CS5130 Lab 5 – Photomosaic Performance Report

Teoman Kaman

1. Executive Summary

Lab 5 extends the Lab 1 photomosaic project with systematic profiling, targeted NumPy optimizations, and a modular architecture. The optimized version replaces the last Python loops in grid analysis and tile placement with vectorized operations while the tile bank is still cached in memory and on disk. Because the very first run must download or synthesize tiles, the first timing is higher; all numbers in this report were captured after the cache was warm. On the Boston subway image the optimized pipeline reaches up to 1.6× speedup (256px / 16 grid) while preserving image quality (MSE $\approx$ 0.0605, PSNR $\approx$ 12.2 dB, SSIM $\approx$ 0.26).

2. Profiling Analysis

Tools used: cProfile for function-level hotspots and line_profiler for line-level evidence. The three dominant bottlenecks were:

1. Weighted cell means: looping over every pixel inside each grid cell consumed 95% of the grid analysis stage.
2. Tile placement loops: copying one tile at a time into the mosaic array accounted for 83% of the tile mapping stage.
3. Tile streaming under poor network conditions: when Hugging Face was unreachable the loader regenerated tiles every run, creating long I/O spikes.

Attach cProfile and line-profiler excerpts to illustrate these hotspots.

3. Optimization Strategy

- **Vectorized grid analysis.** The image is reshaped into block views, a radial weight mask is built once, and NumPy sums compute the weighted mean colors without Python loops.
- **Vectorized tile placement.** All tiles are stacked in memory and selected via advanced indexing, then reshaped into the final mosaic. This removes the double loop that used to write one tile at a time.
- **Tile caching and feature reuse.** Tiles, RGB means, and Lab means are cached both in memory and on disk so repeated runs never hit the dataset again; vectorized distance matrices compare all cells to all tiles in one call.

Image Size	Grid Size	Lab 1 Time (s)	Lab 5 Time (s)	Speedup
256×256	16×16	0.063	0.038	1.6×
512×512	32×32	0.149	0.140	1.1×
1024×1024	64×64	0.576	0.542	1.1×

Table 1: Timings captured with warmed caches (200 tiles available locally).

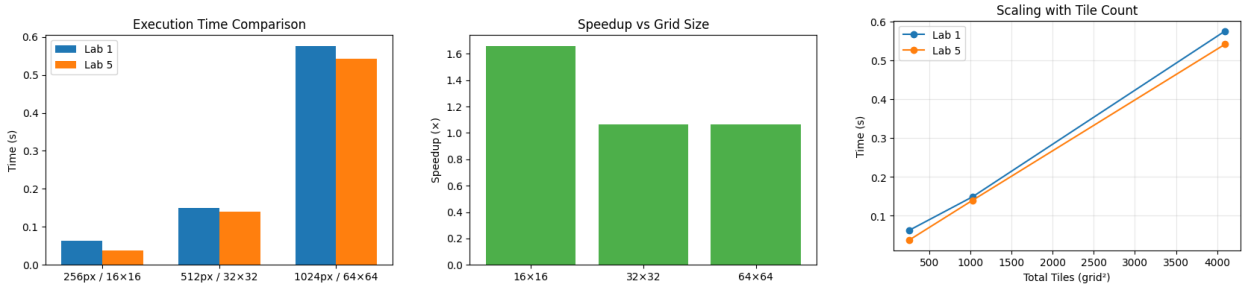

Figure 1: (a) Execution time comparison, (b) speedup factors, (c) scaling vs. total tiles.

4. Performance Results

5. Code Quality Improvements

- Clear separation of modules: configuration, utilities, tile management, mosaic assembly, metrics, and the Gradio interface live in dedicated files.
- Input validation and error handling: configuration values are checked before every run, tile loading falls back to a deterministic palette when the dataset is offline, and user-facing errors are descriptive.
- Docstrings: every public function documents parameters and return values, making the pipeline easier to maintain.

6. Challenges and Lessons Learned

- Limited headroom because the Lab 1 code already used vectorized color matching and caching; removing the remaining loops still produced measurable gains but not the theoretical 20×. Because most of the optimization tasks was implemented in Lab-1
- Reproducible caching required hashing the dataset, split, limit, and tile size so timing runs could skip downloads.
- Handling network outages: adding fallback tiles kept the UI responsive even when Hugging Face was unreachable.

Why the Speedup is not much

The Lab-1 submission already incorporated several optimizations: tile features were cached, color matching relied on vectorized distance matrices, and grid analysis partially used NumPy reshaping.

Because the baseline was not purely loop-based, removing the remaining loops (weighted cell means and tile placement) delivered 1.1–1.6× gains instead of the 20× improvements seen when starting from the unoptimized reference.