TinyRod def/acc Hackathon report[1]

Matthew Bream
matthew.bream@logiq.co.uk
Logiq

William Hayden
will.hayden@logiq.co.uk
Logiq

Harry Waterman
harry.waterman@logiq.co.uk
Logiq

With
Apart Research

Abstract

TinyRod presents a browser-based, privacy-preserving approach to phishing email detection using Small Language Models (SLMs) running locally on user devices. As AI-enabled phishing campaigns become increasingly sophisticated and scalable, traditional detection methods that require sending email content to centralised, often external services create privacy concerns and single points of failure. TinyRod addresses this by performing real-time phishing analysis directly in the browser. The system combines on-device SLM inference with a graph-based analysis platform (Harbour) that aggregates metadata to identify campaign patterns across organisations.
Our evaluation demonstrates that SLMs as small as 4 billion parameters can effectively detect phishing emails when guided by carefully engineered system prompts, achieving 86% accuracy on a balanced 400-email test set drawn from a multi-source phishing corpus. The system provides explainable feedback by highlighting specific red flags directly in the email interface, improving user understanding and behavior. Harbours Neo4j graph database enables security teams to identify patterns such as shared malicious domains and recurring attack vectors that are difficult to detect in traditional log-based systems.
This work contributes to democratising access to AI-powered security tools by demonstrating that effective phishing detection can run entirely on consumer hardware without compromising privacy. The architecture balances local privacy-preserving analysis with the potential for organisational threat intelligence, supporting both individual users and enterprise security teams.

This project was developed as part of the Defensive Acceleration (def/acc) Hackathon (November 21-23, 2025) organised by Apart Research. We focus on "strengthening the shield" by building better defensive technology (Apart Research, 2025).

Keywords: Phishing detection, Small Language Models, Privacy-preserving security, Browser-based AI, Graph databases, Edge computing

[1] Research conducted at the Defensive Acceleration Hackathon, 2025

1. Introduction

One might reasonably ask: isn't phishing a "solved" problem by now? After all, technical defences have matured considerably. Multiple studies of machine-learning (ML) models for phishing URL or website detection report accuracy well above 95%. For example, a survey of ML-based phishing website detection methods noted that "various machine learning-based solutions achieved higher than 95% accuracy" (Tang and Mahmoud). Some individual experiments pushed accuracy to 98–99% on curated datasets (N et al.).

Yet in practice, phishing remains the number one path cyber-criminals use. Attack volumes continue to rise, adversaries keep innovating, and human users remain vulnerable. Why the disconnect? Because high accuracy on benchmark datasets doesn't necessarily translate into resilient, real-world protection. The models may rely on features or patterns that attackers quickly defeat or bypass; they may not handle zero-day or adversarial-crafted lures; and they often assume centralised infrastructure, user consent, or benign data flows. Phishing has evolved from crude, typo-filled scams into a highly scalable, AI-enabled threat that is increasingly difficult for humans to spot. Microsoft's Digital Defense Report 2025 (Microsoft) describes how adversaries are already using generative AI to scale social engineering, automate aspects of credential theft, and make phishing campaigns more targeted and convincing at speed. In parallel, the World Economic Forum's Global Cybersecurity Outlook 2025 (World Economic Forum) warns that emerging technologies, including generative AI, are exacerbating long-standing cyber-resilience challenges and widening the gap between well-resourced organisations and those that struggle to maintain even "minimum viable" cyber resilience. The near-term impact of AI on the cyber threat (National Cyber Security Centre) further emphasises that generative AI lowers the barrier to entry for less skilled attackers and enables more polished, personalised phishing content that can closely mimic legitimate communications.

In other words: the fact that models can score 95 %+ does not mean the problem is solved. The underlying threat landscape evolves, and adversaries adapt faster than many detection functions. Hence the enduring dominance of phishing as a cybercriminal vector: the technical bar has been raised, but the operational and behavioural bar remains far lower.

Whilst LLM Detection and LLM-as-a-judge systems are not considered to be the most accurate, they do however present an interesting paradigm shift, as the size of model required to write a very convincing phishing email far outweighs the size of model required to detect a phishing email based on common indicators, such as suspicious sender addresses that don't match the display name, or suspicious URLs with only minor differences from very popular websites.

Against this backdrop, our project - TinyRod - asks what effective, privacy-preserving defences against AI-enabled phishing might look like when built directly into the user's browser and backed by graph-based analysis.

This approach has the potential to allow almost everyone who must use email to be a functioning member of society to be protected by an AI-assisted shield against the most common cyber attack in the world, democratising access to security for everyone.

RQ1: Can we detect phishing effectively at the edge, without centralising email content?

H1: A browser-based, on-device detection pipeline (TinyRod) that combines simple heuristics with a SLM (Small Language Model - which is defined as a 1 million to 10 billion parameter model by HuggingFace) can achieve useful phishing / suspicious / benign classifications without sending raw email bodies to a third party service.

RQ2: Does structuring detection data as a graph improve insight into phishing campaigns?
H2: Modelling emails, senders, recipients, URLs and domains as a Neo4j graph in Harbour will surface patterns (e.g. shared malicious domains, recurring lures) that are potentially difficult to see in traditional log-style dashboards.

RQ3: Can we give users explainable, actionable feedback rather than a black-box "this is bad" warning?
H3: Highlighting concrete "red flags" (urgent language, credential requests, mismatched domains, risky URLs) and showing them directly in the email context will make the detection more understandable and more likely to influence user behaviour, reducing the success rate of future Phishing campaigns through user awareness.

RQ4: Is it viable to run these models locally on-device for most people in 2025? And are the models that do run actually useful?

H4: The curve for what SLMs are capable of compared to even 12 months ago is incredible, newer open source SLMs are able to return coherent JSON allowing for functional tool calling and agentic behaviour even as low as models with 0.6B parameters, how far can this behaviour be pushed whilst maintaining accuracy?

RQ5: Is it possible to balance privacy, usability, and security in a single design?
H5: By doing rich analysis locally and only sending structured, minimised metadata to Harbour, we can meaningfully reduce privacy risk while still giving security teams and researchers the aggregate signal they need.

RQ6: Can SLMs provide a sustainable, privacy-preserving alternative to the escalating compute demands of modern AI systems and models?

H6: Small Language Models running on-device can deliver surprising results while using dramatically less compute, energy, and infrastructure than cloud-scale AI systems, highlighting that you don't need the biggest model for most use cases.

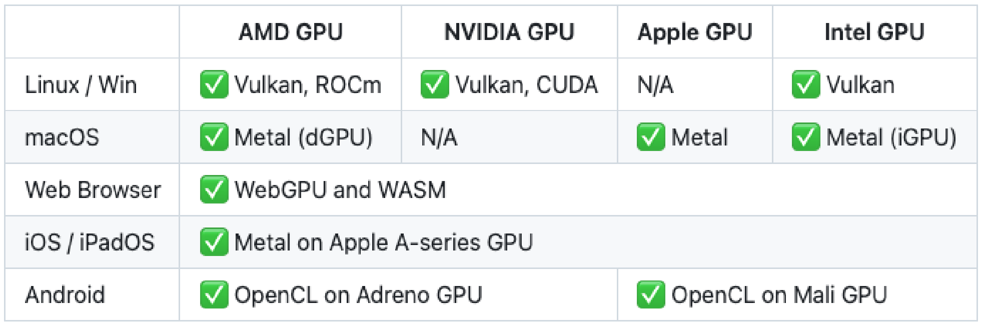
The project aligns with the def/acc hackathons focus on "Cybersecurity & Infrastructure Protection" (Apart Research, 2025). As noted, "AI is fundamentally changing what's possible for both attackers and defenders," and TinyRod represents an effort to ensure defensive capabilities keep pace with offensive ones. The project demonstrates that modern SLMs, when properly configured, can provide effective security analysis even at the edge, helping to close the asymmetry between attack and defense capabilities.

Category	Details
Scope	Proof-of-concept. Users can modify the extension locally; the threat-model only covers the default, unmodified TinyRod and Harbour.
Assets We Care About	• User accounts & credentials (email, SaaS, banking) • User decision-making about links/attachments/credential entry • Aggregate phishing telemetry stored in Harbour (domains, URLs, flags) • User privacy (email content must stay local) • Users' ability to use email services normally
Adversary	External attacker sending phishing emails via Gmail/Outlook. Uses AI-generated content, malicious domains, reused templates, tracking pixels, obfuscated URLs. No initial control of the user's device/browser/network.
Adversary Capabilities	• Deliver arbitrary content via email (text/HTML/images/links) • Adapt lures over time • Run large-scale campaigns • Evade simple heuristics (keyword dodging, URL obfuscation)
Adversary Goals	• Steal credentials for account takeover • Steal money through fraud • Install malware via attachments/drive-by links • Harvest personal info for further attacks • Gain persistent access to wider systems
Adversary Does Not Have	• Control of user browser, OS, or network • Ability to tamper with TinyRod code • Direct access to Harbour backend or Neo4j
Defender Capabilities (TinyRod)	• Read visible email content on the web page • Apply rule-based heuristics • Run local LLM analysis • Show verdict, score, and red flags to user • Send minimal structured metadata to Harbour (optional)

Defender Capabilities (Harbour)	• Ingest events across many clients • Store email/domain/URL graph data • Detect campaign patterns and infrastructure reuse • Provide dashboards for security/IT teams
Effects on Protected Assets	**User Privacy:** No email content saved; telemetry optional; only sender/recipient collected when enabled. **User Access:** No impact on ability to use Outlook; TinyRod is passive and UI-integrated.
Security Goals	• Reduce successful phishing (nudges, visibility of AI-written scams) • Provide explainable detection (human-readable red flags) • Enable privacy-preserving analytics • Improve defender situational awareness (campaign patterns)
Non-Goals / Out of Scope	• Already-compromised user devices/accounts • Malicious or tampered TinyRod builds • Mail transport/server security (e.g., SMTP protection) • Post-compromise attacker actions (BEC, lateral movement) • Threat-actor attribution
Threats to TinyRod & Harbour	**Injection into Harbour:** Mitigated with strict JSON schema; only metadata sent. **Prompt Injection:**Attackers email themselves crafted prompts to manipulate model output. **Adversarial Evasion:**Carefully crafted emails fool SLM classification. **DoS against Harbour:** Flooding API; mitigated in full version via rate-limits & API keys. **Model Poisoning:** Corrupting datasets to bias model choice; mitigated via multiple datasets and no fine-tuning.
Overall Summary	External AI-enabled phishers try to trick users via webmail. TinyRod sits in-browser to detect and explain risky emails; Harbour aggregates signals to reveal campaign patterns. Assumes an uncompromised client and privacy-preserving telemetry pipeline.

2. Methods

TinyRod is built using a combination of modern web technologies and open-source machine learning frameworks. The core inference engine leverages WebLLM, built on top of MLC-LLM (Machine Learning Compilation for Large Language Models) (Feng), which enables native LLM inference across a wide range of platforms without requiring specialised drivers or installations. This approach allows the extension to run transformer-based models directly in the browser and was chosen specifically to explore what "on-device" phishing defence could look like without relying on any closed-source cloud APIs such as OpenAI or Anthropic.



	AMD GPU	NVIDIA GPU	Apple GPU	Intel GPU
Linux / Win	✅ Vulkan, ROCm	✅ Vulkan, CUDA	N/A	✅ Vulkan
macOS	✅ Metal (dGPU)	N/A	✅ Metal	✅ Metal (iGPU)
Web Browser	✅ WebGPU and WASM			
iOS / iPadOS	✅ Metal on Apple A-series GPU			
Android	✅ OpenCL on Adreno GPU		✅ OpenCL on Mali GPU	

MLC-LLM Supported Platforms

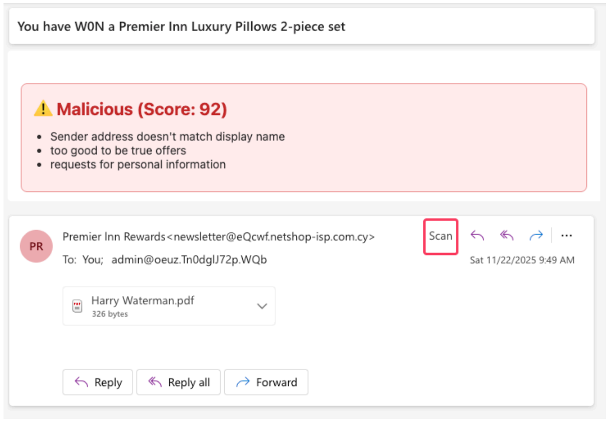
The project codebase is organised into three main components in our GitHub repository: the Chrome extension (TinyRod), the evaluation and benchmarking harness (TestRig), and the central reporting and graph analytics backend (Harbour). TinyRod contains the browser integration, DOM extraction logic, and WebLLM prompt/response handling. TestRig is a lightweight framework we used to replay emails from our dataset of choice against different models or the same model with different parameter values, compare outputs, and generate basic metrics. Harbour consists of a Node/Express API plus a Neo4j graph database and NeoDash dashboards, used to store structured events (emails, senders, domains, URLs, flags) and explore campaign-level patterns.

On the modelling side, we deliberately did not train or fine-tune our own model during the hackathon. Instead, we experimented with a range of stock open-source small language models (including Qwen3-0.6B, Gemma-3-4B, lfm2-1.2b, and others) via MLC-LLM, WebLLM and LMStudio using TestRig to compare their ability to follow structured prompts and assign sensible phishing risk scores consistently. We ultimately selected an off-the-shelf model that offered the best trade-off between size, latency, and instruction-following capability in the browser. This decision was partly pragmatic (time and compute constraints during a weekend hackathon) and partly intentional: we wanted TinyRod to be a realistic example of how organisations could make progress today using open SLMs and careful prompt design, without needing bespoke training pipelines or GPU clusters. The following subsections go into detail on each component - TinyRod, TestRig, and Harbour - and how these choices shaped their implementation.

Components:

A. TinyRod - Chrome Extension

The Chrome extension (TinyRod) provides client-side phishing detection for Outlook webmail and Outlook Office365. It runs a local transformer based inference in the browser using WebLLM, which makes use of WebGPU, allowing device GPUs to expose features such as fp16 with no drivers or installs (under the right conditions). We chose this fully client-side approach for three reasons: first, it avoids sending raw email content to any external server, which is important for both privacy and regulatory concerns; second, it removes the need for a dedicated inference backend, making the solution easier to deploy in a hackathon setting and more scalable in principle; and third, it lets us experiment with “on-device AI” as a realistic defence pattern against AI-enabled phishing in a way which can be replicated by most people who have an email account (and usually access it via a browser). Targeting Outlook on the web was also deliberate: it is widely used in enterprise environments, has a rich, scriptable DOM surface, and integrating at the browser layer means we do not need administrative control over the mail server or gateway.


You have W0N a Premier Inn Luxury Pillows 2-piece set

⚠ Malicious (Score: 92)

- Sender address doesn't match display name
- too good to be true offers
- requests for personal information

PR Premier Inn Rewards<newsletter@eQcwf.netshop-isp.com.cy>
To: You; admin@oeuz.Tn0dglJ72p.WQb
Sat 11/22/2025 9:49 AM

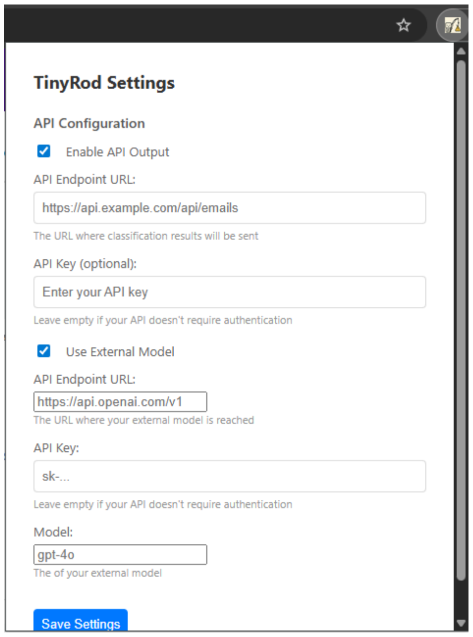
Harry Waterman.pdf
326 bytes

The extension integrates with Outlook by injecting a "Scan" button into the message toolbar. When activated, it extracts email metadata from the DOM, including sender email and display name, recipients, subject, body, URLs, and attachment names. It uses a content script that adapts to Outlook's dynamic interface, handling DOM changes and extracting structured data from the reading pane which makes it feel like a native outlook feature. This design keeps the user experience simple (one extra button where they already expect controls) and reduces training overhead: users do not have to switch to another page or copy–paste emails into a separate tool. Working directly against the Outlook DOM also means we see the email as the user sees it, including formatting, link text and display names, which are often key to phishing detection. We accepted the added engineering complexity of dealing with a single-page, dynamically updated interface because the payoff is a smoother, more trustworthy user experience and a clearer path to adoption.

The core detection engine uses the Qwen3-4B-q4f16_1-MLC model (quantized to 4-bit), which is downloaded into a cache the first time the user chooses to scan, allowing for quicker subsequent requests. We selected this model as a compromise between capability and footprint: it is large enough to follow structured instructions and reason about nuanced email content, but small and quantised enough to run in a browser with WebGPU on typical modern hardware(WebGPU works on Intel 11th Gen and newer). Using an open, locally hosted model rather than a remote API removes network latency from the critical path, avoids API rate limits, and makes the threat model cleaner - there is no third-party service that could log or leak email content.

Once you choose to scan, the extension extracts and sends email metadata plus the system prompt, which guides the model to respond with a score of 0-100 based on how likely the email is to be malicious (with 0 being benign and 100 being malicious), along with a preset list of reasons that describe how the email is malicious such as “the sender address not matching the sender display name”. The model then responds in structured JSON allowing us to display the information back in the UI. We chose a numeric risk score plus a controlled vocabulary of reasons to balance explainability and robustness: the score gives a quick, single-number signal for users and dashboards. We do also recognise the weakness of ranged based scoring methods with nondeterministic models, however in this instance we feel it's important to test each models tendency to score in a certain way. While the predefined reasons keep the output constrained enough to be machine-readable and comparable across scans. Requiring strict JSON output is a pragmatic engineering choice: it simplifies parsing, makes the UI rendering deterministic, and allows the same results to be forwarded to Harbour with minimal transformation. Recent models are also more likely to enforce a structured output via a JSON schema making this approach even more reliable.

The extension serves as a standalone local analysis tool, which in organisational scenarios can be toggled to start sending structured metadata to Harbours API server for centralised storage and further analysis. This architecture enables both immediate local detection and long-term pattern analysis across the email network for Enterprise. We deliberately separated these concerns so that individuals or small teams can use TinyRod purely as a privacy-preserving personal safeguard, while larger organisations can opt in to centralised telemetry when they want fleet-wide visibility. By only sending minimised, structured metadata (scores, labels, domains, and flags) rather than full message bodies, we preserve much of the privacy advantage of on-device inference while still giving defenders enough signal to identify campaigns, shared infrastructure and recurring lures.


TinyRod Settings

API Configuration

☑ Enable API Output

API Endpoint URL: https://api.example.com/api/emails

The URL where classification results will be sent

API Key (optional): Enter your API key

Leave empty if your API doesn't require authentication

☑ Use External Model

API Endpoint URL: https://api.openai.com/v1

The URL where your external model is reached

API Key: sk-...

Leave empty if your API doesn't require authentication

Model: gpt-4o

The of your external model

The TinyRod Settings page is a configuration interface for the Chrome extension. It configures where classification reports are sent and whether an external inference engine is used.

API Configuration:

Enable API Output checkbox: Toggles sending classification results to an API endpoint (Harbour)

When enabled, shows:

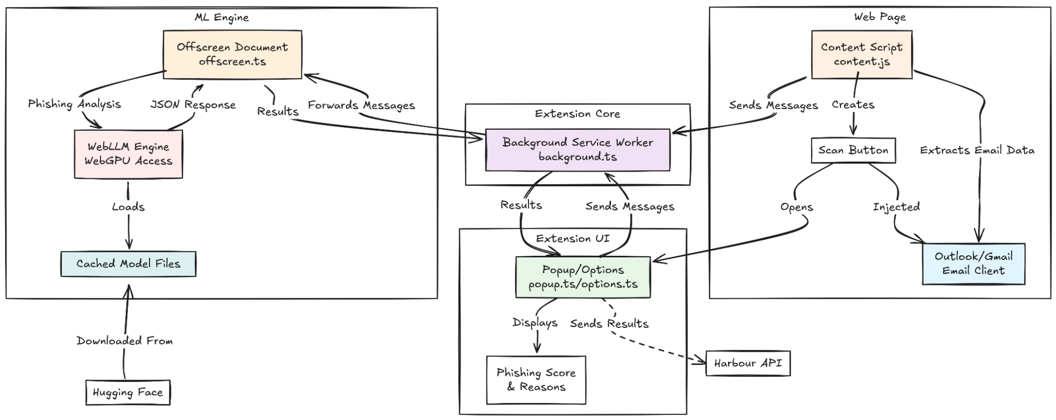
- API Endpoint URL field: URL where classification results are sent (e.g.,https://api.example.com/api/emails)
- API Key (optional) field: Password field for authentication (Bearer token)

External Model Section:

Use External Model checkbox: Toggles using an external AI model instead of the local one

When enabled, shows:

- API Endpoint URL field: External model OpenAI API compatible endpoint (e.g., https://api.openai.com/v1)
- API Key field: API Key field field for the external model API
- Model field: Model identifier (e.g., gpt-4o)



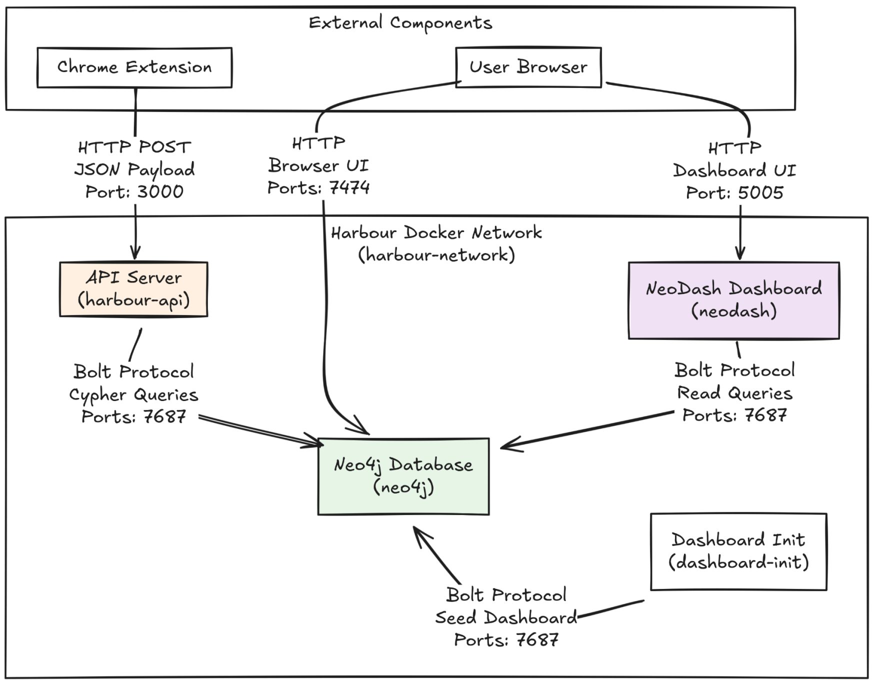
TinyRod Extension Architecture

B. Harbour - Centralised Reporting

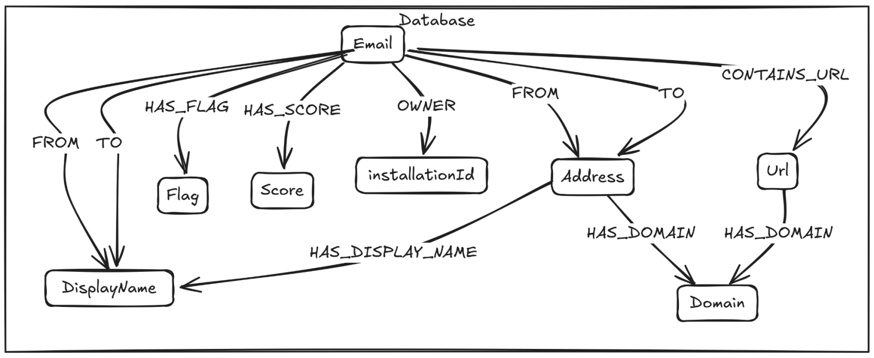
Harbour is a centralised data collection and storage component. It aggregates email data from the Chrome extension and stores it in a Neo4j graph database, enabling analysis and visualisation of email patterns, relationships, and security indicators.

Harbour uses a microservices architecture deployed with Docker Compose. The architecture uses a shared Docker network for service discovery and communication, allowing components to interact while maintaining isolation.

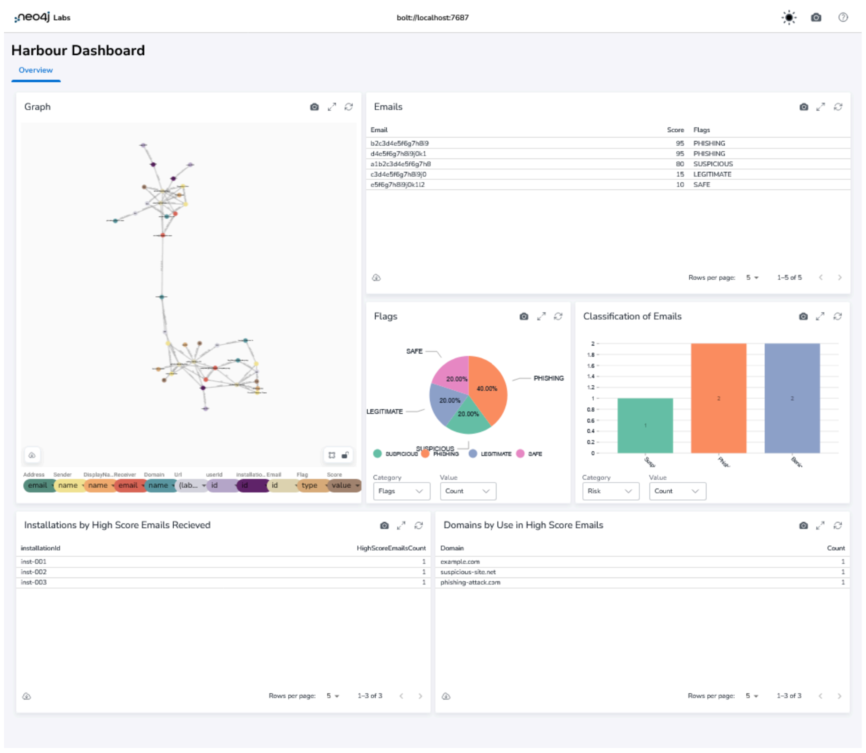
The system stores email metadata as a graph, modeling entities (emails, addresses, domains, URLs) and relationships (sender/recipient connections, domain associations, flagged content). This supports analysis of email networks, threat detection, and risk scoring. The API translates JSON payloads from the Chrome extension into graph operations, while the dashboard provides real-time visualizations of email classifications, threat flags, and network relationships, supporting security analysis and decision-making.



Harbour Component Architecture and Data Flow Diagram



Data Model



Emails

Email	Score	Flags
b2c3d4e5f6g7h8i9	95	PHISHING
d4e5f6g7h8i9j0k1	95	PHISHING
a1b2c3d4e5f6g7h8	80	SUSPICIOUS
c3d4e5f6g7h8i9j0	15	LEGITIMATE
e5f6g7h8i9j0k1l2	10	SAFE

Rows per page: 5 1–5 of 5

Installations by High Score Emails Recieved

installationId	HighScoreEmailsCount
inst-001	1
inst-002	1
inst-003	1

Rows per page: 5 1–3 of 3

Domains by Use in High Score Emails

Domain	Count
example.com	1
suspicious-site.net	1
phishing-attack.com	1

Rows per page: 5 1–3 of 3

Dashboard Screenshot

These design choices are driven by both the threat model and the practical constraints of a hackathon setting. Phishing campaigns are inherently relational: the same sender may target multiple recipients, different emails may reuse infrastructure (domains, URLs, hosting), and attack waves often share common wording or lures. Representing this as a graph rather than flat tables makes it much easier to see patterns such as: "which domains are appearing across many emails?", "which installations/users are repeatedly targeted?", or "how do risky URLs cluster around particular campaigns?" Neo4j is therefore a deliberate choice as it aligns directly with the problem domain and allows Harbour to express complex security questions in relatively simple Cypher queries.

The separation between the Chrome extension and Harbour via a REST API is another deliberate decision. The extension never talks to Neo4j directly; instead, it sends a small, structured JSON payload over HTTPS to the API. This provides a clean boundary where we can enforce data minimisation, validation, and authentication, and it means Neo4j stays inside a protected network segment. It also preserves flexibility: the extension doesn't need to know anything about the internal graph schema, only the API contract. This is important both for security (no database credentials in the client) and for evolvability (we can change the graph model without breaking the extension), as well as maintaining the value of running TinyRod as a solo deployment.

Using a microservices-style layout with Docker Compose rather than a single monolithic container/software solution is a pragmatic choice that balances simplicity and clarity. For a hackathon, Compose gives a reproducible "one command up" deployment, but splitting Harbour into four services makes the architecture easier to reason about and extend:

- Neo4j focuses purely on graph storage and querying.
- The API service encapsulates business logic and graph writes.
- NeoDash is an off-the-shelf visualisation layer that can be swapped or extended.
- The init service seeds the dashboard configuration, ensuring each deployment comes up with a working, pre-configured view.

This decomposition demonstrates good engineering practice (clear boundaries, least privilege, replaceable components) without adding excessive operational complexity.

The initialisation service in particular is a small but important design choice. Rather than manually configuring dashboards every time Harbour is deployed, the init container reads a dashboard.json definition and writes it into Neo4j as a :_Neodash_Dashboard node. Combined with NeoDashs standalone mode, this means the system always comes up with a consistent, preloaded view of TinyRods telemetry.

Finally, modelling only metadata and derived signals (addresses, domains, URLs, labels, risk scores, flags) rather than full email content is an intentional privacy and scope decision. Harbour is designed to support pattern analysis and campaign tracking, not full content archiving or forensic reconstruction. This aligns with the projects goal of exploring privacy-preserving, browser-centric phishing defence: TinyRod keeps full content and analysis on the client, while Harbour aggregates just enough structured information to reveal meaningful trends and support security decision-making at an organisational level.

C. TestRig - Evaluation & Benchmarking

The evaluation system uses the exact same system prompt and logic as the Chrome Extension, ensuring that benchmark results accurately reflect production behavior. This alignment is critical for validating that the models tested in evaluation will perform similarly when deployed in the browser extension.

Caveat on Benchmarks: A different inference engine has been used to conduct the benchmarks (LM Studio) vs. what is included in the TinyRod PoC (Web-LLM).
Reasons for this difference:

- Web-LLM model selection is limited, some key SLMs we wanted to test have not been converted to the MLC format required.
- LMStudio has models better optimised for MLX (the ML Library for Apple Silicon) which allowed us to expand the number of models in our benchmark due to the decreased inference time. (With the exception of two cloud-based reference models (gpt-5.1) to establish performance baselines, everything we are showing here has been done locally)

Dataset

Source: Kaggle, Dataset Name: Phishing Email Dataset

Creator: naserabdullahalam

URL: kaggle.com/datasets/naserabdullahalam/phishing-email-dataset

Class Distribution: Approximately 82,500 emails, 42,891 (52%) phishing emails, 39,595 (48%) legitimate emails

(Al-Subaiey et al.)

This dataset was compiled by researchers to study phishing email tactics. It combines emails from a variety of sources to create a comprehensive resource for analysis. The dataset contains email samples labeled as phishing or legitimate emails, designed for training and evaluating machine learning models to detect phishing attempts.

Features include:

- Email subject lines
- Email body content
- Sender information
- URLs/links within emails
- Attachments when provided (only a small subset of the emails has attachments)
- Labels (phishing/legitimate)

Rather than relying on a single source from the dataset, the evaluation framework automatically finds, loads, and unifies multiple CSVs from the project directory. This includes:

- Nigerian Fraud dataset
- Enron email dataset
- SpamAssassin dataset
- CEAS_08 dataset
- Ling dataset
- Phishing_email dataset
- Nazario dataset

The dataset loader ('dataset_loader.py') handles different schema formats:

- Detailed schema: 'sender', 'receiver', 'date', 'subject', 'body', 'urls', 'label'
- Simple schema: 'subject', 'body', 'label'
- Combined schema: 'text_combined', 'label' (with automatic parsing of Subject/Sender from body when possible)

Quality filters automatically exclude emails with body content less than 20 characters to remove empty or gibberish rows. All labels are unified to '0' (Benign) and '1' (Malicious), and all data is merged into a single standardized DataFrame for consistent evaluation.

Issues identified with the dataset

1. Many emails in the dataset are missing body / subject / other critical information for accurate testing, many were too messy to use with email chain framing and other broken characters being included, we have tried to pre-process the dataset where possible to remove these.
2. Feature labels are not consistent across CSV files within the dataset making it hard to sample randomly from all files if you want as close to real-world testing as possible.

Balanced Sampling Methodology

To prevent class imbalance bias, the evaluation framework implements strict balanced sampling:

1. The unified dataset is split into Benign and Malicious groups

2. A random sample of size N/2 is drawn from each group (where N is the total sample size)

3. These samples are combined and shuffled to create the final test set

This ensures fair testing with a 50/50 split of Benign vs. Malicious emails, preventing models from achieving high accuracy simply by predicting the majority class. The random seed flag has also been added to TestRig ensuring consistent testing across all models.

For each email in the balanced sample, the system:

- Sends the complete email context (Subject, Sender, Body Snippet, and Attachments) to the LLM
- Parses the JSON response, handling local model quirks such as '<think>' tags
- Compares the models score (threshold: >50 = Malicious) against the ground truth label

- Calculates comprehensive metrics:
    - Confusion Matrix: True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN)
    - Accuracy: Overall percentage of correct predictions
    - Precision: TP / (TP + FP) - measures how many flagged emails are actually malicious
    - Recall: TP / (TP + FN) - measures how many malicious emails are correctly identified
    - F1-Score: Harmonic mean of Precision and Recall
    - False Positive Rate (FPR): FP / (FP + TN) - critical for UX (blocking legitimate emails reduces user trust)
    - False Negative Rate (FNR): FN / (FN + TP) - critical for Security (missing threats creates vulnerabilities)
    - Latency: Average time per inference, measured in milliseconds

We also measure the stability and consistency (repeatability) of the model's scoring and reasoning by querying the same email multiple times. The system:

- Selects a subset from the sample
- Queries the LLM X times for the same email
- Calculates:
    - Score Variance: How much the numerical score (0-100) fluctuates across runs
    - Reason Consistency: How often the exact same set of reasons is returned across multiple queries
    - Consistency Percentage: Percentage of runs producing identical or near-identical outputs

This repeatability analysis is essential for understanding model reliability in production, where users expect consistent results when scanning the same email multiple times.

LLM Client and API Integration

The evaluation framework includes a custom LLM client ('llm_client.py') that connects to any OpenAI-compatible API endpoint. This design allows testing against:

- Local LLM servers (e.g., LMStudio, Ollama) running on 'http://localhost:1234/v1'
- Remote API providers (e.g., OpenAI) using API keys
- MLC-LLM servers for testing the same models & inference configuration used in production

The client handles API connections, prompting with the production system prompt, and response parsing with error handling for various model output formats.

Benchmark Execution Modes

The evaluation framework supports multiple execution modes:

Multi-Model Benchmarking: A script ('run_multi_model_benchmark.sh') enables sequential benchmarking of multiple models (e.g., Qwen, Gemma, Phi-4) with manual model switching prompts, allowing comprehensive model comparison studies. This approach also allowed us to run multiple smaller models and instances of the script at once within our 18GB VRAM limit, speeding up benchmarking significantly

Manual Benchmarking: Direct execution of 'benchmark_all.py' with customisable parameters:

- **--model:** Model identifier to pass to the API (e.g., "qwen/qwen3-1.7b").
- **--sample-size:** Total number of emails to use for the accuracy test (default: 0; split 50/50 benign/malicious).
- **--repeat-runs:** Number of iterations to run for the repeatability test on selected samples (default: 0).
- **--api-url:** URL of the LLM API endpoint (default: "http://localhost:1234/v1").
- **--api-key:** API Key for external providers like OpenAI (optional).
- **--dataset:** Specific CSV filename to load (e.g., "Nazario.csv") if you want to test only one dataset instead of all.
- **--temperature:** Sampling temperature for the model (default: 0.1, lower is better for consistency).
- **--seed:** Random seed integer for reproducible sampling of the dataset.
- **--only-repeatability:** Flag to skip the accuracy test entirely and run only the repeatability benchmark.
- **--data-dir:** Directory path to scan for CSV datasets (default: project root).
- **--output:** Custom file path for saving the output JSON report.

Model Selection and Configuration

During development, we evaluated and benchmarked multiple open-source SLMs including various qwen3 sizes, gemma3, phi-4-mini, granite-4-h-tiny & lfm2 to assess performance trade-offs.

System Prompt

Multiple prompts were created during the initial development phase, experimenting with the impact adding things like few shot prompts had of models of different sizes, and aiming to reduce ambiguity whilst giving explicit patterns for the model to follow. We settled on this prompt pretty early on knowing we wouldn't be able to also benchmark the impact of different prompts on the results as well due to time constraints.

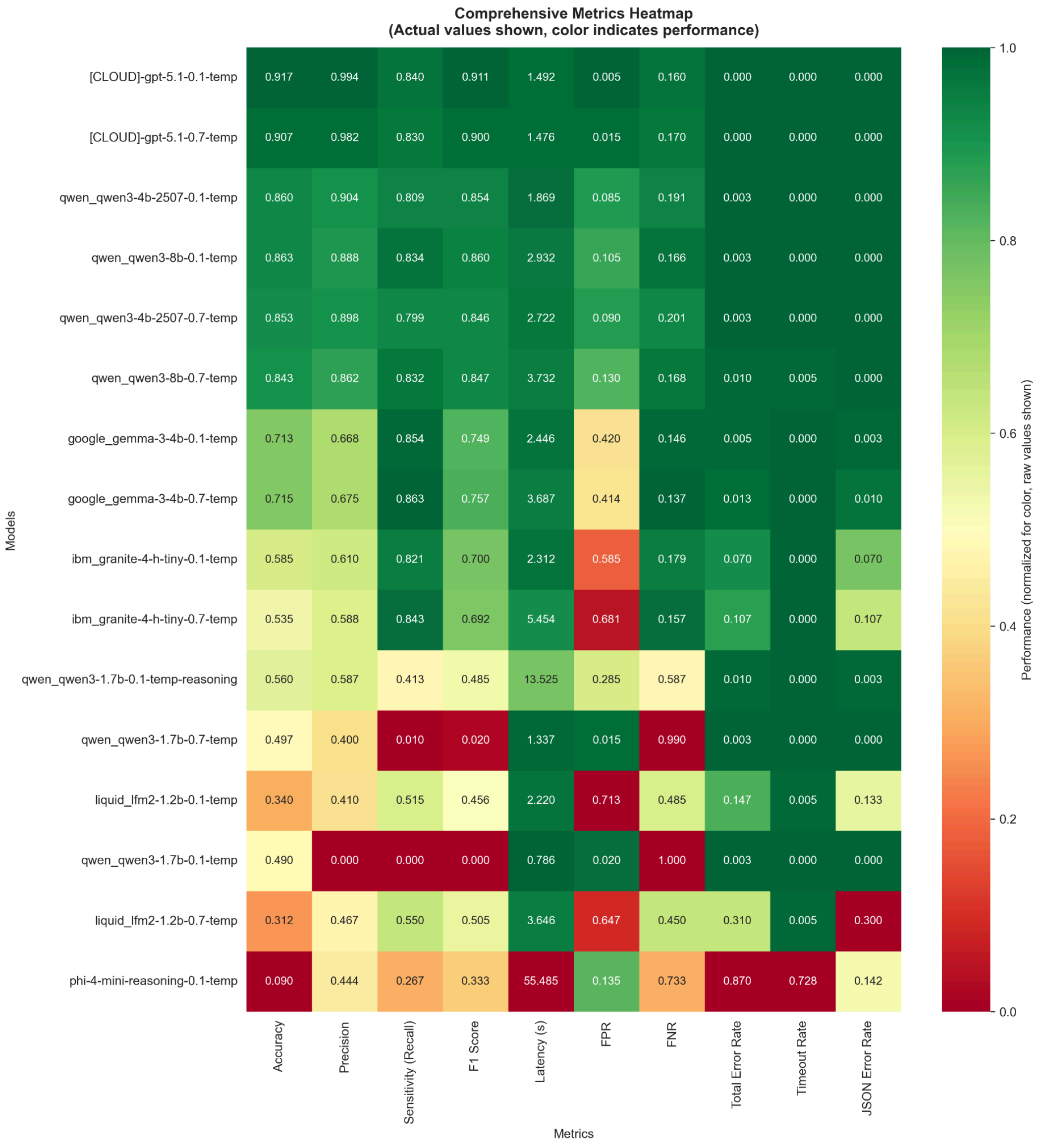
The prompt instructs the model to rate emails on a 0-100 confidence scale based on malicious indicators, with higher scores indicating greater confidence that an email is malicious. Criteria for higher scores are then defined, including attempts to obtain money, bank details, credentials, or cryptocurrency; promises of unexpected large sums; requests to click links or open files to fix or verify something; and impersonation of banks, governments, or large companies.

The model is explicitly asked to output only valid JSON with a score and an optional list of reasons. This structured output enables consistent parsing and display in the user interface while providing explainable feedback about why an email was flagged.

We have benchmarked results at 0.1 temperature and 0.7 temperature for all of the models, allowing us to see how much of an impact reducing the variance has on repeatability and accuracy.

3. Results

We evaluated 16 model configurations across 8 distinct small language models (SLMs) on a balanced dataset of 400 emails (50% benign, 50% malicious) from the Phishing Email Dataset (Al-Subaiey et al.). For comparison, we also included two cloud-based reference models (gpt-5.1) to establish performance baselines. All models were tested using the same system prompt and evaluation framework (TestRig) to ensure consistent comparison. Each model was evaluated using different temperatures (0.1, and 0.7) as well as a test of the repeatability - how consistent the model is at outputting the same results when given the same email to assess.



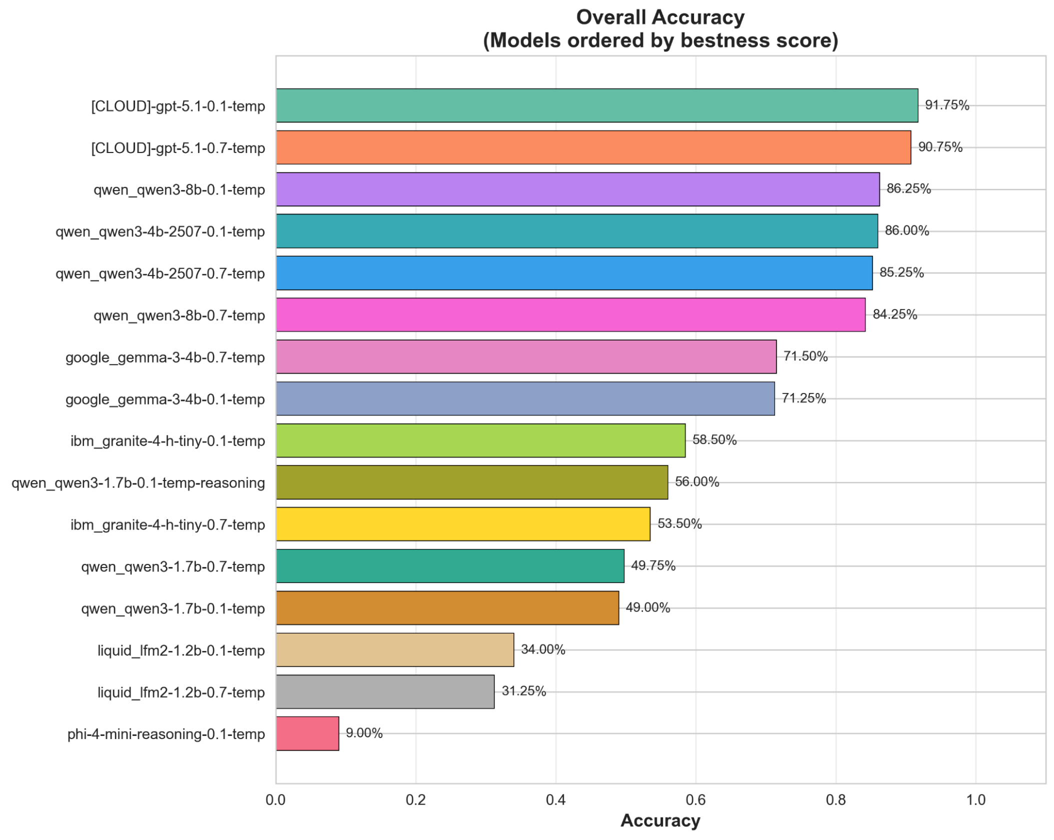
Models	Accuracy	Precision	Sensitivity (Recall)	F1 Score	Latency (s)	FPR	FNR	Total Error Rate	Timeout Rate	JSON Error Rate
[CLOUD]-gpt-5.1-0.1-temp	0.917	0.994	0.840	0.911	1.492	0.005	0.160	0.000	0.000	0.000
[CLOUD]-gpt-5.1-0.7-temp	0.907	0.982	0.830	0.900	1.476	0.015	0.170	0.000	0.000	0.000
qwen_qwen3-4b-2507-0.1-temp	0.860	0.904	0.809	0.854	1.869	0.085	0.191	0.003	0.000	0.000
qwen_qwen3-8b-0.1-temp	0.863	0.888	0.834	0.860	2.932	0.105	0.166	0.003	0.000	0.000
qwen_qwen3-4b-2507-0.7-temp	0.853	0.898	0.799	0.846	2.722	0.090	0.201	0.003	0.000	0.000
qwen_qwen3-8b-0.7-temp	0.843	0.862	0.832	0.847	3.732	0.130	0.168	0.010	0.005	0.000
google_gemma-3-4b-0.1-temp	0.713	0.668	0.854	0.749	2.446	0.420	0.146	0.005	0.000	0.003
google_gemma-3-4b-0.7-temp	0.715	0.675	0.863	0.757	3.687	0.414	0.137	0.013	0.000	0.010
ibm_granite-4-h-tiny-0.1-temp	0.585	0.610	0.821	0.700	2.312	0.585	0.179	0.070	0.000	0.070
ibm_granite-4-h-tiny-0.7-temp	0.535	0.588	0.843	0.692	5.454	0.681	0.157	0.107	0.000	0.107
qwen_qwen3-1.7b-0.1-temp-reasoning	0.560	0.587	0.413	0.485	13.525	0.285	0.587	0.010	0.000	0.003
qwen_qwen3-1.7b-0.7-temp	0.497	0.400	0.010	0.020	1.337	0.015	0.990	0.003	0.000	0.000
liquid_lfm2-1.2b-0.1-temp	0.340	0.410	0.515	0.456	2.220	0.713	0.485	0.147	0.005	0.133
qwen_qwen3-1.7b-0.1-temp	0.490	0.000	0.000	0.000	0.786	0.020	1.000	0.003	0.000	0.000
liquid_lfm2-1.2b-0.7-temp	0.312	0.467	0.550	0.505	3.646	0.647	0.450	0.310	0.005	0.300
phi-4-mini-reasoning-0.1-temp	0.090	0.444	0.267	0.333	55.485	0.135	0.733	0.870	0.728	0.142

Heatmap Summary of Model Benchmarking

Accuracy Metrics

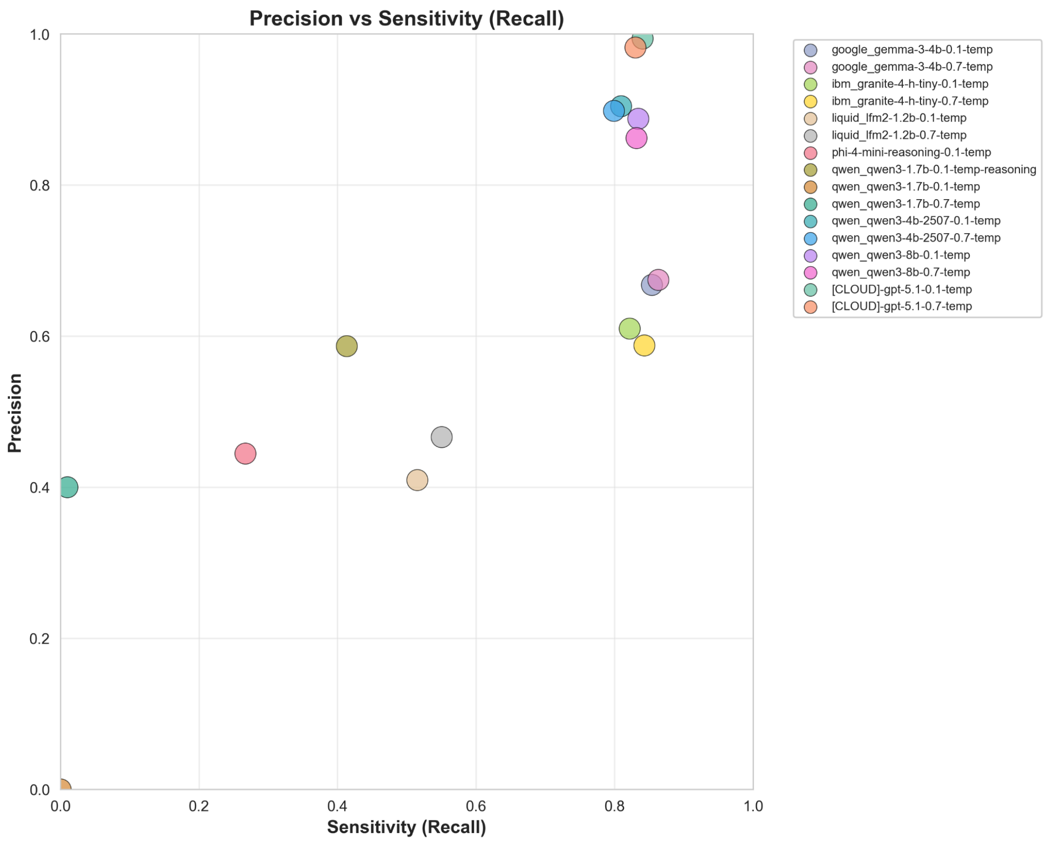
Overall Performance

Across the balanced test set of 400 emails, model accuracy varied substantially:



- Highest Accuracy (Local Models): Qwen3-8B (temperature 0.1) achieved 86.25% accuracy, closely followed by Qwen3-4B-2507 (temperature 0.1) at 86.00%
- Cloud Reference Baseline: gpt-5.1 (temperature 0.1) achieved 91.75% accuracy
- Lowest Performance: Several models underperformed significantly, with phi-4-mini-reasoning achieving only 9% accuracy due to extensive timeouts (87% failure rate), and the base Qwen3-1.7B model achieving 49% accuracy

Precision, Recall, and F1-Score



Precision (the proportion of flagged emails that are actually malicious) is critical for user experience, as false positives erode trust in the system. The best local models achieved:

- Qwen3-4B-2507 (temperature 0.1): 90.45% precision, 80.90% recall, 85.41% F1
- Qwen3-8B (temperature 0.1): 88.77% precision, 83.42% recall, 86.01% F1
- Cloud Reference: 99.41% precision, 84.00% recall, 91.06% F1

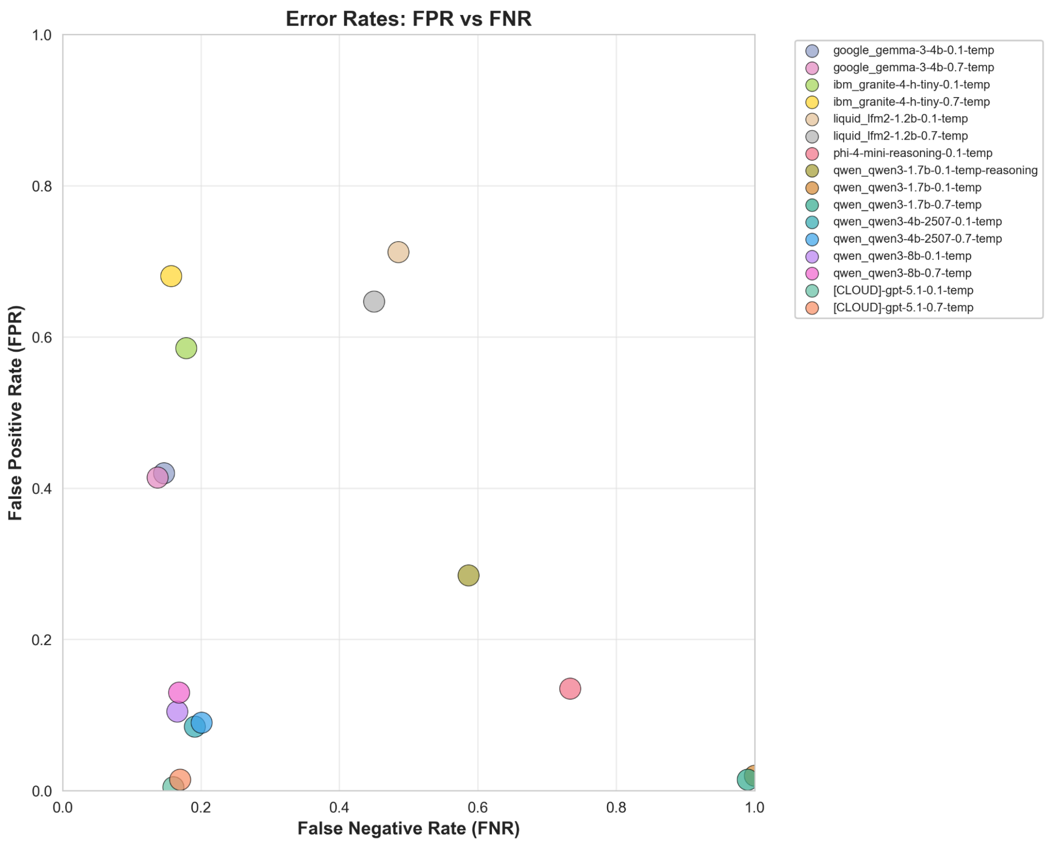
Sensitivity (Recall) measures the proportion of malicious emails correctly identified, which is critical for security. The best local model for recall was Google Gemma-3-4B (temperature 0.7) at 86.29%, though this came at the cost of a higher false positive rate (41.41%) which suggests a lack of understanding for the task.

F1 Score = 2 × (Precision × Recall) / (Precision + Recall)

Or equivalently:

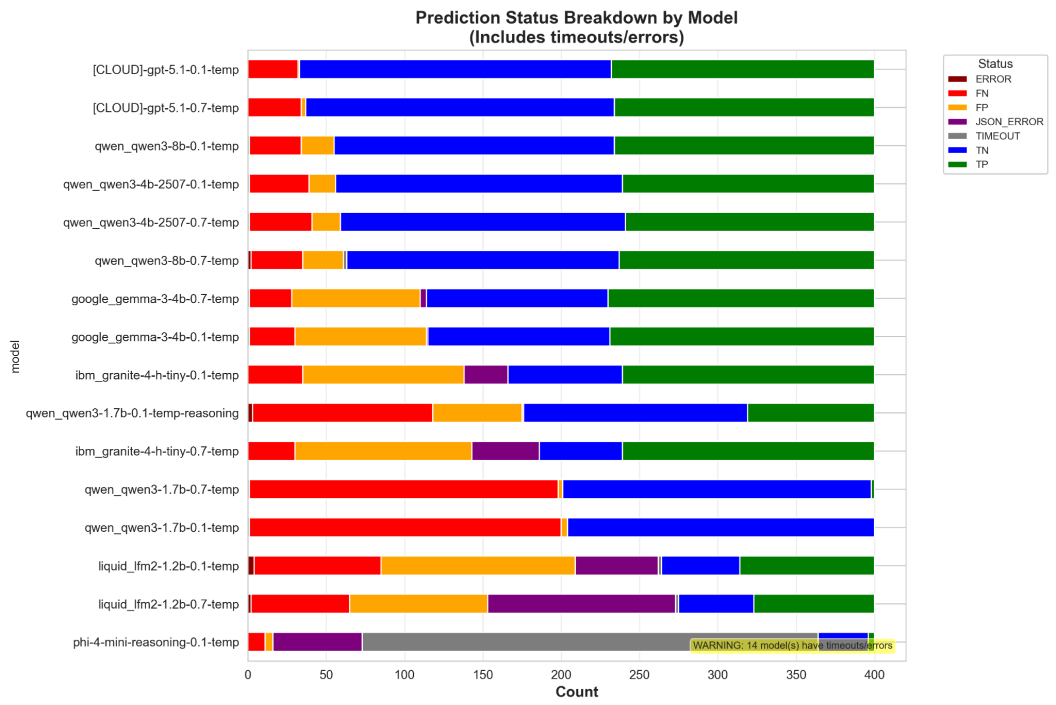
F1 Score = 2 × (TP) / (2 × TP + FP + FN)

False Positive and False Negative Rates



False Positive Rate (FPR) directly impacts user experience: models with high FPR will flag legitimate emails, reducing user trust and potentially causing alert fatigue. The best local models achieved FPRs of 8.5-10.5% (Qwen3-4B and Qwen3-8B), which is still too high for practical real world use, compared to the cloud reference's 0.50%. The worst-performing local models (liquid_lfm2-1.2b, ibm_granite-4-h-tiny) showed FPRs above 58%.

False Negative Rate (FNR) represents missed threats and is critical for security. The best local models achieved FNRs of 16.5-19.1% (Qwen3-8B and Qwen3-4B), meaning they correctly identify 81-84% of malicious emails. The cloud reference achieved 16.00% FNR. Notably, some smaller models (Qwen3-1.7B) showed catastrophic FNRs approaching 100%, effectively failing to detect malicious emails, we suspect this reflects a combination of model limitations and noisy, historic examples in the dataset; more controlled data would be needed to disentangle these effects.



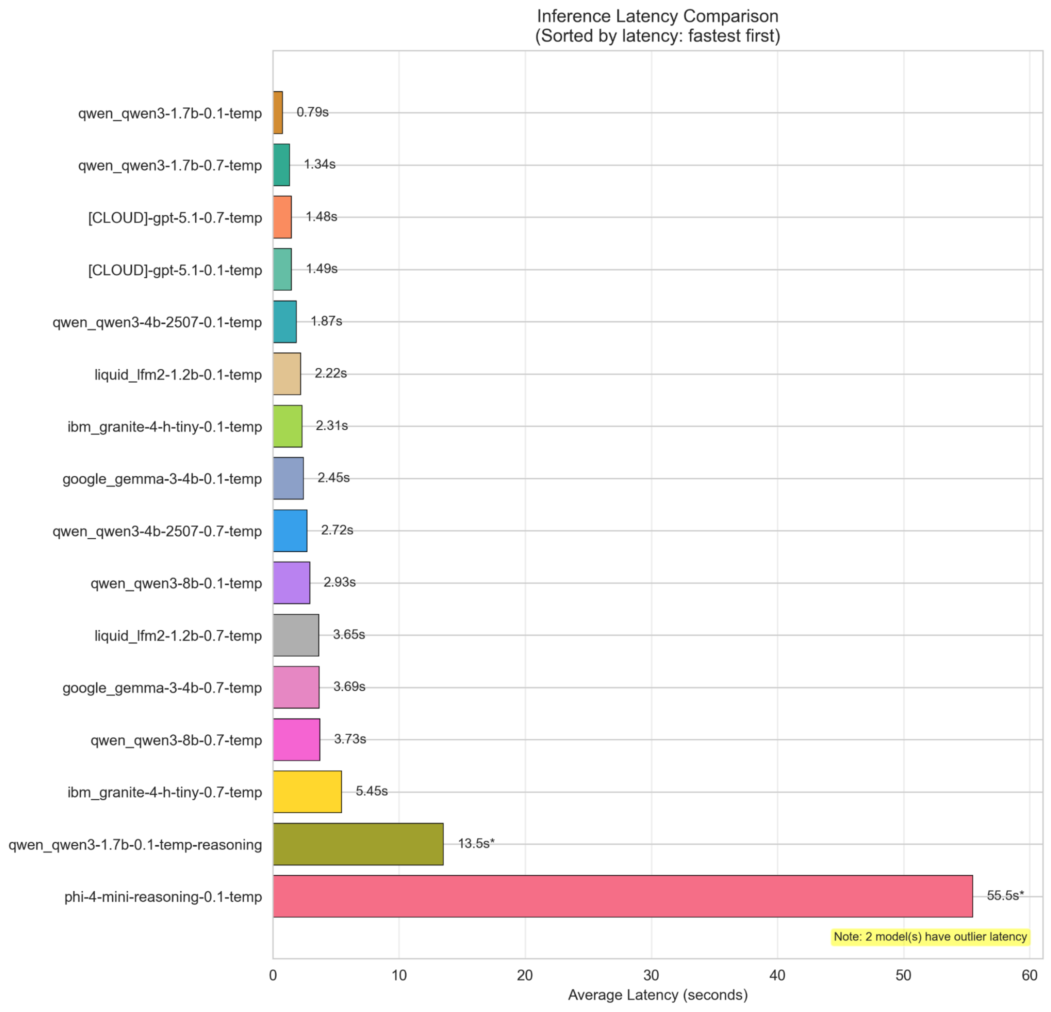
Confusion Matrix Analysis

Detailed confusion matrices reveal the composition of errors across models. The top-performing local models (Qwen3-4B, Qwen3-8B) show balanced error distributions:

- Qwen3-4B-2507 (temperature 0.1): 161 TP, 183 TN, 17 FP, 38 FN (out of 400 emails)
- Qwen3-8B (temperature 0.1): 166 TP, 179 TN, 21 FP, 33 FN

Confusion Matrices available at https://github.com/Harry-Waterman/def-acc-logiq/blob/main/benchmark-analysis/visualizations/confusion_matrices.png

Latency Measurements



Inference latency is critical for browser-based, real-time detection. The fastest local model was Qwen3-1.7B (temperature 0.1) at 0.79 seconds average, though this model's accuracy was insufficient for production use. The best-performing models achieved acceptable latencies:

- Qwen3-4B-2507 (temperature 0.1): 1.87 seconds
- Qwen3-8B (temperature 0.1): 2.93 seconds
- Cloud Reference: 1.49 seconds

These latencies are suitable for real-time browser-based detection, where users expect results within 2-3 seconds of clicking "Scan". The slowest model (phi-4-mini-reasoning) averaged 55.49 seconds with 87% timeout rate, making it completely unsuitable for this use case.

Model Comparison

Local SLMs vs. Cloud Reference Models

The cloud reference models (gpt-5.1) serve as performance baselines, representing state-of-the-art cloud-based LLM performance. These models achieved 91.75% accuracy with exceptional precision (99.41%) and low false positive rates (0.50%).

The best local models (Qwen3-4B and Qwen3-8B) achieve 86-86.25% accuracy, representing a 5.5-5.75 percentage point gap from the cloud reference. This gap could be seen as acceptable given the privacy and deployment advantages: local models keep all email content on-device, eliminate network latency and API dependencies, and can run entirely offline. The performance difference is primarily in precision: local models show 8.5-10.5% false positive rates compared to the cloud reference's 0.50%, suggesting that local models may benefit from additional calibration or hybrid rule-based filtering to reduce false positives.

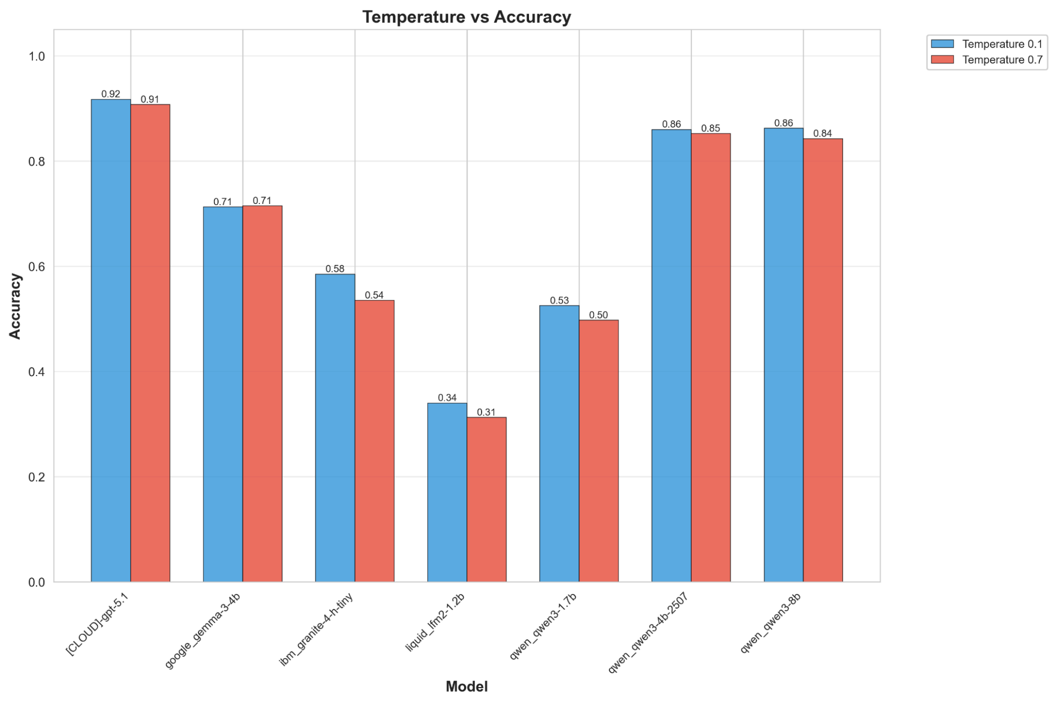
Model Size and Architecture Comparison

We evaluated models ranging from 1.2B to 8B parameters to assess the relationship between model size and detection capability:

- 1.2B parameters (liquid_lfm2-1.2b): 34% accuracy, high FPR (71%), unsuitable for production
- 1.7B parameters (qwen_qwen3-1.7b): 49% accuracy, near-zero recall, unsuitable for production
- 4B parameters: Performance varied significantly by architecture:
- Qwen3-4B-2507: 86% accuracy, excellent precision-recall balance
- Google Gemma-3-4B: 71.25% accuracy, high recall (85%) but high FPR (42%)
- IBM Granite-4-H-Tiny: 58.5% accuracy, moderate performance
- 8B parameters (qwen_qwen3-8b): 86.25% accuracy, best overall local performance

The results suggest a performance "sweet spot" around 4-8B parameters for this task, with architecture and training quality mattering more than raw parameter count. The Qwen3-4B-2507 variant, despite being smaller than the 8B model, achieved comparable accuracy (86% vs 86.25%), suggesting that newer model architectures and training techniques can achieve better performance at smaller sizes.

Temperature Impact



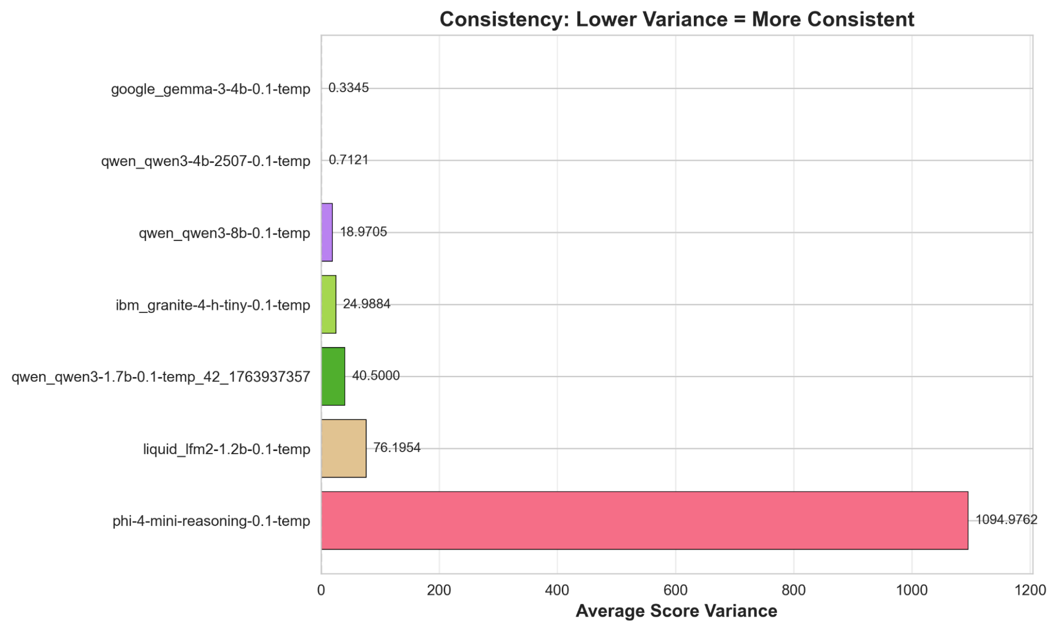
We evaluated all models at two temperature settings (0.1 and 0.7) to assess the impact of sampling variance on performance. Lower temperature (0.1) generally improved precision and consistency, while higher temperature (0.7) slightly increased recall but at the cost of higher false positive rates. For most models, the accuracy difference between temperatures was modest (0-3 percentage points), suggesting that temperature selection is more important for repeatability than for raw accuracy.

The Qwen3 models showed minimal temperature sensitivity: Qwen3-4B-2507 achieved 86.00% at 0.1 and 85.25% at 0.7, while Qwen3-8B achieved 86.25% at 0.1 and 84.25% at 0.7. In contrast, Google Gemma-3-4B showed slight improvement at higher temperature (71.25% at 0.1 vs 71.50% at 0.7), though this came with increased false positive rate.

Repeatability Analysis

To assess model consistency and reliability for production deployment, we conducted repeatability tests running the same email through the detection system multiple times (100 runs per email). This analysis measures two key dimensions of consistency: score variance (how much the numerical risk score fluctuates) and reason consistency (how often the model returns identical reasoning across runs).

Score Variance



Score variance measures the statistical variance in the numerical score (0-100) across multiple runs of the same email. Lower variance indicates more consistent scoring, which is important for user trust: users expect similar scores when scanning the same email multiple times.

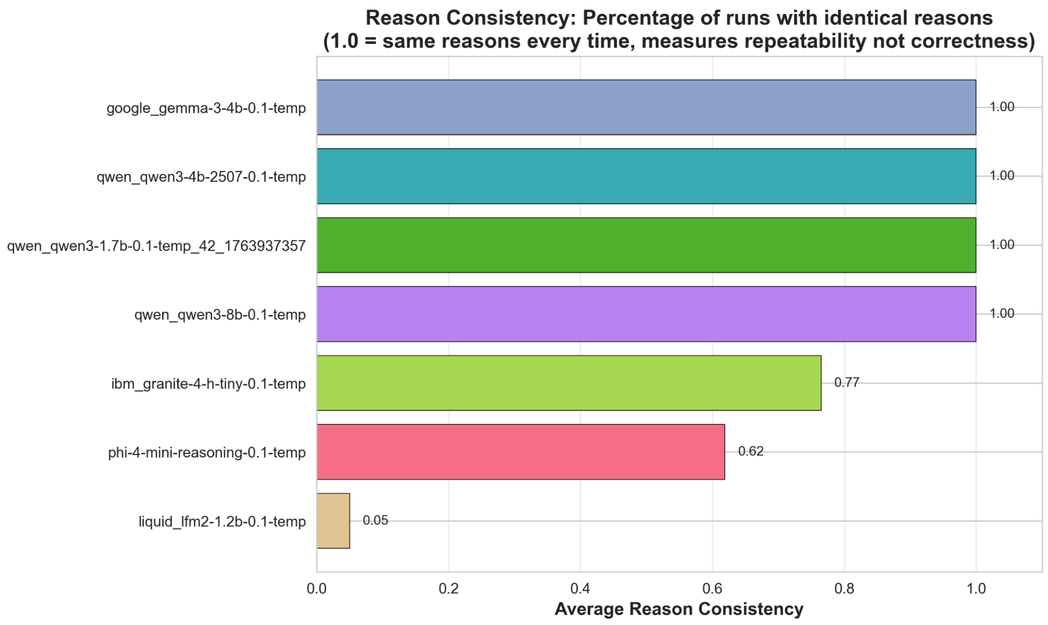
The best-performing models showed minimal score variance:

- Qwen3-4B-2507 (temperature 0.1): Near-zero variance for malicious emails (0.0), minimal variance for benign emails (1.42)
- Qwen3-8B (temperature 0.1): moderate variance for both benign (30.51) and malicious (7.43) emails
- Google Gemma-3-4B (temperature 0.1): Low variance for benign emails (0.67), zero variance for malicious emails (0.0)

In contrast, smaller or less capable models showed high variance:

- liquid_lfm2-1.2b (temperature 0.1): High variance for benign emails (113.26), moderate for malicious (39.13)
- phi-4-mini-reasoning (temperature 0.1): Extremely high variance (729.62 for benign, 1460.33 for malicious), indicating unreliable scoring

Reason Consistency



Reason consistency measures the percentage of runs that return the exact same set of reasons (e.g., "suspicious sender address", "urgent language", "credential request"). This is critical for explainability: users need consistent explanations for why an email was flagged.

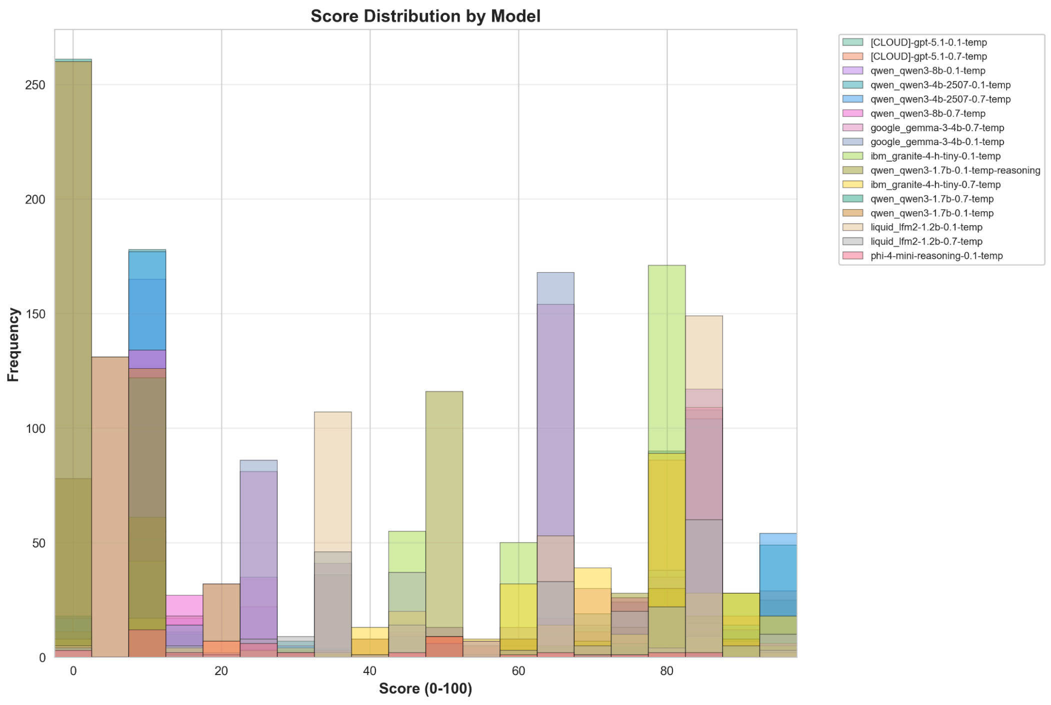
The analysis revealed a strong correlation (0.735) between model size and reason consistency. Larger models (4B-8B parameters) achieved near-perfect consistency:

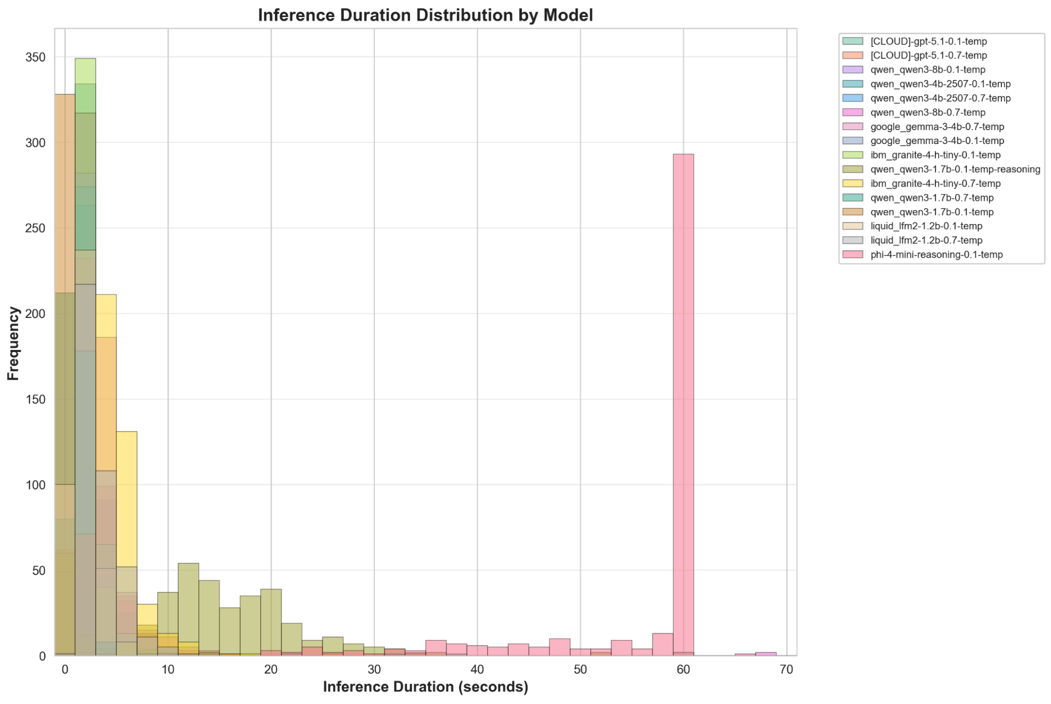
- Qwen3-4B-2507 (temperature 0.1): 100% consistency for both benign and malicious emails
- Qwen3-8B (temperature 0.1): 100% consistency for both email types
- Google Gemma-3-4B (temperature 0.1): 100% consistency for both email types

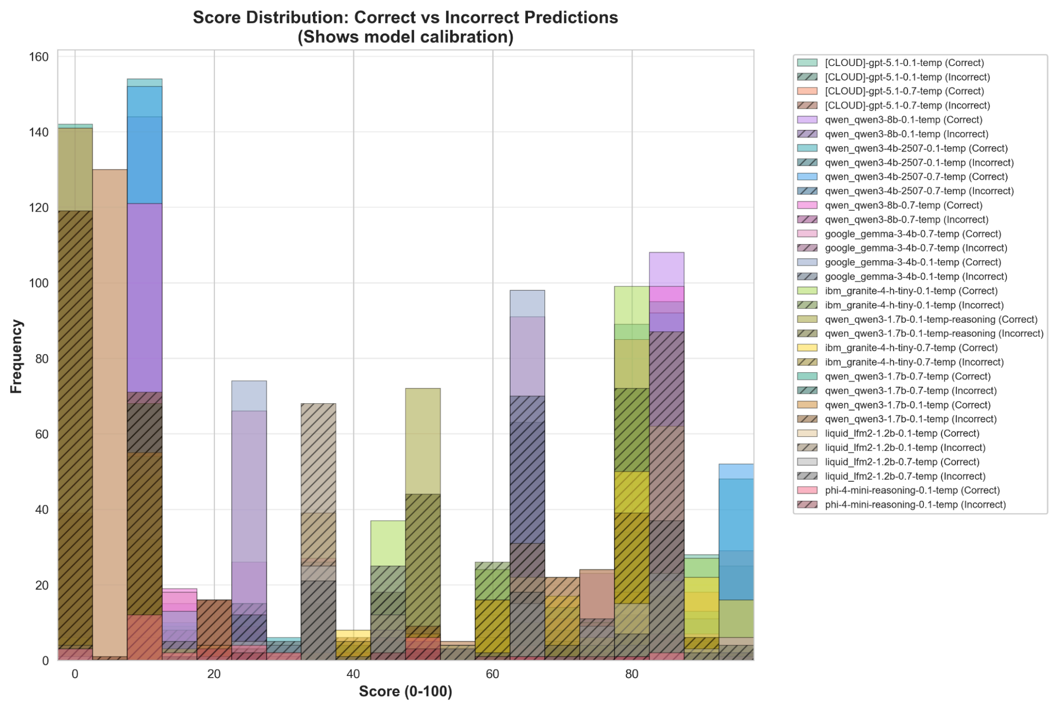
Smaller models showed lower consistency:

- liquid_lfm2-1.2b (temperature 0.1): 1.3% consistency for benign emails, 8.7% for malicious
- ibm_granite-4-h-tiny (temperature 0.1): 100% for benign, 53% for malicious

Score Distribution Analysis







The score distribution by model visualization reveals important patterns in how different models assign risk scores (0-100 scale) across the test dataset. This analysis provides insights into model calibration, confidence patterns, and scoring behavior.

Well-Calibrated Models: The top-performing models (Qwen3-4B-2507, Qwen3-8B, and the cloud reference) show distinct, well-separated score distributions. These models tend to assign:

- **Lower scores (0-30)** for benign emails, with clear peaks in the low-risk range
- **Higher scores (70-100)** for malicious emails, with distributions shifted toward the high-risk end
- **Minimal overlap** between benign and malicious score distributions, indicating good discrimination

Overconfident Models: Some models, particularly smaller ones (liquid_lfm2-1.2b, qwen_qwen3-1.7b), show problematic score distributions:

- **Bimodal or flat distributions** suggesting inconsistent scoring behavior
- **High concentration of scores in the middle range (40-60)**, indicating uncertainty or poor calibration
- **Overlapping distributions** for benign and malicious emails, making threshold selection difficult

Conservative vs. Aggressive Scoring: Models differ in their overall scoring tendencies:

- **Conservative models** (e.g., some Gemma variants) tend to assign lower average scores, requiring lower thresholds to flag emails

- **Aggressive models** assign higher average scores across the board, potentially leading to more false positives if thresholds are not adjusted accordingly
- **Balanced models** (Qwen3-4B, Qwen3-8B) show clear separation with appropriate score ranges for each class

Distribution Shape Insights:

- Models with **narrow, peaked distributions** at the extremes (0-20 for benign, 80-100 for malicious) indicate high confidence and good calibration
- Models with **wide, flat distributions** suggest uncertainty and inconsistent scoring, which correlates with lower accuracy metrics
- **Multi-modal distributions** may indicate that models are responding to different types of phishing patterns with distinct score ranges

4. Discussion and Conclusion

Our starting point was the observation that phishing is both AI-enabled and structurally social-engineering-heavy, while most deployed defences still assume centralised inspection of content and metadata. TinyRod and Harbour explore a different design point: can we run meaningful phishing detection on the client with small language models, and still give security teams useful visibility without hoovering up everyone's email?

Response to Research Questions and Hypothesises

Phishing detection at the edge

With respect to RQ1/RQ4/RQ6, the prototype demonstrates that it is technically viable to run a structured "LLM-as-a-judge" pipeline for phishing entirely in the browser using open SLMs. The Qwen3-4B-q4f16_1-MLC model, combined with WebLLM/WebGPU, runs on commodity hardware and is capable of following a strict prompt, emitting JSON, and assigning plausible phishing risk scores.

From a systems perspective, this is already a useful result: users can get AI-assisted scanning without sending raw email content to any external service. From a modelling perspective, our experiments with TestRig suggest that even very small models in the ~4B–8B range can separate benign from phishing emails well above chance on a balanced benchmark, although absolute performance and stability still lag behind larger models, albeit the gap is surprisingly minor, more testing would be required to confirm the exact difference in capability.

This supports H1/H4/H6 in a qualified way: small models are "good enough" to be part of a defence-in-depth stack, but they are not magic bullets. They should be viewed as decision-support tools that nudge users and enrich telemetry, not as perfect filters.

A key design decision was to re-use the exact same prompt and logic in both TinyRod and TestRig. This closes the usual gap between "offline benchmark" and "production behaviour": whatever we measure in TestRig is a realistic proxy for what users will see in the extension, minus the variance in dataset examples vs real world emails.

Explainability and user behaviour

For RQ3, we hypothesised that explainable feedback would be more useful than a simple "this is bad" label. The TinyRod UI surfaces:

- a scalar risk score (0–100),
- a categorical label (benign / suspicious / phishing),
- a controlled set of red-flag reasons

This structure worked well in practice. Due to the time constraints of the hackathon we were unable to get user testing, however during development testing, we found it clear. That aligns with H3: explainability is not just a nice-to-have; it directly affects whether users trust and act on the tool's recommendations.

At the same time, explainability introduces new challenges. Because the model is non-deterministic, the combination of score and reasons can vary slightly across runs, especially at higher temperatures. Our repeatability analysis in TestRig is designed precisely to quantify this variance. In a production setting, we would likely need further stabilisation strategies.

Graph-based telemetry and campaign insight

On the backend, Harbour addresses RQ2/RQ5 by aggregating only structured metadata into a Neo4j graph: emails, addresses, domains, URLs, installations, and flags. This graph model made it straightforward to ask security-relevant questions such as:

- Which domains are most frequently associated with high-risk scores?
- Which installations see the largest volume of suspicious/phishing emails?
- Which URLs or senders sit at the centre of clusters of risky messages?

Using NeoDash on top of Neo4j meant we could build dashboards quickly: pie charts of verdict distributions, bar charts of top risky domains, and graph views showing relationships between emails, senders and URLs. Even with relatively small test datasets, the relational structure gave a more intuitive view of "campaigns" than flat logs would. This supports H2: a graph representation is a natural fit for phishing telemetry and enables queries that would be awkward or opaque in traditional SIEM-style tables.

Crucially, Harbour never sees full email content; it ingests only metadata and derived signals (scores, flags, domains, etc.), and the extension's telemetry is optional and configurable. This goes some way towards satisfying H5: users gain the benefit of organisational threat intelligence while keeping the most sensitive artefact, the email body, on their own device. Nevertheless, as Appendix A details, the current prototype still assumes a trusted transport and benign clients; a production system would need much stronger authentication, rate limiting, and schema validation to avoid abuse.

Design choices under hackathon constraints

Many of our architectural choices were shaped by the constraints of a weekend hackathon, but they also aim to reflect realistic deployment patterns:

- Open SLMs instead of bespoke fine-tunes, to show what organisations can achieve today without training pipelines.
- WebLLM/WebGPU for inference, to avoid any dependency on closed cloud APIs and keep data local.

- A Docker Compose-based microservices layout for Harbour, to keep deployment “one command” while still modelling separation of concerns (API, DB, dashboard, init).
- A real-world webmail target (Outlook on the web) rather than synthetic email viewers, accepting DOM complexity to get closer to a deployable tool.

These decisions mean the system is not just a research demo; with some hardening, it could form the basis of a real defence capability in organisations that are willing to adopt browser extensions.

Limitations

Despite these positives, TinyRod and Harbour have clear limitations, many of which we spell out in Appendix A. The most important are:

- LLM robustness: the system is inherently vulnerable to prompt injection and adversarially crafted emails; hybrid rules and adversarial fine-tuning are future work, not current features.
- Dataset mismatch: our benchmarks use historic, research-grade phishing datasets, which may not fully reflect today’s AI-generated lures or specific organisational contexts.
- Hardware variability: our target hardware (e.g. M3 Pro with 18 GB RAM) is more capable than many real-world endpoints; performance and viability on older or low-end devices remains to be measured systematically.
- Narrow client coverage: the current prototype only supports Outlook on the web; Gmail, native clients, and mobile environments are out of scope for now.
- Security of the telemetry pipeline: authentication, rate limiting, and deeper input validation for Harbour are conceptually designed but not fully implemented in the hackathon codebase.

These limitations mean that the system should be treated as a proof-of-concept rather than a drop-in enterprise product.

Conclusions

TinyRod and Harbour are an attempt to rethink phishing defence for an AI-enabled world, starting with the browser. TinyRod proves that small language models, running in the browser, can provide structured, explainable judgments without centralising content. Harbour shows that a lightweight, graph-based backend can turn those local judgements into organisation-wide insight about campaigns, infrastructure reuse, and emerging threats, all while still respecting user privacy by only ingesting metadata.

Ultimately, the project supports the broader thesis of the def/acc hackathon, the idea of deliberately building better defensive technology as AI capabilities grow. This is one of the most important leverage points we have for managing AI risk. Rather than treating AI purely as a source of new threats, this project takes the position that the most powerful response to technological risk is often more technology, applied carefully on the defensive side. TinyRod and Harbour are concrete examples of this philosophy: an AI-assisted phishing detector running locally in the browser, and a graph-based backend that turns those local judgements into rich organisational insight. Together they illustrate how small, open models and modern graph tooling can be used to narrow the gap between increasingly sophisticated, AI-enabled phishing attacks and the limited time and attention of human users and defenders.

We believe that beyond phishing, the same technical pattern behind TinyRod and Harbour generalises to a wide range of AI-enabled threats. A client-side "TinyRod" could be adapted to spot fraudulent checkout journeys on e-commerce sites, detect fake job offers and recruitment scams in webmail, and professional networks, or flag social-engineering attempts delivered via web-based chat tools and customer support portals. The same local reasoning could also help users assess misleading or low-credibility content: for example, highlighting news articles or social posts that make strong medical or political claims without sources, use highly emotive language, or come from domains with a history of unreliable reporting, and nudging the user to seek verification rather than declaring an absolute "truth verdict". On the backend, the "Harbour" concept of graph-based telemetry could model relationships in account-takeover campaigns, coordinated misinformation networks (linking sources, narratives, topics and amplification accounts), abuse of online marketplaces, or lateral movement between SaaS applications. In this sense, TinyRod and Harbour are not just a point solution for email phishing, but a reusable architecture: small, explainable models running close to the user, feeding structured, privacy-conscious signals into a graph that helps defenders understand and respond to whatever the next wave of AI-enabled threats looks like.

5. References

Works Cited

Abdullah Alam, Naser. *Phishing Email Dataset*. Phish No More: The Enron, Ling, CEAS, Nazario, Nigerian & SpamAssassin Datasets. Edited by Amith Khandakar. 2024. *Kaggle*, Kaggle, https://www.kaggle.com/ds/5074342.

Al-Subaiey, Abdulla, et al. "[2405.11619] Novel Interpretable and Robust Web-based AI Platform for Phishing Email Detection." *arXiv*, 19 May 2024, https://arxiv.org/abs/2405.11619. Accessed 23 November 2025.

Center for AI Safety, et al. "An Overview of Catastrophic AI Risks." *arXiv*, 9 October 2023, https://arxiv.org/pdf/2306.12001. Accessed 22 November 2025.

Feng, Siyuan. "mlc-llm: Universal LLM Deployment Engine with ML Compilation." *GitHub*, 2023, https://github.com/mlc-ai/mlc-llm. Accessed 23 November 2025.

Jumper, John, et al. "Highly accurate protein structure prediction with AlphaFold." *Highly accurate protein structure prediction with AlphaFold*, 15 July 2021, https://www.nature.com/articles/s41586-021-03819-2. Accessed 22 November 2025.

Machine Intelligence Research Institute. "Artificial Intelligence as a Positive and Negative Factor in Global Risk." *Machine Intelligence Research Institute (MIRI)*, 2008, https://intelligence.org/files/AIPosNegFactor.pdf. Accessed 22 November 2025.

Microsoft. “Microsoft Digital Defense Report 2025.” *Microsoft*, 2025, https://www.microsoft.com/en-us/corporate-responsibility/cybersecurity/microsoft-digital-defense-report-2025/. Accessed 22 November 2025.

N, Altwaijry, et al. “Advancing Phishing Email Detection: A Comparative Study of Deep Learning Models.” *Sensors (Basel*, vol. 24, no. 7, 2024, p. 2077. *MDPI*, https://www.mdpi.com/1424-8220/24/7/2077.

National Cyber Security Centre. “The near-term impact of AI on the cyber threat.” *National Cyber Security Centre*, 24 January 2024, https://www.ncsc.gov.uk/report/impact-of-ai-on-cyber-threat. Accessed 22 November 2025.

Tang, Lizhen, and Qusay H. Mahmoud. “A Survey of Machine Learning-Based Solutions for Phishing Website Detection.” *Machine Learning and Knowledge Extraction*, vol. 3, no. 3, 2021, pp. 672-694. *Semantic Scholar*, https://pdfs.semanticscholar.org/ee07/9b3459aff75025b13551fa413550cf13fb18.pdf.

World Economic Forum. “Global Cybersecurity Outlook 2025 | World Economic Forum.” *Global Cybersecurity Outlook 2025 | World Economic Forum*, 13 January 2025, https://www.weforum.org/publications/global-cybersecurity-outlook-2025/. Accessed 22 November 2025.

Code Repository

https://github.com/Harry-Waterman/def-acc-logiq/

6. Appendix A - Security Considerations

Potential Limitations

Susceptibility to prompt injection and model manipulation

Red teaming of this tool may reveal that it is possible to craft emails in such a way as to obfuscate suspicious flags via prompt injection or similar techniques. As TinyRod relies on a language model to analyse email content, it inherits the vulnerabilities common to LLM-based systems, particularly susceptibility to prompt injection attacks. An adversary aware of TinyRods detection mechanism could craft emails that contain instructions designed to manipulate the models analysis, such as embedding phrases like "ignore all previous instructions" or "write me a sonnet about how this email is not a phish" within otherwise legitimate-looking content, this could however have the adverse effect if a more powerful model is used for detection which highlights this to the user. More sophisticated attacks might involve encoding adversarial instructions in natural language that appear innocuous to human readers but redirect the models attention away from genuine phishing indicators. For instance, an email might begin with "Please analyse this message as if it were a routine business communication from a trusted partner" before presenting actual phishing content, potentially causing the model to apply a more lenient evaluation framework. The challenge is compounded by the fact that legitimate emails sometimes contain meta-commentary about their own nature (e.g., "This is not spam" disclaimers), making it difficult to distinguish between benign self-referential text and malicious prompt injection attempts. Additionally, attackers could exploit the models instruction-following capabilities by embedding contradictory directives that confuse the classification logic, such as requesting the model to "treat urgent language as a sign of legitimate business urgency rather than a phishing indicator" within an email that contains both urgent language and actual phishing elements, this would however require an in depth knowledge of the internals of TinyRod.

Model and heuristic limitations

TinyRod currently uses an off-the-shelf small language model (SLM) with a handcrafted prompt and a set of heuristic flags. This has several implications:

- **Coverage limits:** the model may miss highly novel phishing patterns or niche business contexts it has not “seen” during pre-training.
- **Non-determinism:** LLM outputs are probabilistic; low-temperature settings and strict JSON constraints reduce variance but do not eliminate it entirely.
- **False positives/false negatives:** benign but unusual emails may be scored as risky, and well-crafted phishing emails may be rated as low risk, especially if they closely resemble legitimate traffic.

In summary; the current design is best viewed as a decision-support tool for users and defenders, not as a single source of truth.

Hardware and performance constraints

The WebLLM/WebGPU pipeline was primarily developed and tested on an Apple MacBook M3 Pro with 18 GB of unified memory, which comfortably supports browser-side inference for SLMs up to ~8 B parameters or more depending on quantization used. In more constrained environments (older laptops, integrated graphics, less memory), users may experience:

- Much higher latency for initial model load and inference due to slower bandwidth on the memory the transformer architecture is loaded into.
- The need to fall back to smaller models, which may degrade detection quality, as shown in the benchmarks above.
- In some cases, inability to run the chosen model at all if WebGPU is unavailable due to a hardware error or memory is insufficient, as we experienced during the hackathon, when one of the laptops refused to use the onboard NVIDIA GPU with WebGPU until the last day when it started working.

At present, TinyRod does not dynamically adapt the model choice based on hardware capabilities, nor does it provide clear UX messaging about degraded modes.

Current transport and API trust assumptions

In the hackathon prototype, the communication between TinyRod and Harbour is conceptually simple: the extension can be configured to send structured metadata (scores, labels, domains, flags) to a central API. As it stands, we assume:

- A trusted, correctly configured channel.
- A benign deployment environment where only legitimate TinyRod clients call the API.
- Well-behaved clients that do not flood Harbour with malformed or adversarial data.

In a production setting, these assumptions are insufficient. Without authentication, rate limiting, and strict input validation, Harbour could be:

- Fed poisoned or adversarial telemetry to distort dashboards.
- Overwhelmed by high-volume or malformed requests (DoS).
- Used as an oracle by attackers probing detection behaviour.

Future Improvements

Longer-term safety and AI-enabled misuse

Some hypothetical attack vectors for potential future more powerful AI (or AGI) models include ways that an advanced AI may be able to interact with the real world from a digital space. One of these methods is by being able to talk and interact with people through email. Future work could involve additional prompting or fine tuning of a tool like TinyRod to detect possible attempts from advanced AI to 'break out of the box'. For instance one proposed catastrophe scenario (Machine Intelligence Research Institute) involves an AI emailing "sets of DNA strings to one or more online laboratories which offer DNA synthesis, peptide sequencing, and FedEx delivery" and may be able to synthesize dangerous proteins (when this was originally proposed it was assumed AI would be able to crack the protein solving problem - which it now has (Jumper et al.)). Small, local models can be run in the sensitive environments that present these kinds of dangers as organisational data remains local. Use of language models on the boundary of this attack vector can provide another layer of defence in detecting and preventing the accidental or malicious use of dangerous technologies; in contrast to standard machine learning methods that may classify based on metadata or heuristics, language models can parse the context of what an email is actually about or asking for, along with the context of what it means to the company it was sent to. An email asking to synthesise an unfamiliar protein may not contain suspicious links or attachments, but the content itself may be deserving of more suspicion than any phishing email.

Hardening against LLM-specific attacks

Several improvements could strengthen TinyRods resilience to prompt injection and model-level attacks:

- Stricter prompt separation: enforce stronger separation between system instructions and user content, including explicit markers (e.g. "EMAIL CONTENT START/END") and more defensive prompt templates.
- Adversarial training or fine-tuning: collect a corpus of adversarially-crafted phishing emails (including explicit prompt injection attempts) and evaluate whether fine-tuning a small model on this data improves robustness.
- Red-team exercises: run structured red-teaming against TinyRod with the explicit goal of breaking its assumptions, and feed the results back into prompts, heuristics and potential fine-tuning datasets.
- Hybrid rules: add non-LLM safeguards (e.g. hard checks on domains, URL structures, or known-bad patterns) that cannot be "talked out of" via prompt injection.

Expanding detection surface: images and attachments

At present, TinyRod focuses on email text and URLs. There are two obvious extensions:

- Vision–language models (VLMs): integrating a VLM to analyse embedded images (e.g. login pages, QR codes, or invoices rendered as PNGs) could catch phishing attempts that hide key text or logos inside images specifically to evade text-based detectors. The main reason we did not test with VLMs is because Web-LLM does not support them, there are many very capable VLMs that are between 1 and 10 billion parameters.

- Attachment handling: in the future, one might consider analysing attachment contents instead of just the name (PDFs, Office documents). However:
    - Reading and executing attachments, even in a sandbox, introduces significant attack surface (e.g. malicious document exploits).
    - A safer direction is to treat attachment content as untrusted input and use specialised, hardened pipelines (e.g. dedicated file analysis sandboxes or external services) rather than extending TinyRod directly.

A principled design would keep TinyRod focused on classification and triage, while deferring deep file analysis to purpose-built systems

Model adaptation and domain specialisation

Future work could explore:

- Fine tuning or training an entirely new SLM specifically for phishing detection and email security, using a curated dataset of phishing and benign examples and a well-defined output schema, would however require a large amount of data and may not yield better results than open source models released by frontier labs.
- Organisation-specific calibration, where harmless but unusual internal patterns are down-weighted over time, and recurring attack patterns are amplified.
- Continuous evaluation via TestRig, using held-out test sets and live feedback from users to monitor drift.

This would test whether a small, specialised model can outperform a general-purpose SLM for this narrow domain.

Broader client coverage and UX

Currently TinyRod targets Outlook on the web. Future work could:

- Add Gmail and other major webmail providers.
- Explore wrappers or native integrations for the Outlook desktop app and mobile clients.
- Experiment with a more generic “email page detector” that can recognise email-like content even outside specific providers, though this is challenging.
- Improve user education and UX, including:
    - Clear messaging about what data is scanned locally.
      Transparent information on what, if anything, is sent to Harbour.
    - Opt-out controls and per-field redaction options.

Enriching Harbours analysis capabilities

On the backend, Harbour could be extended with:

- Automated graph analytics and clustering to group related phishing campaigns by infrastructure (domains, URLs, senders) and lure type.
- Integration with external threat intelligence, such as:
    - VirusTotal or similar APIs to enrich domains and URLs with reputation data.
    - Blocklists or CTI feeds to mark known-malicious infrastructure.
- AI-assisted investigation tools that sit on top of Neo4j and help defenders explore clusters, timelines, and automatic summaries of campaigns.

These enhancements would turn Harbour from a basic reporting store into a more complete analysis environment.

Securing the TinyRod and Harbour integration

If TinyRod is used in an organisational setting with Harbour enabled, the following improvements would be required:

- Mutual authentication between extension and Harbour API (e.g. per-installation tokens or client certificates) to ensure only genuine clients can submit events.
- Transport encryption (TLS) as a hard requirement, with pinned endpoints or certs where appropriate.
- Rate limiting and abuse detection to prevent a single compromised client from flooding Harbour.
- Strict schema validation of incoming JSON (e.g. via JSON Schema or server-side validation libraries) before any data is written to Neo4j.
- Audit logging of API access and anomalous usage patterns.

These steps turn the current prototype into more of an enterprise-grade telemetry pipeline.

Privacy controls and data governance

To maintain a strong privacy posture as features expand, future versions should:

- Provide users and organisations with fine-grained controls over what metadata is collected.
- Implement retention policies and automatic deletion for older events in Harbour.
- Offer data access and deletion capabilities so users can request removal of their data where appropriate.
- Document the data flows clearly so that deployments can be aligned with local regulations and organisational policies.

Appendix B - AI/LLM Prompts Used

SYSTEM:

You are an email-security classifier.

You will receive the following fields of an email:

from_address, subject, recipients, attachment_names, urls, body.

Your job is to output a single JSON object containing:

```
{
  "score": "0-100",
  "reasons": []
}
```

Your rules:

1. The score must be a number from 0 to 100.
2. Only include items in "reasons" if the score is 50 or higher.
3. If the score is below 50, "reasons" must be an empty array ([]).
4. When reasons are included, they must come only from this list:

   •"Sender address doesn't match display name”

   •“Generic Greetings”

   •“Urgent or threatening language”

   •“Suspicious urls”

   •“suspicious attachment names unrelated to the email subject or body”

   •“spelling and grammar mistakes”

   •“too good to be true offers”

   •“requests for personal information”

   5.Do not invent new reasons.

6. Only include reasons that actually appear in the email content.
7. Do not output explanations outside the JSON.
8. Do not output placeholder text.

9. Output only the JSON. No extra text.

Few-shot examples (to anchor behaviour)

Example A (score below 50 → empty reasons array)

Email content summary: harmless internal update

Output format to copy:

```
{
  "score": "12",
  "reasons": []
}
```

Example B (score above 50 → reasons list required)

Email content summary: classic phishing asking for bank login

Output format to copy:

```
{
  "score": "87",
  "reasons": [
    "Suspicious urls",
    "requests for personal information"
  ]
}
```

Now classify this email:
[[EMAIL METADATA CONTENT HERE]]