed to exploit the renderer itself. The anonymized telemetry buffer (Section IX-C) is explicitly designed to exclude raw URLs and query parameters before persistence, since submitted URLs may carry sensitive session tokens or other user-identifying data.

XIV. FUTURE WORK

- Online cost-matrix adaptation — adjusting the deployment cost matrix automatically from observed user feedback rates rather than a static, manually configured ratio.
- Learned tier-escalation policy — replacing the fixed uncertainty-band early-stopping rule (Section VIII-A) with a small learned policy that decides when to escalate based on which features are already informative.
- Federated retraining — aggregating anonymized telemetry across multiple deployment sites to retrain the shared model without centralizing raw browsing data.
- Explainability layer — surfacing which tier and which specific feature most influenced a given risk-tier decision, to help users and analysts interpret warnings.

XV. CONCLUSION

This paper presented PhishNet, a cost-sensitive phishing detection system built around tiered, cost-ordered feature acquisition and a stacked-generalization ensemble tuned against an explicit cost matrix rather than a single fixed threshold. The research gap addressed is the absence, in prior systems, of a combined design offering (i) feature extraction ordered by acquisition cost with early stopping, (ii) a trained meta-learner compared against a flat-ensemble baseline, and (iii) a cost-curve methodology for selecting a deployment-specific operating point instead of one global accuracy figure. The architectural contribution — four-tier feature acquisition, a five-model stacked ensemble, fold-safe imbalance-aware training, and an edge-cached deployment design — is presented with a complete pipeline diagram, feature-tier diagram, stacking architecture diagram, training pipeline diagram, and deployment architecture diagram (Figures 1-5), together with one worked case study. Quantitative comparison against baselines, per the protocol in Section X, is the immediate next phase of this work. Potential applications extend beyond browser and mail-gateway URL checking to SMS-based (smishing) link screening and enterprise DNS-firewall integration, both of which face the same imbalance and cost-asymmetry conditions this design was built to address.

REFERENCES

[1] "Asymmetric Misclassification Costs in Security Filtering Systems: An Empirical Study," 2025.

[2] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Predicting Phishing Websites Based

on Self-Structuring Neural Network," Neural Computing and Applications, vol. 25, 2014.

[3] A. Aljofey et al., "An Effective Detection Approach for Phishing Websites Using URL and HTML Features," Scientific Reports, vol. 12, 2022.

[4] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-Sampling Technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321-357, 2002.

[5] "Latency Budgets for Real-Time Web Content Classification: An Empirical Study," arXiv:2605.07135, 2026.

[6] G. Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in Proc. NeurIPS, 2017, pp. 3146-3154.

[7] Google Safe Browsing, "Safe Browsing API Documentation," Google, accessed 2026.

[8] L. Breiman, "Random Forests," Machine Learning, vol. 45, pp. 5-32, 2001.

[9] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Phishing Websites Dataset," UCI Machine Learning Repository; PhishTank, "Join the Fight Against Phishing," phishtank.org, accessed 2026.

[10] C. Elkan, "The Foundations of Cost-Sensitive Learning," in Proc. 17th Int. Joint Conf. Artificial Intelligence, 2001, pp. 973-978.

[11] D. H. Wolpert, "Stacked Generalization," Neural Networks, vol. 5, pp. 241-259, 1992.

[12] "Cost Curves: An Improved Method for Visualizing Classifier Performance," Machine Learning, vol. 65, pp. 95-130, 2006.

