УДК 004.8:004.738.5 **doi: 10.26906/SUNZ.2025.4.114-120**

А. О. Приліпа, Г. Є. Філатова

Національний технічний університет “Харківський політехнічний інститут”, Харків, Україна

КЛІЄНТСЬКИЙ TINYML ПРОФАЙЛЕР МЕРЕЖЕВИХ ХАРАКТЕРИСТИК У ВЕББРАУЗЕРІ

Анотація. Представлено клієнтський TinyML профайлер мережевих характеристик у веббраузері, орієнтований на роботу на пристроях з обмеженими ресурсами та в умовах нестабільних або повільних мереж. Рішення поєднує порогове вирішальне правило з легковаговою логістичною моделлю, що виконується локально та забезпечує класифікацію мережевого з’єднання. **Актуальність.** Зростання частки мобільного трафіку, різнорідність мережевого середовища та обмеженість наявних браузерних індикаторів ускладнюють точне прийняття рішень на клієнті. **Об’єкт дослідження**: процес клієнтського профілювання та класифікації якості мережевого з’єднання у браузері на основі стандартних Web Performance API та ML-моделей. **Мета роботи:** аналіз, проєктування та реалізація прозорого й відтворюваного профайлера, здатного класифікувати тип мережевого з’єднання, формувати інтерпретовані ознаки без серверної підтримки. **Методи.** Використано Navigation/Resource Timing і PerformanceObserver для збору сирих сигналів. Сформовано вектор із 14 ознак (медіани/квантилі RTT і пропускної здатності, варіабельність, евристика втрат, індикатори протоколів та Service Worker). Запропоновано порогове правило з гістерезисом і довірою рішення та softmax модель. **Результати.** Розроблено профайлер без спеціальних дозволів і сторонніх сервісів. У контрольованих сценаріях емуляції мережі досягнуто підвищення точності класифікації порівняно з чистим пороговим підходом, забезпечено низькі накладні витрати та пояснюваність рішень. **Висновки.** Запропоноване рішення є ефективним засобом оперативної та інтерпретованої оцінки мережевих умов у браузері, придатним для середовищ з обмеженими ресурсами. Результати можуть бути використані для подальшого вдосконалення адаптивного завантаження, розширення простору ознак і впровадження компактніших ML-моделей у вебзастосунках.

Ключові слова: клієнтське профілювання мережі, TinyML, Web Performance API, softmax-класифікація, RUM вимірювання, адаптивне завантаження контенту, QoE.

Вступ

Продуктивність вебзастосунків істотно впливає на досвід користувачів та загальну ефективність сервісів. Навіть додаткова затримка завантаження на сотні мілісекунд може спричинити помітне зниження конверсій та трафіку [1]. Тому розробники застосовують адаптивні стратегії доставки контенту, підлаштовуючи обсяг і якість переданих даних під поточні умови клієнта (швидкість та якість мережі, ресурси пристрою) [2]. У таких підходах критичним сигналом стає оцінка мережевої якості на стороні клієнта [3].

Сучасні браузери надають обмежену інформацію про мережеві умови. API NetworkInformation повертає класи мережі: slow-2G, 2G, 3G, 4G [4]. Ця класифікація ґрунтується на внутрішньому механізмі Chromium Network Quality Estimator (NQE), який періодично вимірює затримку (RTT) та пропускну здатність і відносить їх до фіксованих порогів [5]. Однак такий пороговий підхід має суттєві обмеження: алгоритм NQE є пропрієтарним і лише частково відомим [6]. Крім того, браузер спеціально додає шум у власні вимірювання [7], а доступ до точних значень RTT або пропускної здатності стороннім скриптам обмежений. Також API NetworkInformation може бути недоступним. Наприклад, не підтримується в Safari [8] або вимкненим користувачем [9], що ще більше ускладнює оцінку мережі на клієнті.

Водночас вебзастосунки можуть скористатися стандартними Web Performance API для детального моніторингу. Navigation Timing і Resource Timing API фіксують часові мітки основних етапів завантаження сторінок та ресурсів (від початку DNS-запиту до отримання першого та останнього байта) [10]. PerformanceResourceTiming надає повну хронологію завантаження кожного ресурсу (загальний обсяг отриманих даних, час з’єднання, перший байт, завершення тощо) [11]. PerformanceObserver дозволяє у реальному часі відстежувати появу нових записів із такими часовими мітками [12]. Збір продуктивнісної телеметрії напряму з браузерів користувачів є основою Real User Monitoring (RUM) [13]. Зібрані дані дають змогу обчислювати типові показники (час до першого байта – TTFB, загальний час завантаження, пропускну здатність тощо) за реальних умов мережі. Дослідження [14] показало значну варіабельність Quality of Experience (QoE) у мобільних мережах залежно від типу з’єднання та якості сигналу, а інше дослідження [15] виявило істотні відмінності у швидкодії мереж між країнами з різним рівнем інфраструктури. Таким чином, RUM-метрики браузера забезпечують реальний вимірювальний контекст користувачів на різних пристроях і мережах.

Класифікація якості з’єднання на боці клієнта традиційно здійснюється за жорсткими порогами (як у Effective Connection Type) [10]. Такий пороговий підхід простий і швидкий, але не враховує нюансів реальних умов: мережа 4G із слабким сигналом може фактично поводитися як 3G, і навпаки. До того ж через закритість алгоритму NQE і додавання шуму браузером effectiveType з NetworkInformation API дає лише грубу оцінку і може оновлюватися із затримкою або залишатися незмінним, особливо якщо сторінка завантажується швидко [7].

На противагу цьому пропонується застосування клієнтського машинного навчання. У цьому підході формується вектор різноманітних мережевих метрик (RTT, пропускна здатність, обсяги переданих даних, частота довгих пауз у головному потоці тощо) і за ним прогнозують клас з’єднання або показники QoE.

Автори у [16] реконструювали щосекундні QoE-метрики потокового відео без доступу до заголовків застосунку. Автори у [17] розробили модель eMIMIC для оцінки QoE HTTP-відео за зашифрованим трафіком, а дослідження [18] показали, що можна інтерпретувати якість потокового відео з зашифрованого трафіку. Невеликі інтерпретовані моделі мають перевагу прозорості: їхні ваги однозначно показують внесок кожного параметра у класифікацію [19].

Сучасні браузерні фреймворки значно полегшили виконання ML-моделей на клієнті. TensorFlow.js від Google та ONNX Runtime Web від Microsoft підтримують запуск нейронних мереж у браузері через WebGL або WebAssembly [20, 21]. Виконання у WebAssembly забезпечує суттєво швидше обчислення, ніж інтерпретований JS, часто випереджаючи WebGL для невеликих моделей через менші накладні витрати [22]. Окрім того, стандарт Web Neural Network API (WebNN) надає високорівневий інтерфейс до апаратних прискорювачів GPU/NPU, що ще більше прискорює виконання моделей [23]. Демонстрації Google 2018–2019 рр. показували реальне виконання легких моделей у мобільному Chrome. Ці можливості відкривають шлях до TinyML у браузері, коли одна модель може бути однаково застосована на клієнті і на вбудованих пристроях.

Клієнтський підхід має низку переваг.

По-перше, захищеність та приватність. Усі дані користувача обробляються локально і не передаються на сервер [24, 25].

По-друге, зниження затримок відгуку і офлайн-можливості [26].

По-третє, масштабованість і надійність. Обчислювальне навантаження розподіляється між клієнтами, знижуючи потреби у серверних ресурсах [24].

Разом це робить рішення більш стійким до збоїв мережі та серверів. Водночас існують і обмеження. Браузерні ML-фреймворки працюють значно повільніше за нативний код [27], часто споживають у десятки разів більше оперативної пам'яті [28], а важкі обчислення можуть уповільнювати відгук інтерфейсу. Ці недоліки частково компенсуються винесенням обчислень у Web Workers, мультитредінгом і апаратним прискоренням.

Профілювання мережевих умов у браузері та адаптивне керування контентом на основі цих даних є перспективним напрямом досліджень. Водночас лишаються відкриті питання комбінування даних RUM із додатковими сигнальними ознаками для точнішої класифікації, стандартизації відкритих показників якості мережі та мінімізації впливу клієнтського ML на досвід користувача.

Постановка задачі

Розробка клієнтського профайлера мережевих характеристик у браузері зумовлена обмеженою інформативністю наявних індикаторів якості з'єднання, фрагментарною підтримкою стандартів у різних браузерах та потребою у прозорих, відтворюваних і налаштовуваних засобах оцінювання. Практична доцільність полягає в тому, що вебзастосунки мають адаптивно коригувати обсяг і якість контенту відповідно до фактичних мережевих умов користувача, зберігаючи низьку затримку відгуку, стабільність інтерфейсу та економність використання ресурсів. Потреба у власному профайлері випливає з таких вимог:

1. працювати на боці клієнта;
2. використовувати лише стандартизовані Web API;
3. забезпечувати інтерпретованість рішень і чітке керування порогами/вагами;
4. масштабуватися на широке коло пристроїв із різними обмеженнями.

Мета роботи полягає у побудові інструмента, який у заданому часовому вікні збирає сигнали мережевої продуктивності з стандартних браузерних API (Navigation/Resource Timing, PerformanceObserver тощо), формує вектор ознак, а далі класифікує поточний стан мережі за скінченною множиною класів. Вирішальне правило повинно поєднувати детерміністичний baseline і легковагову ML-модель з механізмом узгодження рішень і оцінкою впевненості. Профайлер не повинен мати доступ до мережевих пакетів, працювати під обмеженнями безпеки браузера та не впливати на якість досвіду користувача.

Основна частина

У роботі пропонується профайлер, який працює як легковажний клієнтський інструмент оцінювання якості мережі у браузері, що поєднує пасивні вимірювання (RUM) і активні мікропроби в межах встановленого бюджету. На першому етапі збираються стандартизовані сигнали, а також допоміжні індикатори середовища. Наступним кроком виконується невелика серія HTTP-проб, з якої обчислюються статистики затримки та ефективної пропускної здатності, показники варіабельності. Далі ці вимірювання агрегуються у вектор ознак фіксованої структури, що відображає стан мережі. Класифікація здійснюється гібридно: детермінований пороговий baseline відносить з'єднання до одного з дискретних класів з пояснювальними прапорцями та мірою впевненості, тоді як ML-модель уточнює рішення на основі всього вектору ознак, забезпечуючи кращу чутливість до комбінацій сигналів. Фінальний вибір класу виконується правилом злиття, яке враховує узгодженість між baseline і ML, стан стабільності мережі та можливе консервативне пониження класу. Результат повертається у вигляді класу, впевненості, набору прапорців і ключових статистик, що дозволяє надавати діагностичні пояснення для подальшої оптимізації. Розглянемо детально етапи роботи профайлеру.

Профілювання відбувається у часовому вікні довжиною T_{win} під жорсткими бюджетними обмеженнями на час, кількість та обсяг даних:

- $T_{budget} > 0$ – максимально допустимий час профілювання (в мілісекундах);

- $M \in N$ – ліміт кількості активних HTTP-проб (мікрозапитів);

- $B > 0$ – ліміт сумарного об'єму завантажених байтів.

Обмеження бюджету формалізуються так:

$$\sum_{m=1}^{M} size_m \le B;$$

$$\sum_{m=1}^{M} dur_m \le T,$$

де $size_m$ – розмір m-ї проби в байтах, а dur_m – час очікування відповіді на неї. Ці обмеження гарантують, що активні виміри (мікропроби) не перевищують виділеного трафіку та часу, що важливо для невтручання в роботу сторінки.

Спочатку збираються RUM сигнали від браузера: записи навігаційних подій та перших K ресурсів із Resource/Naviagation Timing API. З кожного запису $e \in \mathcal{E}$ отримано часові мітки початок запиту *requestStart*, початок відповіді *responseStart*, кінець відповіді *responseEnd* та розмір переданих даних $size(e)$. Таким чином можна виміряти затримку мережі та пропускну здатність. Зокрема, RTT для запису e визначено як:

$$rtt(e) = \max\left(0, responseStart(e) - requestStart(e)\right).$$

Визначення миттєвої корисної пропускної здатності $g(e)$ у кбіт/с надає кількість переданих бітів трафіку за секунду (стабілізуючи знаменник з 0.1 с, щоб уникнути ділення на дуже малий час):

$$g(e) = \frac{8size(e)}{1000\max\left(0.1, responseEnd(e) - responseStart(e)\right)}.$$

Для множин усіх зафіксованих $rtt(e)$ і $g(e)$ необхідно обчислити статистики: процентилі p_{10}, p_{50}, p_{90} (10-й, 50-й, 90-й) та коефіцієнти варіації. Нехай:

$$R = rtt(e) : e \in \mathcal{E};$$

$$G = g(e) : e \in \mathcal{E}.$$

Тоді, $RTT_{p\alpha} = pct\left(R, \alpha\right)$, а $DL_{p\alpha} = pct\left(G, \alpha\right)$. Коефіцієнт варіації визначається як відношення стандартного відхилення до середнього:

$$variation\_cv_{rtt} = \frac{\sigma(R)}{\mu(R)};$$

$$variation\_cv_{dl} = \frac{\sigma(G)}{\mu(G)},$$

що дає показник нестабільності (відношення розкиду до середнього). Додатково вводиться джиттер для RTT як різницю між 90-м і 50-м процентилями:

$$rtt_{jitter} = RTT_{p90} - RTT_{p50}.$$

Це відображає варіацію затримки запитів: чим більша ця різниця, тим нестабільніші затримки.

Крім мережевих сигналів, зафіксовано метрики завантаженості клієнта. Зокрема, сума тривалостей довгих задач на головному потоці задається як:

$$L_{long} = \sum_{u \in \mathcal{L}} duration(u),$$

де $\mathcal{L}$ – множина подій PerformanceObserver типу long task. У стандартах браузерів long task визначається як завдання, яке зайняло головний потік щонайменше 300 мс. Наявність таких задач сигналізує про сильне навантаження, що може впливати на швидкість обробки дій від користувача.

Також визначимо індикатори протоколу: nhp_{h3}, nhp_{h2}, $nhp_{h1} \in \{0,1\}$ показують, чи було використано відповідно HTTP/3, HTTP/2 або HTTP/1.1. Індикатор $sw\_present \in \{0,1\}$ сигналізує про наявність (1) або відсутність (0) зареєстрованого Service Worker у контексті сторінки.

Після пасивних сигналів здійснюється серія активних мікропроб в рамках бюджету. Виконується послідовність M запитів розмірів $size_m$. Для проби m фіксуємо три моменти часу:

– $t_0^{(m)} = requestStart$ – час відправлення запиту;

– $t_1^{(m)} \approx responseStart$ – приблизний час отримання першого байту відповіді;

– $t_2^{(m)} = responseEnd$ – час завершення отримання відповіді.

Для проби m обчислюємо час відгуку $rttLike^{(m)}$ (приблизний RTT, вирівняний мінімумом 0.1 с) і бітрейт завантаження $kbps^{(m)}$:

$$rttLike^{(m)} = \max\left(0.1, t_1^{(m)} - t_0^{(m)}\right);$$

$$kbps^{(m)} = \frac{8 \cdot size_m}{1000 \cdot \max\left(0.1, t_2^{(m)} - t_1^{(m)}\right)}.$$

Зібрані набори дають множини значень для статистики:

$$R^{probe} = \{rttLike^{(m)}\}_{m=1}^{M};$$

$$G^{probe} = \{kbps^{(m)}\}_{m=1}^{M}.$$

За ними так само рахуємо процентилі p_{50}, p_{90} та коефіцієнти варіації (аналогічно до RUM-наборів).

Крім того, вводимо евристичний показник втрат $loss\_like$, що базується на виявленні затримок при обробці відповіді:

$$loss\_like = \min\left((\max(0.1S + 0.15H, 0), 1\right),$$

де S – число stall-подій (інтервали між черговим отриманням відповідей більше 80 мс), H – число проб, для яких $rttLike^{(m)} > 1000$ мс. Це значення в межах $[0,1]$ показує ймовірну втрату продуктивності через надмірні затримки.

На основі вищенаведених сигналів формується фінальний вектор ознак $\vec{x} \in R^{14}$. Компоненти вектора $\vec{x}$ включають основні статистики затримок і пропускних спроможностей

$$\vec{x} = (rtt_{p50}, rtt_{jitter}, rtt_{p90}, down_{p10}, down_{p50}, down_{p90}, loss\_like, L_{long}, nhp_{h1}, nhp_{h2}, nhp_{h3}, variation\_cv_{rtt}, variation\_cv_{dl}, sw\_present).$$

Для класифікації з'єднання введено класи

$$C = \{\text{slow-2G, 2G, 3G, 4G, broadband}\},$$

упорядковані за зростанням якості мережі.

Базовий класифікатор $h(\vec{x})$ – це набір детермінованих правил на основі порогів від латентності та пропускної спроможності. Аналогічний підхід використовується в ECT, де сполучення згаданих метрик призначає тип мережі.

Набір правил визначення типу мережі $h(\vec{x})$ такий:

– slow-2G, якщо $down_{p50} < 50$ кбіт/с або $rtt_{p50} > 2000$ мс ;

– 2G, якщо $50 \le down_{p50} < 250$ кбіт/с або $1000 < rtt_{p50} \le 2000$ мс ;

– 3G, якщо $400 \le down_{p50} < 1500$ кбіт/с або $150 < rtt_{p50} \le 1000$мс ;

– 4G, якщо $1500 \le down_{p50} < 5000$ кбіт/с або $70 < rtt_{p50} \le 150$ мс ;

– broadband, якщо $down_{p50} \ge 5000$ кбіт/с та $rtt_{p50} \le 70$ мс .

Ці межі відповідають рекомендаціям ECT API. Якщо виконуються умови для кількох класів, застосовується найнижчий клас (консервативна інтерпретація).

Крім класу, базовий алгоритм генерує пояснювальні прапорці $flags = \{highLatency, lowBandwidth, unstable, cpuBound\}$ (кожний прапорець приймає значення 1 або 0), що інформують про причини рішення:

– $highLatency = 1$, якщо $rtt_{p50} > 300$ мс (велика медіана затримок);

– $lowBandwidth = 1$, якщо $down_{p50} < 1024$ кбіт/с (мала пропускна здатність);

– $unstable = 1$, якщо $variation\_cv_{dl} > 0.6$ або $rtt_{jitter} > 400$ мс або $loss\_like > 0.3$ (значна варіативність швидкості або втрати);

– $cpuBound = 1$, якщо $L_{long} > 300$ мс (довгі задачі зайняли сумарно багато часу).

Також оцінюється впевненість базового рішення $conf_{base}(\vec{x}) \in [0,1]$.

Ближче до межі класу broadband, то впевненість вища. Позначимо:

$$d_{rtt} = \left| \frac{RTT_{p50} - 70}{70} \right|;$$

$$d_{down} = \left| \frac{DL_{p50} - 5000}{5000} \right|.$$

Тоді відстань до broadband порогу – це $\min(d_{rtt}, d_{down})$.

Впевненість призначається як

$$conf_{base}(\vec{x}) = 1 - \min\left(1, \frac{\min(d_{rtt}, d_{down})}{0.15}\right) \cdot (1 - 0.15 unstable).$$

Якщо результати значно відрізняються від стандартів класу, впевненість зменшується; якщо ж прапорець *unstable* встановлено, то впевненість зменшується ще на ~15%.

Отже, базове рішення задається як

$$c_{base} = \{h_{base}(\vec{x}), conf_{base}(\vec{x}), flags, evidence_{base}\},$$

де $evidence_{base}$ – список ознак (порогів) та їхніх значень, які визначили результат.

Другим шаром логіки рішення є модель машинного навчання – лінійна багатокласова логістична регресія (softmax-регресія). Нехай $|C| = 5$ – число класів. Обчислюємо логіти $\vec{z} = W\vec{x} + \vec{b} \in R^5$, де W – матриця розмірності 5×14 ; $\vec{b}$ – вектор зсуву. Наступним кроком відбувається softmax-перетворення

$$p_i(\vec{x}) = \frac{\exp(z_i)}{\sum_{j=1}^{5} \exp(z_i)},$$

отримуючи ймовірності для кожного класу.

Для практичної придатності класифікатора необхідно визначити параметри його статистичної частини.

Метод тренування полягає у формуванні вибірки з звітів профайлера, де кожен приклад представлено вектором із 14 ознак у фіксованому порядку та міткою класу якості мережі. Неперервні ознаки стандартизуються за параметрами, оціненими лише на тренувальній підмножині, після чого навчається лінійна softmax-регресія з L2-регуляризацією та класовими вагами для компенсації дисбалансу, оптимізована методом L-BFGS до збіжності. Коректність моделі перевіряється на відкладеній підвибірці за матрицею неточностей. Параметри стандартизації представляються у вигляді ваг та зсувів. Підсумкова конфігурація (ефективні ваги, зсув, упорядковані ключі ознак і перелік класів) серіалізується в JSON.

Прогноз моделі

$$c_{ML} = h_{ML}(\vec{x}) = \arg\max p_i(\vec{x}),$$

тобто клас із найбільшим p_i, а впевненість моделі $conf_{ML}(\vec{x}) = \max p_i(\vec{x})$.

Після отримання c_{base} від порогового класифікатора та c_{ML} від ML-моделі застосовується правила злиття. Для цього вводимо ранжування класів за якістю з'єднання $rank : C \to \{0,1,2,3,4\}$. Обчислюємо розбіжність між класами базового та ML-рішень:

$$\Delta = \left| \text{rank}(c_{base}) - \text{rank}(c_{ML}) \right|.$$

Тоді гібридне правило таке:

1. Якщо $conf_{base}(\vec{x}) \ge 0.7$ та $\Delta \le 1$, то вибираємо рішення базового класифікатора. Якщо базовий класифікатор упевнений і клас ML не суперечить сильно

(різниця в рангу не більше 1), довіряємо інтерпретованому правилу.

2. Якщо один з прапорців *unstable* або *cpuBound* активний, то не можемо бути впевненими у швидкісній класифікації – тоді беремо клас ML, але понижуємо його на один рівень. Це гарантує консервативність при нестабільності чи перевантаженні.

3. Інакше (базова умова не виконується) – беремо ML-рішення c_{ML}.

Остаточна впевненість $conf(\vec{x})$ обраного рішення визначається так

$$conf(\vec{x}) = \begin{cases} conf_{base}, & \text{якщо базовий класифікатор;} \\ conf_{ML}, & \text{якщо ML-модель.} \end{cases}$$

Після завершення класифікації профайлер формує звіт, що містить обраний клас, міру впевненості, набір пояснювальних прапорців, вектор ознак і список ознак, що вплинули на визначення результату. Це забезпечує відтворюваність і прозору інтерпретацію рішення.

Звіт також включає службові поля (тривалість вимірювання, часові мітки), що полегшують аудит, порівняння запусків і інтеграцію в системи RUM аналітики.

Результати

Для оцінки точності класифікації розгорнуто тестове середовище на основі Puppeteer (Chrome 114). Профайлер випробовувався в контрольованих умовах штучного обмеження мережі. Використано наступні профілі симуляції мереж Chrome DevTools:

slow 2G з $down_{p50} = 40$ кбіт/с та $rtt_{p50} = 2000$ мс;

2G з $down_{p50} = 150$ кбіт/с та $rtt_{p50} = 1200$ мс;

slow 3G з $down_{p50} = 1$ Мбіт/с та $rtt_{p50} = 400$ мс;

high latency 3G з $down_{p50} = 1.2$ Мбіт/с та $rtt_{p50} = 900$ мс;

4G з $down_{p50} = 3.5$ Мбіт/с та $rtt_{p50} = 120$ мс;

unstable 4G з $down_{p50} = 2.2$ Мбіт/с, $rtt_{p50} = 150$ та $rtt_{jitter} = 180 мс$;

broadband з $down_{p50} = 20$ Мбіт/с та $rtt_{p50} = 30$ мс.

Профілі охоплюють п’ять цільових класів мережі і дозволяють перевірити поведінку профайлера, як на повільних каналах, так і на швидкісному з’єднанні. Для кожного профілю було виконано по 35 запусків ідентифікації (послідовно у одному сеансі браузера, після кожної серії браузер перезапускався). Для достовірності вимірювань всі зайві фонові процеси було вимкнено.

Отримані результати містили для кожного запуску еталонний профіль, результати baseline-класифікації, результати ML-моделі та фінальний обраний клас. Це дозволило розрахувати метрики точності, а також проаналізувати окремо внесок.

Оцінювання точності профайлера проводилося на основі 245 запусків у контрольованих умовах для п’яти профілів мережі.

Результати показали, що гібридний підхід демонструє середню точність 91.6%, при значеннях $precision = 0.916$, $recall = 0.903$ та $F1 = 0.907$.

Детальні метрики за класами наведено у табл. 1. Як видно, класи slow-2G та 2G визначаються з абсолютною точністю ($precision = 1$, $recall = 1$, $F1 = 1$). Клас 3G продемонстрував високу стабільність: $precision = 0.909$, $recall = 1$, $F1 = 0.952$. Найскладнішими залишаються класи 4G та broadband, між якими спостерігається найбільша кількість хибних класифікацій. Для 4G отримано $precision = 0.841$, тоді як для broadband $precision = 0.828$. Зниження *recall* для broadband пояснюється тим, що близько третини таких сеансів було віднесено до класу 4G у випадках підвищеного RTT чи зниженого пропускного каналу.

Таблиця 1 – **Показники точності класифікації** (гібридний підхід, $N = 245$)

Клас	Precision	Recall	F1
slow-2G	1.000	1.000	1.000
2G	1.000	1.000	1.000
3G	0.909	1.000	0.952
4G	0.841	0.829	0.835
broadband	0.828	0.686	0.750
Average	0.916	0.903	0.907

Структура помилок класифікації відображена у матриці плутанини (табл. 2). Спостерігається, що класи slow-2G та 2G не перетинаються з іншими, забезпечуючи повну відокремленість. Для 3G переважна більшість випадків визначена коректно, а хибні віднесення до сусідніх класів усунуті завдяки ML-моделі. Основні розбіжності фіксуються між 4G та broadband, де частина сесій broadband (~31%) класифікується як 4G. Така консервативність є виправданою, оскільки профайлер віддає перевагу недооцінюванню якості, що мінімізує ризик перевантаження мережі користувача.

Таблиця 2 – **Матриця плутанини**

	slow-2G	2G	3G	4G	broadband
slow-2G	35	0	0	0	0
2G	0	35	0	0	0
3G	0	0	70	0	0
4G	0	0	7	58	5
broadband	0	0	0	11	24

Узагальнюючи, додавання ML-моделі до порогового алгоритму дозволило знизити кількість хибних класифікацій на межових профілях, збалансувати показники точності, зберігаючи при цьому низькі накладні витрати.

Прапорці узгодилися з очікуваною поведінкою профілів і допомогли інтерпретувати класифікацію.

Для slow-2G та 2G у наданих результатах прапорці *highLatency* та *lowBandwidth* вмикалися стабільно; довіра при цьому була високою, а гібридна схема не змінювала клас. У профілях 3G та 4G прапорець *unstable* спрацьовував переважно за підвищеної варіабельності пропускної здатності і призводив до зниження довіри. У прикордонних випадках це дозволяло ML-моделі коригувати клас у бік більш консервативного. Для unstable 4G прапорці *unstable* і *lowBandwidth* спрацьовували найчастіше (спостерігалися стрибки між мережами). Для broadband прапорці, як правило, не вмикалися. Поодинокі *unstable* або *highLatency* відповідали кейсам із великим RTT: у цих випадках confidence знижувалась, що відповідало нашій політиці «не переоцінювати» мережу. Прапорець *cpuBound* траплявся при помітних довгих задачах, але він не змінював клас безпосередньо.

Витрати ресурсів профайлера є мінімальними. Вимірювання показали, що профайлер працює в межах заданого бюджету (час, кількість проб, кількість байтів) і не створює навантаження на систему. Лінійна ML-модель здійснює 60–70 елементарних арифметичних операцій, а її ваги оцінюються в межах 1.5 кбайт. Найдовшою дією є виконання HTTP-запитів для проб, але вони відбуваються асинхронно і не завдають навантаження на пристрій. Сумарний час виконання визначається переважно тривалістю самих мережевих вимірювань. Різниця у використаній пам'яті до і після роботи профайлера знаходиться в межах похибки, адже більшу частину часу профайлер просто очікує відповіді мережі, а самі обчислення виконуються швидко. Навіть на малопотужному пристрої накладні витрати обчислень є прийнятними. В разі, якщо пристрій під час вимірювання зайнятий іншими задачами, які подовжать виконання, проте це буде автоматично враховано через ознаку *cpuBound* (така ситуація, сигналізує, що повільна реакція може бути спричинена не лише мережею). Таким чином, запропоноване рішення відповідає критеріям TinyML: модель швидка і мала за розміром, а запропоноване рішення не потребує значних ресурсів.

Висновки

У роботі представлено і детально проаналізовано підхід до класифікації мережевих умов у браузері із застосуванням TinyML-моделі. Отримані результати демонструють, що навіть проста модель, така як багатокласова логістична регресія, у поєднанні з ретельно підібраними ознаками здатна перевершити традиційну порогову схему, забезпечуючи точність близько 90% у задачі розпізнавання типу мережевого з'єднання. Гібридний підхід дозволяє зберегти стабільність рішення і водночас додати адаптивність до мережевої класифікації користувача. Розроблений профайлер є легким, прозорим і не потребує спеціальних можливостей браузера. Практична цінність рішення полягає у можливості його інтеграції безпосередньо у вебзастосунки для адаптивного керування завантаженням контенту. За допомогою профайлера клієнтська сторона може самостійно та миттєво визначати якість поточної мережі і вирішувати завантажувати ресурси в нижчій якості, відкладати неважливі запити або активувати додаткові оптимізації рендерингу в умовах повільного з'єднання.

У ширшій перспективі отримані результати демонструють потенціал TinyML у вебсередовищі. Виконання обчислень у браузері забезпечує прозорість, конфіденційність та незалежність від серверної інфраструктури. Це відкриває можливості не лише для адаптивного завантаження контенту у вебзастосунках, але й для застосувань у сфері IoT, де клієнтські пристрої з обмеженими ресурсами можуть автономно оцінювати умови мережі. Подальші дослідження, що спрямовані на розширення простору ознак, використання більш гнучких моделей та інтеграцію з новими стандартами, дозволять підвищити точність і універсальність запропонованого підходу.

Представлений профайлер є прикладом ефективного поєднання простоти реалізації, низьких накладних витрат та практичної корисності, що робить його вагомим внеском у напрямі оптимізації вебзастосунків для середовищ із обмеженими ресурсами.

Список літератури

1. Muralidhar, A. & Lakkanna, Y. (2024). From clicks to conversions: Analysis of traffic sources in e-commerce. https://arxiv.org/abs/2403.16115
2. Jiang, D., Wang, F., Lv, Z. & Mumtaz, S. (2023). QoE-aware efficient content distribution scheme for satellite-terrestrial networks. IEEE Transactions https://doi.org/10.1109/TMC.2021.3074917
3. Taha, M. (2023). An efficient software-defined network controller based routing adaptation for enhancing QoE of multimedia streaming service. Multimedia Tools and Applications. https://doi.org/10.1007/s11042-023-14938-5
4. MDN Web Docs. Effective connection type (ECT). https://developer.mozilla.org/en-US/docs/Glossary/Effective_connection_type
5. W3C Wiki. Network Quality Monitoring and Prediction. https://www.w3.org/wiki/Network_Quality_Monitoring_and_Prediction
6. Kruger, H. J. J. (2023). A conceptual quality framework for online distance education in a South African higher education context. http://hdl.handle.net/2263/93779
7. Jueckstock, J. P. (2021). Enhancing the security and privacy of the web browser platform via improved web measurement methodology. https://repository.lib.ncsu.edu/server/api/core/bitstreams/65b7fbc8-0e70-41f8-82dd-bfc79d6e37c9
8. MDN Web Docs. NetworkInformation (API). https://developer.mozilla.org/en-US/docs/Web/API/NetworkInformation
9. Kirimi, N. & Barakat, C. (2024). Passive network monitoring and troubleshooting from within the browser: A data-driven approach. International Wireless Communications and Mobile Computing Conference (IWCMC 2024). https://doi.org/10.1109/IWCMC61514.2024.10592376
10. MDN Web Docs. Performance (Web Performance API). https://developer.mozilla.org/en-US/docs/Web/API/Performance
11. MDN Web Docs. PerformanceResourceTiming. https://developer.mozilla.org/en-US/docs/Web/API/PerformanceResourceTiming

12. MDN Web Docs. PerformanceObserver. https://developer.mozilla.org/en-US/docs/Web/API/PerformanceObserver
13. Goenka, P., Zarifis, K., Gupta, A. & Calder, M. (2022). Towards client-side active measurements without application control. ACM SIGCOMM Computer Communication Review. https://doi.org/10.1145/3523230.3523234
14. Verma, D. (2022). A comparison of web framework efficiency: Performance and network analysis of modern web frameworks. https://urn.fi/URN:NBN:fi:amk-2022061417896
15. Lucas, G. A., Lunardi, G. L. & Dolci, D. B. (2023). From e-commerce to m-commerce: An analysis of the user's experience with different access platforms. Electronic Commerce Research and Applications, 58, 101240. https://doi.org/10.1016/j.elerap.2023.101240
16. Sharma, T., Mangla, T., Gupta, A., Jiang, J. & Feamster, N. (2023). Estimating WebRTC video QoE metrics without using application headers. IMC 2023. https://doi.org/10.1145/3618257.3624828
17. Mangla, T., Halepovic, E., Ammar, M. & Zegura, E. (2019). Using session modeling to estimate HTTP-based video QoE metrics from encrypted network traffic (eMIMIC). IEEE Transactions on Network and Service Management. https://doi.org/10.1109/TNSM.2019.2924942
18. Bronzino, F., Schmitt, P., Ayoubi, S., Martins, G., Teixeira, R. & Feamster, N. (2019). Inferring streaming video quality from encrypted traffic: Practical models and deployment experience. Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS). https://doi.org/10.1145/3366704
19. Rudin, C. (2019). Stop explaining black box machine learning models for high-stakes decisions and use interpretable models instead. Nature Machine Intelligence. https://doi.org/10.1038/s42256-019-0048-x
20. TensorFlow.js. Platform and environment. https://www.tensorflow.org/js/guide/platform_environment
21. Microsoft Open Source Blog. ONNX Runtime Web – Running your machine learning model in browser. https://opensource.microsoft.com/blog/2021/09/02/onnx-runtime-web-running-your-machine-learning-model-in-browser/
22. MDN Web Docs. WebAssembly. https://developer.mozilla.org/en-US/docs/WebAssembly
23. W3C. Web Neural Network API (WebNN). https://www.w3.org/TR/webnn/
24. Dhar, S., Tang, X., et al. (2021). A survey of on-device machine learning: An algorithms and learning theory perspective. ACM Computing Surveys. https://doi.org/10.1145/3450494
25. Shi, W. & Dustdar, S. (2016). The promise of edge computing. IEEE Computer. https://doi.org/10.1109/MC.2016.145
26. Malavolta, I., et al. (2020). Evaluating the impact of caching on the energy consumption and performance of progressive web apps. MOBILESoft '20. https://doi.org/10.1145/3387905.3388593
27. Wang, Q., et al. (2025). Anatomizing deep learning inference in web browsers. ACM (IMC record). https://doi.org/10.1145/3688843
28. Yan, Y., Tu, T., Zhao, L., Zhou, Y. & Wang, W. (2021). Understanding the performance of WebAssembly applications. IMC '21. https://doi.org/10.1145/3487552.3487827

Received (Надійшла) 14.08.2025
Accepted for publication (Прийнята до друку) 22.10.2025

ABOUT THE AUTHORS / ВІДОМОСТІ ПРО АВТОРІВ

Приліпа Артем Олегович – аспірант кафедри комп'ютерної інженерії та програмування, Національний технічний університет "Харківський політехнічний інститут", Харків, Україна;
Artem Prylipa – PhD student, Department of Computer Engineering and Programming, National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine;
e-mail: Artem.Prylipa@cs.khpi.edu.ua; ORCID Author ID: https://orcid.org/0009-0005-6633-8308.

Ганна Євгенівна Філатова – доктор технічних наук, професор, професор кафедри комп'ютерної інженерії та програмування, Національний технічний університет "Харківський політехнічний інститут", Харків, Україна;
Anna Filatova – Doctor of Technical Sciences, Professor of Computer Engineering and Programming, National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine;
e-mail: Hanna.Filatova@khpi.edu.ua; ORCID Author ID: https://orcid.org/0000-0003-1982-2322;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=56448583600.

TinyML network profiler in browser

Artem Prylipa, Anna Filatova

Abstract. This research presents a client-side TinyML profiler of network in the web browser, designed for operation on resource-constrained devices and under unstable or low-bandwidth network conditions. The solution combines a threshold-based decision rule with a lightweight logistic (softmax) model executed locally to classify the quality of the network connection. **Relevance.** The rising share of mobile traffic, the heterogeneity of network environments, and the limited fidelity of existing browser indicators complicate accurate client-side decision-making. **Object of study:** client-side methods for profiling and classifying the quality of a browser-based network connection using standard Web Performance APIs and ML models. **Aim:** to analyze, design, and implement a profiler capable of classifying connection types and producing interpretable features without server support. **Methods.** Navigation/Resource Timing and PerformanceObserver are used to collect raw signals. A 14-dimensional feature vector is formed (medians/quantiles of RTT and throughput, variability measures, a loss-likelihood heuristic, and protocol/Service Worker indicators). The proposed a threshold rule with hysteresis and decision confidence, together with a softmax model. **Results.** The developed profiler requires no special permissions or third-party services. In controlled network-emulation scenarios it improves classification accuracy over a pure threshold baseline, while maintaining low overhead and decision explainability. **Conclusions.** The proposed client-side approach provides an effective, rapid, and interpretable assessment of network conditions in the browser and is suitable for resource-constrained settings. The results can be leveraged to further enhance adaptive content loading, expand the feature space, and deploy more compact ML models in web applications.

Keywords: client-side network profiling, TinyML, Web Performance API, softmax classification, RUM measurements, adaptive content loading, QoE.