PixLift: Enhancing Mobile Web Browsing through Real-time Image Super-Resolution

Yonas Atinafu
yma9130@nyu.edu
New York University Abu Dhabi
Abu Dhabi, UAE

Sarthak Malla
sarthak.malla@nyu.edu
New York University Abu Dhabi
Abu Dhabi, UAE

Advised by: Prof. Yasir Zaki

ABSTRACT

The abundance of images on the internet demands efficient image-transfer strategies to optimize page load times and overall site performance. Modern pages carry ever-larger image payloads [6], which harm performance on low-end devices and in regions with limited bandwidth. To address this, we propose PixLift, a system that leverages downscaled images during network transfer and upscales them on the client using an on-device AI upscaler.

PixLift integrates multiple lightweight super-resolution models selected for their performance and compatibility across a spectrum of mobile devices to realize a pipeline for efficient image transfer. By trading local compute for bandwidth, PixLift reduces data transfer without compromising image quality.

In this paper, we tackle three challenges: evaluating the feasibility of scaled image requests from popular sites, building PixLift as a browser extension, and measuring its effect on user experience. We evaluate PixLift on 71.4K real-world webpages and three rooted Android devices (low-end to mid-high-end). Performance is measured via load-time reduction, data-transfer savings, and visual quality metrics (PSNR, SSIM). We also monitor CPU and memory overhead to assess feasibility on constrained hardware.

Our results demonstrate significant page-load speedups and data savings, enabling better web access for users on limited bandwidth networks. This work bridges web optimization, machine learning, and mobile computing to democratize high-quality browsing worldwide.

KEYWORDS

image optimization, network performance, image upscaling, super-resolution, browser optimization

Reference Format:
Yonas Atinafu and Sarthak Malla. 2025. PixLift: Enhancing Mobile Web Browsing through Real-time Image Super-Resolution. In *NYUAD Capstone Project 2 Reports, Spring 2025, Abu Dhabi, UAE.* 9 pages.

1 WORK DIVISION

1.1 Yonas

In Capstone Project 1, I developed PixLift as a Chromium extension that intercepts each image request, fetches a downscaled copy, upscales it with embedded TensorFlow.js super-resolution models. I ran controlled experiments on three Android devices loading synthetic pages with varied image counts to measure page load speed, CPU/memory overhead, and PSNR/SSIM.

In Capstone Project 2, I worked on doing the upscaling in Chromium's GPU process: in the compositor's render loop, each texture quad is pulled back via Skia into CPU memory, normalized into an ONNX Runtime tensor, upscaled, and then reuploaded as a new GPU texture. This GPU→CPU→ONNX→GPU flow embeds real time super-resolution directly into Chromium's multi process architecture.

1.2 Sarthak

In Capstone Project 1, I performed a large-scale empirical study of web image delivery to lay the foundation for *PixLift*, an AI-driven upscaling pipeline. I began by collecting HTTP Archive data from the top 50,000 sites (71,433 pages and over 3.1 million unique image URLs served by 54,754 hostnames), and analyzed format prevalence and transfer sizes to identify performance bottlenecks. Next, I developed and applied a comprehensive set of URL-pattern heuristics (e.g., `width=`, *size*=, `300x300`, `resize/(large|small|medium)`) to detect and quantify image-resizing support. To ensure robust benchmarking, I then created a stratified sampling framework based on each site's resizing-support ratio and total image count, selecting 100 representative pages for end-to-end timing and resource-usage evaluation of *PixLift*. This methodical approach not only uncovers current limitations in web image handling but also provides a rigorously curated testbed for assessing the real-world benefits of our upscaling solution.

In Capstone Project 2, I worked on integrating the *PixLift* pipeline within the Chromium runtime. This involved identifying the correct model format, navigating the Chromium repository to identify the Browser process to load the model, creating a Mojo IPC to expose a connection between the Renderer process and Browser process to facilitate image upscaling and rendering on the fly.

1.3 Common

We worked together on a Renderer process prototype. Blink loads the ONNX model. It upscales each decoded image before compositing.

2 INTRODUCTION

Images are fundamental to the web, with 99.9% of active sites using at least one image to serve their content [14]. On mobile devices, images represent around 38.9% of the page weight (a median of 900KB out of 2,311KB) which drive up load time sna data consumption [2]. High-resolution images deliver visual appeal but create a steep cost for users on metered or slow connections. In regions where data is expensive or scarce, these heavy images can block access entirely, reinforcing digital divides and limiting economic opportunity. [17].

Efficient image transfer strategies are, hence, imperative to optimize webpage loading and overall site performance. The solution we propose involves fetching scaled-down images over the network and restoring their fidelity on the client side. Modern AI super-resolution models (e.g., QuickSR-Net, SESR-M5, Real ESRGAN) can reconstruct high-resolution detail in real time [8, 9]. By running these models in-browser via TensorFlow.js, we trade inexpensive local compute cycles for costly network bytes, dramatically cutting data transfer without sacrificing visual quality.

Before building this pipeline, however, we first quantify today's web image practices. We analyze top 71.4 K sites from the HTTP Archive [1], measuring image transfer sizes, formats, and server support for remote image downscaling. We observe that only 10% of the webpages request images from servers that support image resizing, and a mere 1.5% of the webpages have full downscaled image support. Despite the small support in today's internet practices, *PixLift* remains important to exploit current methods.

PixLift is implemented as a Chromium extension. It intercepts image requests, adds scaling parameters, and then applies one of three embedded super-resolution models to recover the image. The three models we use are QuickSRNet Small 4x, SR-Sub-Pixel CNN, and SESR-M5. We observe that "QuickSRNet Small4x" is the fastest, delivering around 10x speedup over SR - Sub-Pixel CNN, but degrades quality in about 40% of challenging images.

Finally, we evaluate `PixLift` under realistic low-bandwidth mobile conditions (1.6Mbps down/768Kbps up, 150ms RTT) on the top 1000 Pakistani sites [11]. PixLift cuts page-load time by a median of 7 seconds. It saves multiple megabytes per page. CPU rises by 10–20%; memory by 1GB. On a Galaxy A03s, upscaling ten above-the-fold images completes before the ‘onload‘ event. A user study with 100 participants confirms that visual quality remains indistinguishable from originals.

This paper presents our data-driven analysis, system design, and user-centered evaluation, laying the groundwork for practical, large-scale image optimization in bandwidth constrained environments.

3 RELATED WORK

Enhancing web browsing performance and user experience has been a focus of several research efforts, with varying emphasis on techniques and trade-offs. However, fewer studies have specifically investigated the role of images in this context. We summarize three significant works that are closely related to our study.

3.1 WebLego

WebLego [19] introduces a novel approach to web performance optimization by racing semantically *similar* images in place of the original content. The key idea behind WebLego is to prioritize faster page load times by relaxing strict content fidelity. By using alternative images that are semantically similar to the original, WebLego ensures that pages load more quickly, which can enhance user experience in high-latency environments. However, the approach comes with a significant drawback: it often increases the overall page size. This makes WebLego unsuitable for use in developing regions where limited bandwidth is a major constraint. Despite this limitation, WebLego demonstrates the potential of image-centric optimizations to improve web performance by trading off content fidelity.

3.2 ScaleUp

ScaleUp [16] is a browser extension specifically designed to address the challenges of low-bandwidth and high-latency networks. It achieves this by dynamically adjusting the browser's scaling factor, effectively reducing the number of objects that need to be loaded above the fold. The extension increases the size of visible content, such as text and images, allowing these elements to load faster while deprioritizing content below the fold. This technique ensures a smoother browsing experience in constrained network conditions. However, ScaleUp's effectiveness diminishes on websites that load substantial content below the fold. This limitation restricts its utility in contexts where below-the-fold content is critical to the user experience. Nevertheless, ScaleUp presents a complementary approach to our proposed system and highlights the importance of above-the-fold prioritization in optimizing browsing performance.

3.3 BrowseLite

BrowseLite [15] is a client-side optimization tool that focuses on reducing data usage by optimizing web images. It achieves this by opportunistically requesting alternative image encodings or partial image data, depending on the network conditions and device capabilities. Unlike server-side solutions, BrowseLite operates entirely on the client side, maintaining user privacy and ensuring compatibility with a wide range of websites. BrowseLite excels in bandwidth savings while preserving a satisfactory level of image quality. However, the tool does not utilize advanced techniques like local AI models to enhance image quality further. Our proposed approach builds on the ideas introduced by BrowseLite by using highly compressed images and subsequently using local machine learning models for upscaling. This integration allows for even greater data savings and improved visual fidelity, addressing some of the limitations of BrowseLite.

3.4 Comparison and Contribution

While WebLego [19], ScaleUp [16], and BrowseLite [15] present innovative solutions to optimize web performance, none of these approaches leverage machine learning models for image enhancement. WebLego compromises content fidelity for speed, ScaleUp focuses on browser scaling without addressing image quality, and BrowseLite prioritizes data savings without incorporating ML-driven upscaling techniques. Our proposed system bridges this gap by introducing a novel method that combines the use of highly compressed images with local ML-based super-resolution models, enhancing both performance and user experience. This hybrid approach addresses the trade-offs in existing methods and provides a scalable solution for optimizing web image delivery, particularly in low-bandwidth environments.

4 METHODOLOGY

We pursued two main strands: a large-scale analysis of real world image delivery and an in-browser evaluation of super resolution upscaling.

4.1 Web Data Analysis

To enable a more comprehensive examination of web image delivery, we drew on the publicly available **HTTP Archive** dataset in Google BigQuery. This resource continuously crawls the most popular sites, organized into tiers (top1K, top5K, top10K) by the Chrome User Experience Report's logarithmic ranking, and stores over 2.1 trillion records (as of December 8, 2024), each capturing a single page's HTML, JavaScript, images, audio, and video assets [1, 3, 4]. Leveraging BigQuery's powerful SQL interface, we extracted the top 50K pages across these tiers and recorded every image's count, byte size, and format. Because we included both landing and internal URLs, our final corpus grew to roughly 71.4K distinct pages.

From this dataset, we pursued three goals. First, we characterized image-format usage—tracking the rise and fall of JPEG, WebP, AVIF, and other types via cumulative distribution analyses. Second, we systematically tested each image hosting domain for built-in resizing support by issuing modified URL parameters (e.g., `width=`, `size=`, “large” 'small/medium') and flagging hosts that returned smaller payloads. Third, we assembled a balanced subset of sites, spanning formats, sizes, and resizing behaviors, to serve as our testbed for evaluating *PixLift*'s bandwidth-vs-compute trade-offs. All of these were thoroughly analyzed to design the pipeline discussed in Section 4.5.

4.2 Representative Set for *PixLift* Pipeline

We create a representative set of websites that mirrors the current state of the web regarding image resizing capabilities. This dataset serves as the benchmark for evaluating the *PixLift* pipeline against two criteria: (1) user Quality of Experience (QoE), assessed using web performance metrics, and (2) resource usage, measured through data consumption, CPU utilization, and battery impact.

4.2.1 Methodology: To construct this representative set, we applied a stratified sampling approach to the corpus of 71.4k webpages. Sampling was based on two critical metrics:

(1) **Percentage image resizing support**: the fraction of hosted images within a site that supports resizing. For example: a site that has 10% of the images uses hostnames that support image resizing.
(2) **Total count of hosted images**: the overall number of images present on the site.

We divided the dataset into distinct strata based on intervals of these metrics. For image resizing support, we used intervals such as [0–10%], [10–20%], ..., [90-100%], while for the total image count, we grouped sites into their quartiles (i.e. [0-Q1], [Q1-Median], [Median-Q3], [Q3-Maximum count]). Each page, thus, falls into a unique combination of these strata.

Within each stratum, we determined the proportional representation of pages relative to the total dataset. Using this proportional distribution, we performed random sampling to select 100 webpages in total. This method ensured the representative set accurately reflected the distribution of webpages with varying levels of image resizing support and image density.

4.2.2 Evaluation of Representativeness. The stratified sampling approach allowed us to create a diverse and balanced dataset, encompassing a wide spectrum of web scenarios. This representative set serves as the foundation for benchmarking the *PixLift* pipeline under real-world conditions. By accounting for both the prevalence of image resizing and image density, we ensure that the results derived from this evaluation can be generalized to the broader web ecosystem.

4.3 Super-Resolution Models

We evaluated seven state-of-the-art image super-resolution models for integration: QuickSRNet (Small, Medium, and Large variants), SESR-M5, SR_Sub-Pixel CNN, Real ESRGAN, and XLSR [7]. Based on compatibility and performance considerations, the following models were selected for testing:

- **QuickSRNet Small 4X**: Lightweight and suitable for real-time processing on low-end devices.
- **SESR-M5**: Known for its efficiency and competitive performance across a wide range of devices.
- **SR_Sub-Pixel CNN**: Offers fast processing with satisfactory image quality enhancement.
- **Real-ESRGAN**: Selected exclusively for mid-to-high-end devices (e.g., Galaxy A34) due to its higher computational demands and superior visual output.

To ensure compatibility and efficient deployment of super-resolution models on web platforms, a multi-stage conversion pipeline was employed to transform the models from their original PyTorch (`.pth`) [5] format into TensorFlow.js (`.tfjs`) [13] format:

(1) **Conversion to ONNX [12]**: Each PyTorch model was exported to the ONNX (`.onnx`) format, enabling interoperability between frameworks.
(2) **Conversion to TensorFlow**: ONNX models were converted into TensorFlow (`.tf`) format, integrating them within the TensorFlow ecosystem.
(3) **Conversion to TensorFlow.js (TFJS)**: Finally, TensorFlow models were converted to TFJS (`.tfjs`) format. This step included quantizing [10] models to Float16 for improved performance and ensuring suitable input shapes for various devices.

After initial assessments, the models selected for further evaluaiton were QuickSRNet Small 4X, SESR-M5, SR_Sub-Pixel CNN, and Real ESRGAN. The QuickSRNet Large, Medium, and XLSR models were excluded due to higher computational demands and compatibility issues with low-end devices.

4.4 Integration with TensorFlow.js

The selected models were integrated using TensorFlow.js, enabling real-time image enhancement directly within the browser environment. TensorFlow.js provides a flexible and efficient platform for deploying super-resolution models by leveraging WebGL for GPU acceleration. This ensures that the extension runs smoothly on devices with varying computational capabilities, including low-end smartphones.

Key configurations include: Setting the TensorFlow.js backend to WebGL (`await tf.setBackend('webgl')`) to utilize GPU acceleration; Optimizing precision settings by disabling WEBGL_FORCE_F16_TEXTURES, allowing the use of lower-precision textures when appropriate; Ensuring system readiness by awaiting TensorFlow.js initialization (`await tf.ready()`).

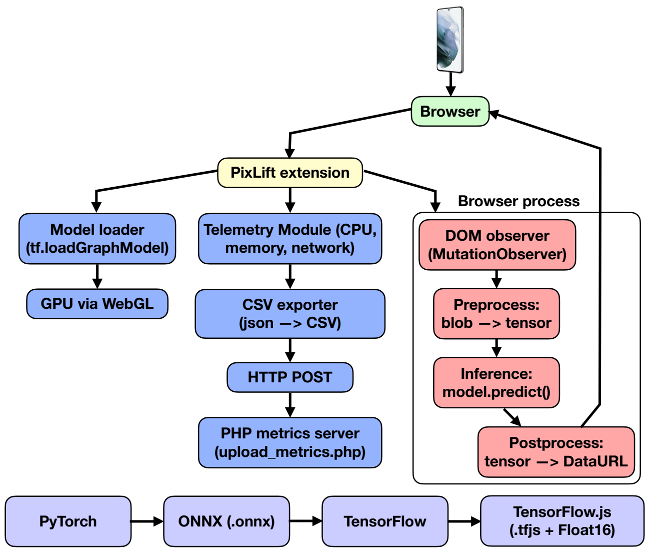

Browser
PixLift extension
Browser process
Model loader (tf.loadGraphModel)
GPU via WebGL
Telemetry Module (CPU, memory, network)
CSV exporter (json —> CSV)
HTTP POST
PHP metrics server (upload_metrics.php)
DOM observer (MutationObserver)
Preprocess: blob —> tensor
Inference: model.predict()
Postprocess: tensor —> DataURL
PyTorch
ONNX (.onnx)
TensorFlow
TensorFlow.js (.tfjs + Float16)

Figure 1: *PixLift* system flow diagram.

4.5 Development of a Browser Extension for Kiwi Browser

The practical application of this project is realized through a browser extension designed to dynamically process images on web pages. The extension incorporates the following features:

- **Image Detection and Monitoring**:
  A `MutationObserver` monitors the web page's `<img>` elements, dynamically queuing newly loaded or modified images for processing.
- **Controlled Concurrency**: The extension processes a maximum of two images concurrently, using a FIFO queue to manage tasks efficiently without overloading device resources.
- **Image Processing Pipeline**:
  - **Preprocessing**: Images are fetched as blobs [18], converted to tensors, resized to the model's input dimensions, and normalized.
  - **Inference**: The preprocessed tensor is fed into the super-resolution model, generating an enhanced tensor.
  - **Postprocessing**: The enhanced tensor is denormalized, rendered onto a canvas, and converted back to a Data URL for display.

5 EVALUATION

This section assesses the performance, visual quality, and resource consumption of our super-resolution extension under controlled conditions and through a user study. To evaluate the extension's scalability and performance, tests were conducted on three rooted Samsung Galaxy devices:

- **Galaxy A12 & A03s (Low-End Devices)**: Models tested include QuickSRNet Small 4X, SESR-M5, and SR_Sub-Pixel CNN. Real ESRGAN was excluded due to its higher resource demands.
- **Galaxy A34 (Mid-to-High-End Device)**: All four models were tested, including Real ESRGAN, leveraging the device's superior computational capabilities.

We report latency, processing time, CPU and memory usage, and two image-quality metrics: Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM).

5.1 Device Specifications

Galaxy A03s: Octa-core Cortex-A53 (4 × 2.35 GHz + 4 × 1.8 GHz), PowerVR GE8320 GPU, 4 GB RAM.
Galaxy A12: Same CPU/GPU class as A03s, 4 GB RAM.
Galaxy A34: Octa-core (2 × 2.6 GHz Cortex-A78 + 6 × 2.0 GHz Cortex-A55), Mali-G68 MC4 GPU, 4 GB RAM.

5.2 Controlled Device Experiments

Experimental Design. We ran experiments for each combination of the model and device. For every model–device pairing we ran 500 page-load trials. Each trial loaded a synthetic page containing one to ten above-the-fold images chosen uniformly at random from a pool of 100 images with mixed resolutions and formats collected through the method described in Section 4.2. The telemetry module embedded in the extension captured all metrics automatically.

Procedure.

(1) Load test page; measure image fetch latency.
(2) Run the super-resolution pipeline (preprocess → inference → postprocess).
(3) Log PSNR, SSIM, CPU, and memory per image and per page load.

Dataset Diversity. Images span JPEG, PNG, WebP, and AVIF formats and resolutions from 360 p to 4 K, ensuring that the pipeline is stressed across a broad range of inputs.

Repetition and Variance Control. Each configuration (device × model × image count) was repeated 50 times to account for noise in CPU scheduling and garbage collection. Mean and 95 % confidence intervals were computed for all metrics.

5.3 User Study

To gauge perceptual improvements, we conducted a double-blind preference test with 100 participants. Twenty images were randomly sampled from the test pool, downscaled to 200 px width, and then upscaled using SESR-M5, SR Sub-Pixel CNN, and QuickSRNet. Each participant viewed the three upscaled variants plus the original and ranked them by perceived quality. Votes were aggregated using Borda counting. The study complements objective PSNR/SSIM scores with human perception.

6 RESULTS

This section reports objective metrics (PSNR, SSIM, latency, processing time, CPU and memory usage) and subjective user feedback for the super-resolution extension on three devices: Galaxy A03s, Galaxy A12, and Galaxy A34. Unless stated otherwise, results reflect 500 controlled page-load trials per model–device pairing and 100 human-rated images.

6.1 Image Quality: PSNR and SSIM

Figure 2 summarizes PSNR and SSIM across models.

- **PSNR.** Values span 15–40 dB. SR_Sub-Pixel CNN yields the highest medians on all devices (26–30 dB). QuickSRNet and SESR-M5 are slightly lower (22–28 dB) but remain above the 20 dB usability threshold.
- **SSIM.** Most scores exceed 0.90. SESR-M5 leads on average (0.95–0.97), while SR_Sub-Pixel CNN follows closely; QuickSRNet is a consistent third but never drops below 0.92.

Insight. SR_Sub-Pixel CNN offers the best objective fidelity, whereas SESR-M5 delivers the strongest perceptual similarity.

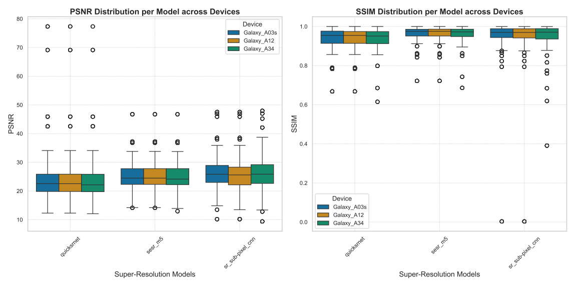


Figure 2: PSNR and SSIM Comparison Across Models. This visualization highlights the performance of different super-resolution models in terms of image quality metrics. Higher PSNR and SSIM values indicate better image fidelity and perceptual quality.

6.2 Processing Time and Latency

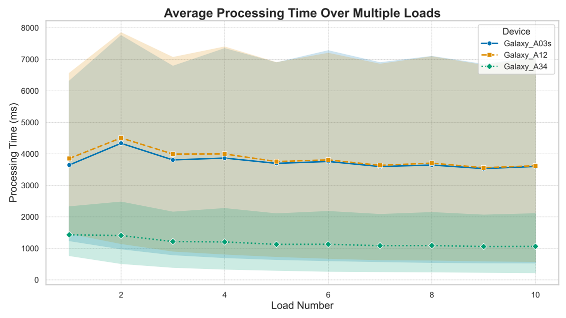


Figure 3: Processing Time Over Multiple Page Loads. This graph illustrates how processing times vary across consecutive page loads for different devices and models.

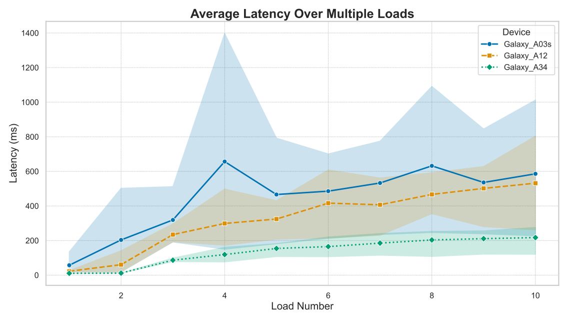


Figure 4: Latency Over Multiple Page Loads. This graph highlights the variation in latency during consecutive page loads, emphasizing device-specific and model-specific performance.

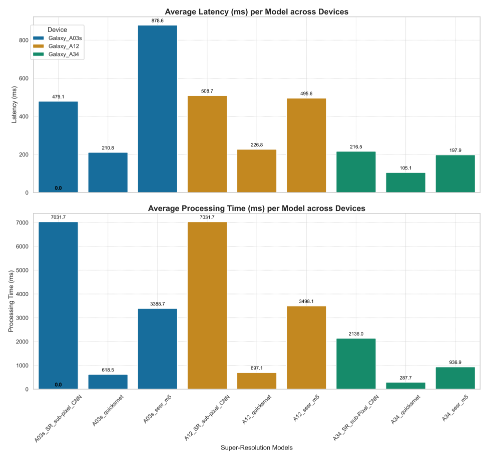


Figure 5: Comparison of Latency and Processing Time Across Devices. This comparison showcases the relationship between latency and processing time for the evaluated super-resolution models.

Figures 3, 4, and 5 detail timing results.

- **Low-end devices (A03s, A12).**
  - Average processing times vary based on the number of images processed, with single and two-image cases showing linear increases in processing time due to the concurrency limit (`MAX_CONCURRENT = 2`).
  - For 1–2 images: Processing times reflect immediate GPU usage without overlap, with times ranging from 3800–4500 ms.
  - For 3+ images: Queuing by the CPU enables overlap between image transfer and GPU processing, resulting in sublinear growth in processing time. Stabilization occurs as tasks are efficiently queued, reducing delays.
  - As for network latency in these devices, it peaks at 1.4 s for A03s and stabilizes near 0.6 s for A12.
- **Mid-range device (A34).**
  - Processing stays within 1–2 s regardless of image count thanks to a faster Mali-G68 GPU.
  - Fetch latency remains 0.15–0.25 s even at maximum load.

A34 completes both transfer and upscaling 50–75 % faster than low-end phones.

6.3 CPU and Memory Usage

As shown in table 1 SESR-M5 squeezes the most work out of each CPU cycle, so it keeps processor demand in the mid-teens while holding memory between 2.1 GB and 4.6 GB.

SR_SubPixel CNN inserts sub-pixel layers that explode intermediate feature maps; this raises its RAM footprint to as much as 5.8 GB even though clever tensor re-arrangements keep CPU load in the 14–17% band. QuickSRNet trims parameters and intermediate buffers to stay near 2 GB on low-end phones, but its lightweight design shifts more scheduling overhead to the CPU, pushing usage toward 20

6.4 User Study

Participants compared originals to outputs from SESR-M5, SR_Sub-Pixel CNN, and QuickSRNet on a 1–10 scale (5 = identical to original). All three models scored significantly above 5 (p<0.01, paired t-test). No statistically significant difference emerged among the three models, though SR_Sub-Pixel CNN showed a marginally higher mean score.

6.5 Summary of Findings

We found that SR-Sub-Pixel CNN delivers the highest fidelity (PSNR), with SESR-M5 close behind on perceptual quality (SSIM), while QuickSRNet Small 4× is the fastest, achieving roughly a 10× speed-up over SR-Sub-Pixel CNN. In resource-limited settings, QuickSRNet is the most efficient on devices with tight RAM budgets, whereas SESR-M5 is preferable when CPU cycles are scarce. Importantly, all of our models finish upscaling before the browser's `onload` event on a midrange A34 device, even with ten above-the-fold images, and low-end hardware easily meets this target for up to five images.

7 CHROMIUM INTEGRATION

Currently, PixLift is developed as a browser extension. Despite its advantage of on-demand deployment, not all mobile browsers allow for extensions. This motivates the need for the pipeline to be directly integrated into the Chromium runtime. In this section, we describe how we attempted to embed PixLift within Chromium's multi-process architecture to provide native super-resolution capabilities.

7.1 Renderer Process

Our initial implementation attempted to execute the ONNX-formatted super-resolution model inside the Renderer process. This process is executed through a third party called Blink, which is responsible for composing the pixels that make up the web pages.

We attempted to load the ONNX model within Blink. In this design, images decoded by Skia were converted into normalized tensors for inference by ONNX Runtime and then transformed back into Skia bitmaps for display. However, during testing, we found that Blink does not include a mechanism to load large model files at runtime. To address this limitation, we moved model loading into the Browser process, which has higher privileges. We implemented a Mojo IPC interface so that the Browser process loads and caches the ONNX model once at startup, then the Renderer process issues inference requests and receives upscaled pixel data in return. This separation of responsibilities eliminates redundant file I/O in each renderer instance and maintains a clear division between resource management and pixel rendering.

Figure 7 shows the dataflow of how images are handled for upscaling in the Chromium runtime. We inject our pipeline within a function called `DecodeFrameBufferAtIndex`, which decodes the image before queuing it up for rendering. We simply take the bitmap of the image, as explain above, and replace it with the bitmap of the upscaled image.

However, this pipeline is not effective yet, with the chromium build unable to load the model correctly and run the image optimization. Below, we discuss shifting the image upscaling into a GPU process, which was also rendered ineffective in producing a viable and stable solution.

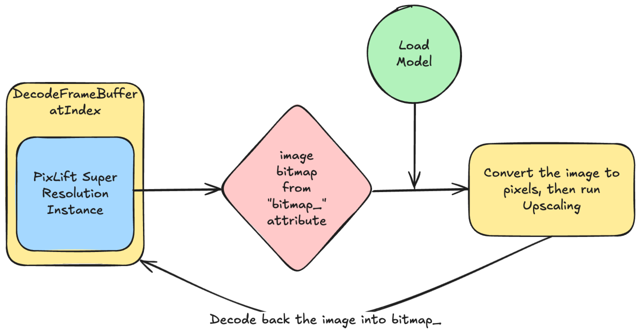
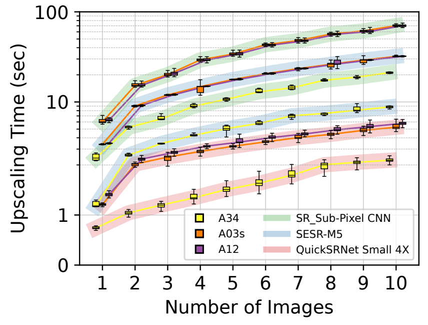

Load Model
DecodeFrameBuffer atIndex
PixLift Super Resolution Instance
image bitmap from "bitmap_" attribute
Convert the image to pixels, then run Upscaling
Decode back the image into bitmap_

Figure 7: *PixLift* chromium runtime integration.

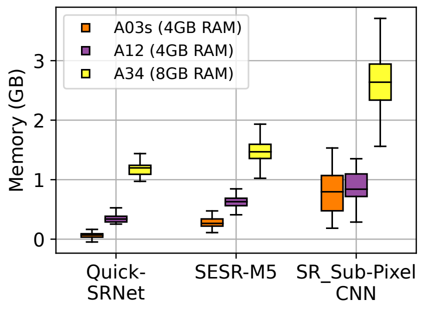
7.2 GPU Process

We implemented the upscaling process in the GPU process as an intermediary step for resolving the issue we encountered in the Renderer process. We postulated that Blink was not able to run the model at all, and needed more computational resources through GPU GPU-accelerated process.

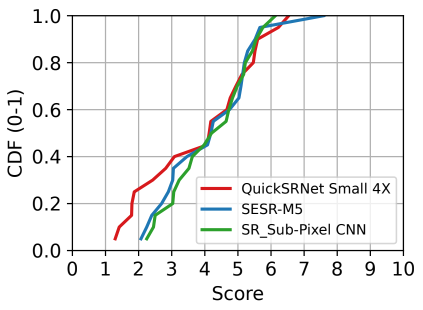
For every frame, the compositor grabs each texture quad's GPU image and uses Skia to read its RGBA pixels back into CPU memory. These pixels are normalized and packed into an ONNX-compatible tensor, which ONNX Runtime upscales to the target resolution. The resulting tensor is denormalized, converted back into pixel data, and encapsulated in a new Skia image. Finally, the extension uploads this image as a fresh GPU texture and replaces the original quad, thus forming a continuous GPU→CPU→ONNX→GPU super-resolution loop.



(a) Upscaling time per model and mobile device.



(b) CPU memory (RAM) usage.



(c) CDF of image quality scores for 20 images and 100 participants. Scores: 0 (much worse), 5 (same), 10 (much better).

Figure 6: *PixLift* performance evaluation: resource usage and visual quality assessment via user study.

Table 1: Resource usage per super-resolution model

Metric	SESR-M5	SR_Sub-Pixel CNN	QuickSRNet
CPU usage (%)	13–18	14–17	18–21
Memory usage (GB)	2.1–4.6	2.6–5.8	2.0 (low-end)

8 CONCLUSION

This work presents *PixLift*, a browser-based pipeline that trims bandwidth by downscaling images in transit and restores visual quality with on-device super-resolution. Extensive tests on the Galaxy A03s, A12, and A34 confirm that the idea is practical across a wide hardware spectrum. SR_Sub-Pixel CNN consistently delivers the highest PSNR, while SESR-M5 achieves the strongest SSIM, preserving fine structure and perceptual detail. QuickSRNet Small 4x stands out for speed and minimal memory use, making it attractive for entry-level phones even though it demands slightly more CPU. On the mid-range A34, all models complete inference before the browser's `onload` event with ten above-the-fold images; low-end devices meet the same target with up to five images, proving the approach scales with available compute.

The study also shows that users rate upscaled images as visually equal to—or better than—their originals, validating the objective gains in PSNR and SSIM. These results demonstrate that modest client-side computation can offset expensive network transfers, yielding faster, data-efficient browsing without server changes or privacy risks.

Future work will refine the extension along several fronts. First, systematic hyper-parameter searches and learning-rate schedules may shave additional milliseconds from inference while preserving quality. Second, moving critical kernels to WebAssembly could accelerate devices lacking robust GPU drivers. Third, adaptive queue management can balance latency and resource use on the fly, especially under constrained memory. Fourth, distilled or sparsity-pruned variants of the current models can target ultra-low-end phones where even 2 GB of RAM is a luxury. Finally, longer-term user studies will capture day-to-day perceptions of speed, data savings, and visual quality, ensuring that technical improvements translate into tangible benefits.

By uniting data-driven insight with careful engineering, offers a scalable path toward richer yet lighter mobile web experiences, helping close the digital divide for users who pay the highest price for every kilobyte.

REFERENCES

[1] [n.d.]. https://har.fyi/guides/getting-started/
[2] 2024. https://almanac.httparchive.org/en/2024/page-weight
[3] 2024. https://developer.chrome.com/docs/crux/methodology/metrics
[4] 2024. https://httparchive.org/
[5] 2024. PyTorch: An open-source machine learning framework. https://pytorch.org.
[6] HTTP Archives. 2022. Page Weight. In *Web Almanac: HTTP Archive's annual state of the web report.* HTTP Archives, Article Part IV Chapter 21. https://almanac.httparchive.org/en/2022/page-weight
[7] Mustafa Ayazoglu. 2021. Extremely Lightweight Quantization Robust Real-Time Single-Image Super Resolution for Mobile Devices, In Proceedings of the IEEE/CVF Computer Vision and Pattern Recognition Workshops (Mobile AI 2021 Workshop). *arXiv preprint arXiv:2105.10288.* https://doi.org/10.48550/arXiv.2105.10288
[8] G. Berger, M. Dhingra, A. Mercier, Y. Savani, S. Panchal, and F. Porikli. 2023. QuickSRNet: Plain Single-Image Super-Resolution Architecture. In *CVF.*

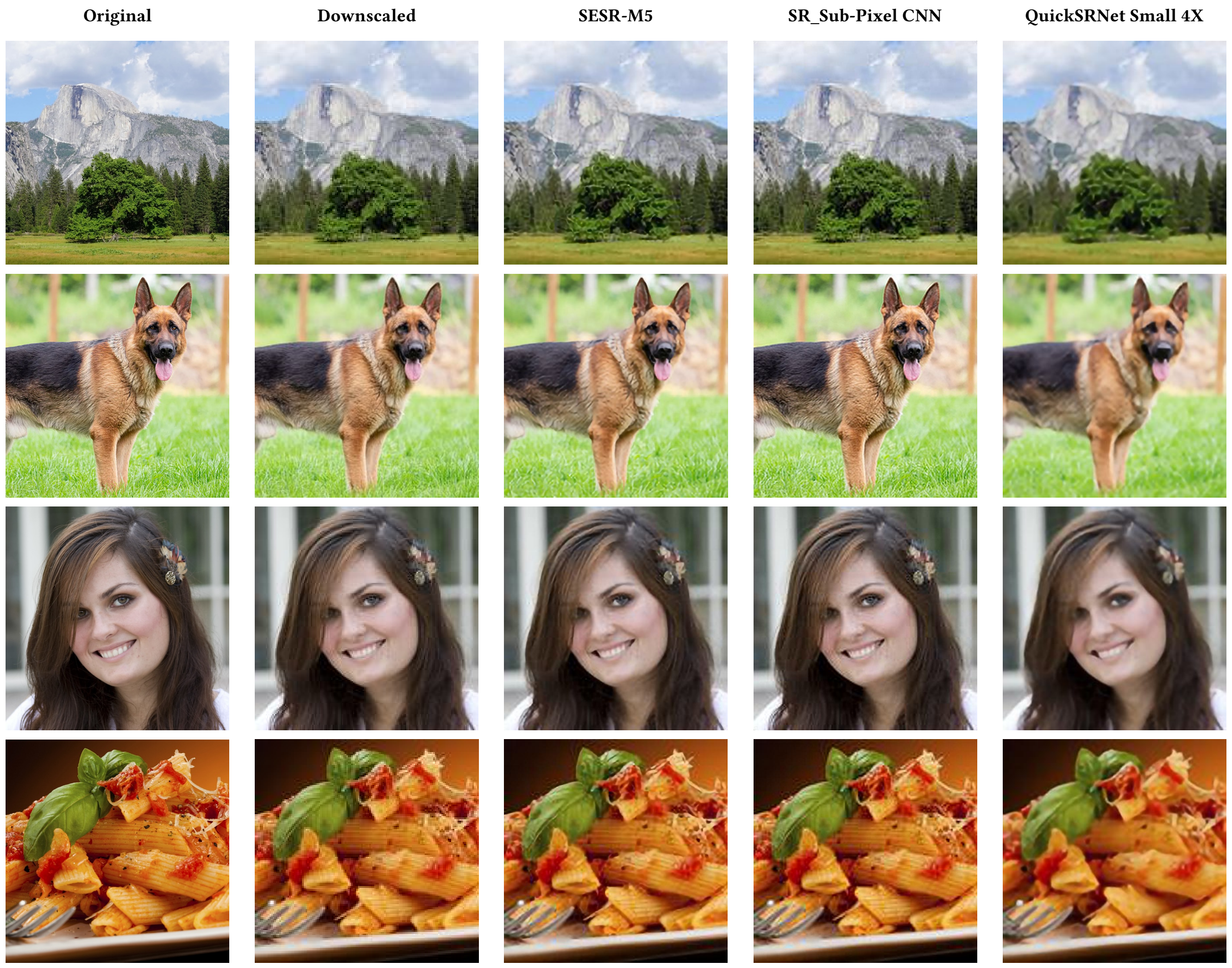


Figure 8: Comparison of Original, Downscaled, SESR-M5, SR_Sub-Pixel CNN, and QuickSRNet Small 4X outputs for the four test images. Zoom in to see the quality difference.

[9] Kartikeya Bhardwaj, Dibakar Gope, James Ward, Paul Whatmough, and Danny Loh. 2024. Super-Efficient Super Resolution for Fast Adversarial Defense at the Edge. In *Proceedings of Arm Research Symposium.*

[10] TensorFlow Blog. 2019. TensorFlow Model Optimization Toolkit. https://blog.tensorflow.org/2019/08/tensorflow-model-optimization-toolkit_5.html.

[11] Moumena Chaqfeh, Rohail Asim, Bedoor AlShebli, Muhammad Fareed Zaffar, Talal Rahwan, and Yasir Zaki. 2023. Towards a world wide web without digital inequality. *PNAS* 120, 3 (2023), e2212649120.

[12] ONNX Contributors. 2024. ONNX: Open Neural Network Exchange. https://onnx.ai/.

[13] TensorFlow.js Contributors. 2024. TensorFlow.js: Machine learning for the web. https://www.tensorflow.org/js.

[14] Stefan Judis and Eric Portis. 2025. Media: 2024: The web almanac by HTTP archive. https://almanac.httparchive.org/en/2024/media

[15] Conor Kelton, Matteo Varvello, Andrius Aucinas, and Ben Livshits. 2021. BrowseLite: A Private Data Saving Solution for the Web. In *Proceedings of the Web Conference 2021 (WWW '21).* ACM, 305–316. https://doi.org/10.1145/3442381.3449885

[16] James Newman, Robert Belson, and Fabián E Bustamante. 2019. Scaling Up Your Web Experience, Everywhere. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications.* 87–92.

[17] Ana Maria Rodriguez. 2022. Mobile data cost have increased, making internet connectivity unaffordable for many. https://a4ai.org/news/mobile-data-cost-have-increased-making-internet-connectivity-unaffordable-for-many/2022.

[18] World Wide Web Consortium (W3C). 2015. File API - W3C Recommendation. https://www.w3.org/TR/FileAPI/.

[19] Pengfei Wang, Matteo Varvello, Chunhe Ni, Ruiyun Yu, and Aleksandar Kuzmanovic. 2021. Web-lego: trading content strictness for faster webpages. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications.* IEEE, 1–10.