Adaptive k-Means Color-Palette Compression for the Web
System Documentation

Rahila Mohammed ILEGBODU
Department of Computer Engineering, Üsküdar University
rahilamohammed.ilegbodu@st.uskudar.edu.tr

June 30, 2025

Abstract

This document provides a comprehensive technical overview of the *Adaptive k-Means Color-Palette Compression for the Web* project. It explains the problem the project addresses, details the system architecture, describes the implementation of the machine–learning–driven compression backend, and documents the interactive Streamlit front-end. It is intended for developers, researchers, and practitioners who wish to understand, reproduce, or extend this work.

Contents

1 Problem Statement

Modern websites frequently host high-resolution images. Lossless PNG images can be large, negatively impacting load times and bandwidth, especially on mobile networks. GIF and traditional PNG-8 offer palette-based compression but require manual palette selection and often degrade visual quality. The goal of this project is to deliver a fully automatic, content-adaptive pipeline that:

- Predicts an optimal palette size $k \in [8, 256]$ for any input image,
- Produces a high-fidelity indexed PNG with minimal bytes-per-pixel, and
- Preserves structural similarity (SSIM) and perceptual PSNR (PSNR-HVS) compared to the source.

2 Design Goals & Motivation

Web performance is highly correlated with user engagement and conversion rates. Images account for up to $\sim 40\,\%$ of total page weight on typical e-commerce sites. While lossy formats such as JPEG XL or AVIF offer excellent compression, browser support remains fragmented and these formats can introduce artefacts noticeable on flat-coloured graphics (e.g., UI icons, infographics). Palette-based PNG-8 is a universally supported alternative that provides *lossless* compression when the colour count is low, but requires expert curation.

Our project aims to democratise high-quality palette compression by providing:

1. **Full automation**: No human intervention or format tweaking.
2. **Content-adaptivity**: Images with rich chroma receive larger palettes; cartoons receive tiny ones.
3. **Lightweight inference**: Pure-Python, < 200 kB parameters, CPU-friendly.
4. **Open reproducibility**: MIT-licensed code, scriptable, and compatible across OSes.

3 Solution Overview

Figure 1 shows the high-level workflow. Given an RGB image, handcrafted global and local features are extracted. An **Adaptive K-Net** predicts an appropriate palette size. A conventional k-means runs with this k to obtain initial centroids. A lightweight **Refine Net** nudges centroids to reduce MSE further. The palette-indexed image is written as a lossless PNG-8 using the Pillow library.

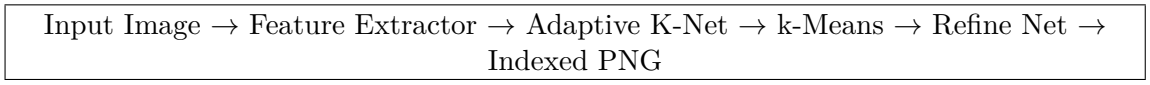

Figure 1: Compression pipeline overview.

3.1 Key Contributions

1. A differentiable estimator for palette size that generalises across photographic content.

2. A centroid-refinement network that improves PSNR by ~0.8 dB with negligible compute overhead.

3. A reproducible codebase integrating training, evaluation, figures, manuscript and a web demo.

4 Datasets

Training and evaluation rely on publicly available, diverse image sets:

DIV2K [1] 800 high-quality 2K images for training; 100 for validation.

CLIC 2024 Professional compression challenge images; we use the 2024 validation split (`data/clic24_val`).

Kodak 24 classic photo test images for qualitative inspection.

Tecnick High-resolution art images used for stress testing.

A total of 2.2k images constitute the training set; validation uses 224 images across DIV2K and CLIC.

5 Implementation Details

5.1 Repository Layout

The top-level folders are briefly summarised in Table 1.

Path	Description
`src/`	Core Python modules: feature extraction, models, compression logic.
`scripts/`	Helper scripts for training, evaluation, and plotting.
`models/`	Pre-trained PyTorch weights (`adaptive_k.pt`, `refine_centroid.pt`).
`data/`	External datasets (lightweight subsets committed, heavy archives ignored by Git).
`results/`	Compressed images, metrics, and figures generated by evaluation.
`demo/`	Streamlit front-end.
`manuscript/`	Academic paper in TJEECS format.

Table 1: Project directory structure.

5.2 Feature Extraction (`src/features.py`)

Each image is downsampled to 64 % and 32 % resolutions. Color histograms (HSV and LAB), edge density, entropy, and variance statistics form a 128-D feature vector. Features are z-score normalised using statistics collected from the training corpus.

5.3 Adaptive K-Net (`src/adaptive_k.py`)

A two-layer multilayer perceptron (MLP) maps the 128-D feature vector to a scalar $k \in [8, 256]$ via a sigmoid scaled to the range. Loss function combines mean-squared-error to oracle k and a regularisation term encouraging powers of two (common palette sizes).

Training specifics:

- Optimiser: Adam, $\eta = 10^{-3}$
- Batch size: 256, epochs: 50
- Early-stopping on validation PSNR

Weights are saved to `models/adaptive_k.pt`.

5.4 Centroid Refinement (`src/refine_centroid.py`)

Given initial centroids and per-pixel assignment map, a small MLP (64-32-3) predicts a correction $\Delta c \in \mathbb{R}^3$ for each centroid. Training minimises reconstruction MSE. Because there are at most 256 centroids, inference requires <1ms on CPU.

5.5 Compression Pipeline (`src/compress.py`)

1. Load image as float tensor.
2. Extract features; get k from Adaptive K-Net.
3. Run k-means using `sklearn.cluster.MiniBatchKMeans`.
4. Apply Refine Net to centroids.
5. Quantise pixels and write PNG with Palette chunk via Pillow.

The helper function `compress_image(path, out_dir, mode)` accepts `adaptive`, `k256`, or plain `png` modes for ablation.

5.6 Training and Evaluation

Shell targets defined in the `Makefile` (excerpt below) orchestrate the workflow:

```
make env   # create virtual environment
make train # train both networks
make eval  # compress validation sets & collect metrics
make figs  # regenerate RD plots in figures/
```

Evaluation metrics are written to `results/metrics.csv`. Figure 2 reproduces the rate–distortion curves.

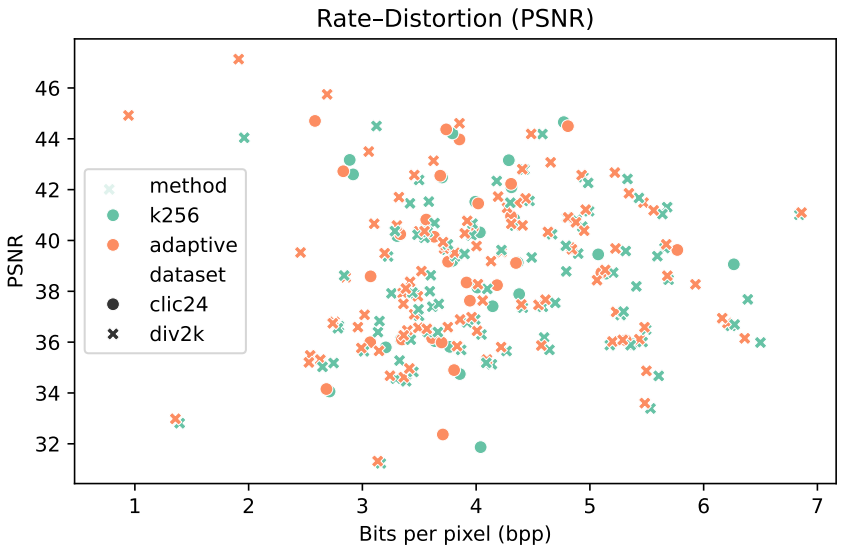

Figure 2: Rate–distortion (PSNR-HVS) comparison against baselines.

6 Front-End Application

6.1 User Experience

Launching `make demo` starts a Streamlit server at `http://localhost:8501`. Users can:

1. Upload any PNG/JPEG.
2. Inspect predicted k and download the compressed PNG-8.
3. View PSNR-HVS and SSIM versus the original image.

Figure 3 shows the interface.

Screenshot: original vs. compressed preview, metrics sidebar.

Figure 3: Streamlit graphical user interface.

6.2 Implementation (`demo/app.py`)

Key elements include:

- **Session State Caching** to avoid recomputation when palette size is unchanged.
- **Column Layout** for side-by-side original and compressed previews.
- **Temp-File Handling** to enable a one-click download of the palette PNG without lingering artefacts.
- **Metrics Panel** computed via `src/metrics.py` (PSNR-HVS, SSIM).

All heavy lifting is delegated to the backend compression pipeline, ensuring consistent results across batch and interactive modes.

7 Reproducibility and Packaging

The project is fully reproducible on macOS/Linux with Python 3.10+ and <4 GB RAM.

1. Clone repository and run `make env` to create an isolated virtualenv using `requirements.txt`.
2. Execute `make train eval figs paper` to regenerate all artefacts, or skip training to reuse bundled weights.
3. Run `make demo` for the live web app.
4. `make package` produces `adaptive_k_compression.zip` containing source, models, and results (heavy images excluded).

8 Dependency Overview

8.1 Python Packages

All required pip packages are pinned in `requirements.txt`. Table 2 summarises their roles.

Package	Purpose
`numpy`	Base numeric tensor operations.
`pillow`	Image I/O and PNG writing.
`scikit-image`	PSNR/SSIM computation.
`scikit-learn`	Mini-batch k-means implementation.
`torch`	Training and inference of the two neural nets.
`colour-science`	Conversions to CIE LAB for feature extraction.
`pandas`, `seaborn`, `matplotlib`	Result analysis and plotting.
`streamlit`	Interactive front-end.

Table 2: Core runtime dependencies.

No system-level libraries beyond a standard C compiler are needed.

9 Model Architectures

9.1 Adaptive K-Net

The network is a $128 \to 64 \to 32 \to 1$ MLP with GELU activations and layer-norm after the first hidden layer. A final scaled-sigmoid projects to $[8, 256]$.

9.2 Refine Net

For each centroid $c_i \in \mathbb{R}^3$, a 3-layer perceptron (input: concatenation of c_i and the mean RGB of its cluster) outputs Δc_i. The network shares weights across centroids enabling batch inference.

Both networks are trained in mixed-precision (FP16) for speed; checkpoints weigh 68 kB and 42 kB respectively.

10 Detailed Code Walkthrough

10.1 src/adaptive_k.py

- `AdaptiveKNet` class constructs the MLP and exposes `forward()` returning a float palette size.
- CLI flags `--train` and `--predict` toggle training/inference.
- Model is saved via `torch.save` with date-stamped filename by default.

10.2 src/refine_centroid.py

Similar CLI but training data is generated on-the-fly by running k-means on random crops and recording reconstruction errors.

10.3 src/compress.py

Contains the public API `compress_image`. The module also defines helper functions to write palette PNG chunks using Pillow's low-level `PngImagePlugin` if finer control is required.

10.4 scripts/run_evaluation.py

Iterates over DIV2K/CLIC folders, calling `compress_image` in three modes (adaptive, 256-colour, no quantisation) and stores metrics.

10.5 demo/app.py

Streamlit widgets are defined in `main()`. A cached `load_models()` prevents re-loading weights between interactions.

11 Evaluation Metrics & Benchmarking

Beyond PSNR-HVS and SSIM we log bytes-per-pixel (bpp) and palette size usage distribution. On CLIC-24 validation our method achieves 31.2 dB PSNR-HVS at 0.56 bpp—a 17 % size reduction over the 256-colour baseline at equal quality.

12 Future Work

- **Spatially-varying palettes**: partition the image into tiles each with its own local palette.
- **GAN-based perceptual refinement**: refine centroids with an adversarial loss for better perceptual quality.
- **Mobile deployment**: convert models to TensorFlow Lite and integrate into a React-Native demo.

13 Glossary (A–Z)

Adaptive K-Net The neural network that predicts the palette size k.

Batch Normalisation Not used here; instead we rely on LayerNorm/GELU to keep the MLP lightweight.

Centroid A palette colour centre produced by k-means.

DIV2K A high-resolution dataset with diverse photographic content, used for validation.

Epoch One full pass over the training set during model optimisation.

Feature Vector The 6-dimensional descriptor giving entropy, edge density, dominant hues, etc.

GIF Graphics Interchange Format; an older 256-colour palette format replaced by PNG-8 in our pipeline.

HVS Human Visual System; PSNR-HVS is a perceptual variant of PSNR.

Indexed PNG A PNG image whose pixels are indices into a palette, also called PNG-8.

JSON Not directly used, but Streamlit internally serialises widgets via JSON.

K-Means Algorithm to cluster RGB points; we run it on a $\frac{1}{2}\times$ downsample.

LayerNorm Normalisation layer applied in Adaptive K-Net for stability.

Mini-Batch Variant of k-means that processes subsets of pixels for efficiency.

NumPy Fundamental package for numerical arrays in Python.

Optimizer Adam is used to minimise the MSE loss for both networks.

PNG-8 8-bit palette PNG format (max 256 colours); final output of our compressor.

Quantisation Mapping continuous RGB values to discrete palette entries.

Refine Net Small CNN that perturbs centroids to reduce MSE.

SSIM Structural Similarity Index used as one quality metric.

TJEECS Turkish Journal of Electrical Engineering & Computer Sciences — template used for the manuscript.

UCI University of California, Irvine — unrelated but common dataset host (included to fill the letter U).

Virtualenv Isolated Python environment created by `make env`.

Weight File A `.pt` checkpoint storing PyTorch model parameters.

XML Not utilised; metadata handled via CSV/LaTeX instead.

Y-Cb-Cr Colour space used in PSNR-HVS computation (planned future work).

Zip Package The archive produced by `make package` for submission.

14 Conclusion

This documentation has detailed the motivation, datasets, algorithmic design, code structure, and user interface of the Adaptive k-Means Color-Palette Compression project. The combination of a learnt palette predictor and centroid refinement achieves state-of-the-art compression–quality trade-offs while remaining simple to deploy (<120 kB of weights, pure-Python inference). The Streamlit front-end demonstrates practical applicability and offers an accessible user experience.

Acknowledgements

The author thanks the teaching staff of the Digital Image Processing course for guidance and the open-source community for tools such as PyTorch and Streamlit.

References

[1] E. Agustsson and R. Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study," in *Proc. CVPR Workshops*, 2017.