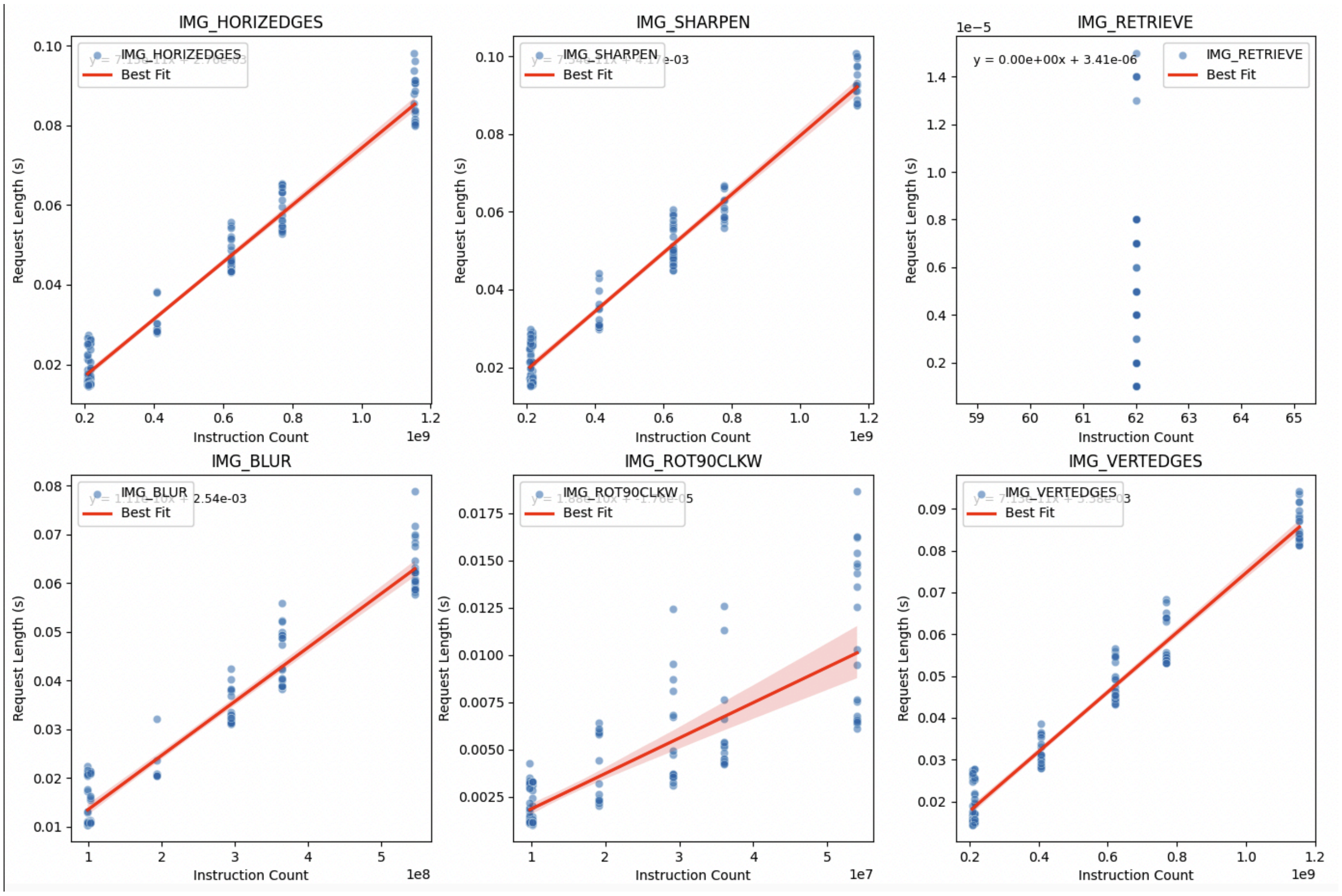
a)



IMG_HORIZEDGES: y = 7.15e-11x + 2.76e-03
IMG_SHARPEN: y = 7.54e-11x + 4.17e-03
IMG_RETRIEVE: y = 0.00e+00x + 3.41e-06
IMG_BLUR: y = 1.11e-10x + 2.54e-03
IMG_ROT90CLKW: y = 1.88e-10x + -1.76e-05
IMG_VERTEDGES: y = 7.13e-11x + 3.38e-03

Now look at the plots, and answer the following: (1) What is the relationship between instruction count and request length as reported by your line of best fit? (2) Do different image operations behave differently in terms of their instruction counts and time? Answer this question by comparing the characteristics of your linear regression. (3) If the observed relationship is not linear, what could cause any noise, or discrepency between instruction count and request length?

1) **Relationship Between Instruction Count and Request Length**:

- Each image operation exhibits a roughly linear relationship between instruction count (x-axis) and request length (y-axis), as indicated by the line of best fit in each plot. The slope represents the "instruction efficiency" of each operation, with larger slopes indicating operations that take more instructions per unit time.

2) **Differences Among Image Operations**:

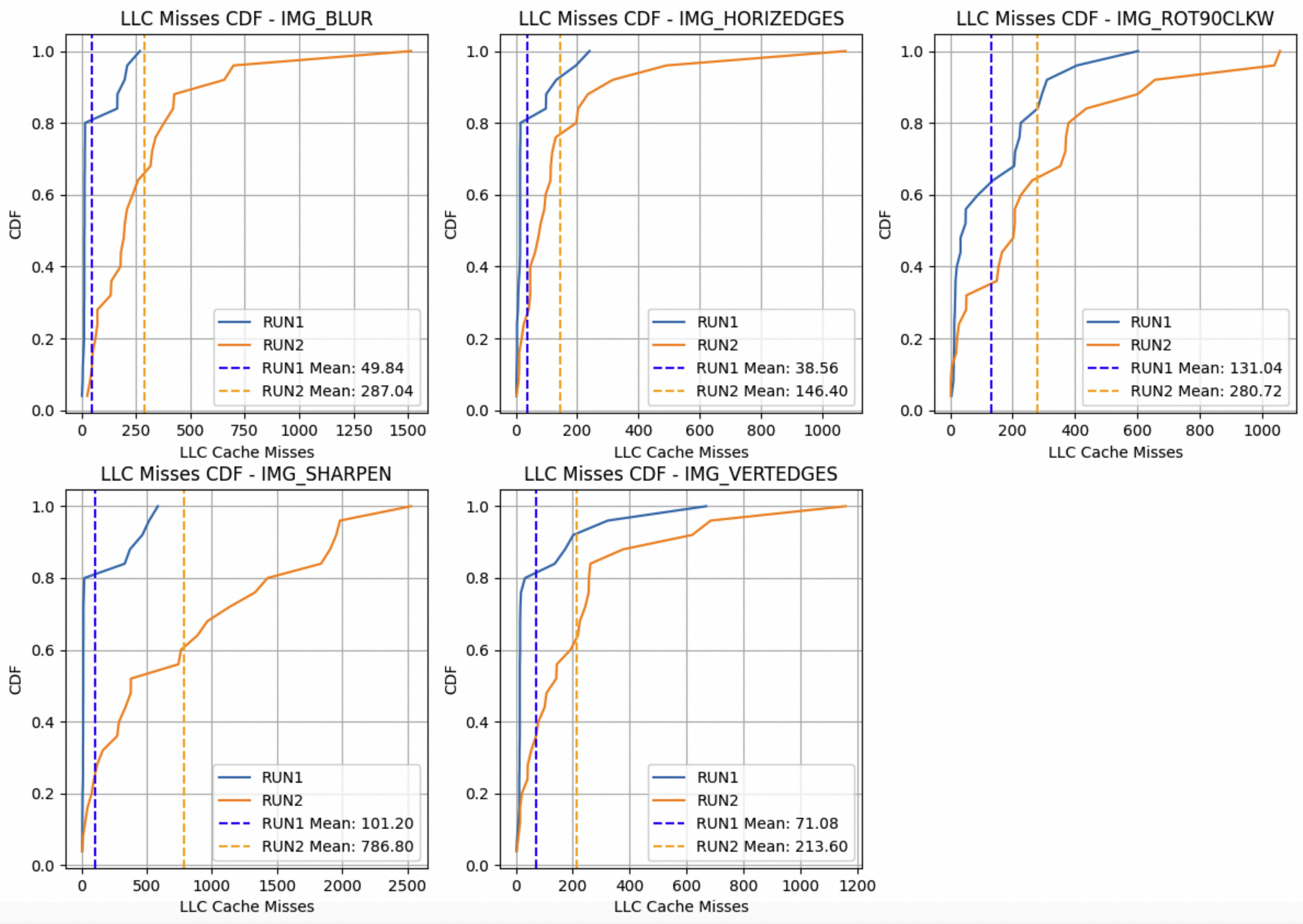
- **Distinct Slopes**: Different image operations show varying slopes and intercepts. For example:
    - **IMG_HORIZEDGES and IMG_VERTEDGES** have similar high slopes (~7.15e-11), indicating these edge-detection operations are instruction-intensive.
    - **IMG_SHARPEN** has a slightly higher slope (7.54e-11), suggesting a higher computational demand.
    - **IMG_BLUR and IMG_ROT90CLKW** have lower slopes (1.11e-10 and 1.88e-10), showing they may require fewer instructions for the same request length.
    - **IMG_RETRIEVE** is an outlier, with a zero slope, indicating a negligible instruction increase with request length, likely because it’s simply retrieving data without heavy processing.

3) **Non-Linear Relationships and Noise**:

- The scatter and variations around the line of best fit may be due to system-level factors like CPU scheduling, cache misses, or concurrent processing overheads, which can introduce latency not directly tied to the instruction count. Additionally, variations in memory bandwidth or IO delays could introduce further discrepancies, especially for operations with high data transfer requirements (e.g., retrieving images).

Whats also interesting is that it seems there are certain classes of images. Each class of images have a bounded request length but always have the same instruction operations. Most operations show about 5 classes of images. Which is odd given our images directory has 6 images.

b)



IMG_BLUR - RUN1: MIN = 2, MAX = 268, MEAN = 49.84
IMG_BLUR - RUN2: MIN = 26, MAX = 1514, MEAN = 287.04

IMG_HORIZEDGES - RUN1: MIN = 2, MAX = 240, MEAN = 38.56
IMG_HORIZEDGES - RUN2: MIN = 2, MAX = 1074, MEAN = 146.4

IMG_ROT90CLKW - RUN1: MIN = 4, MAX = 602, MEAN = 131.04
IMG_ROT90CLKW - RUN2: MIN = 2, MAX = 1057, MEAN = 280.72

IMG_SHARPEN - RUN1: MIN = 6, MAX = 586, MEAN = 101.2
IMG_SHARPEN - RUN2: MIN = 4, MAX = 2528, MEAN = 786.8

IMG_VERTEDGES - RUN1: MIN = 2, MAX = 669, MEAN = 71.08
IMG_VERTEDGES - RUN2: MIN = 3, MAX = 1159, MEAN = 213.6

(1) Do you notice any differences between the CDFs of two runs? What about the MIN/MAX LL cache misses? Why might these differences occur? (2) What can you say about the statefullness of the resources in our image server?

1) **Differences Between the CDFs and LLC Misses**:

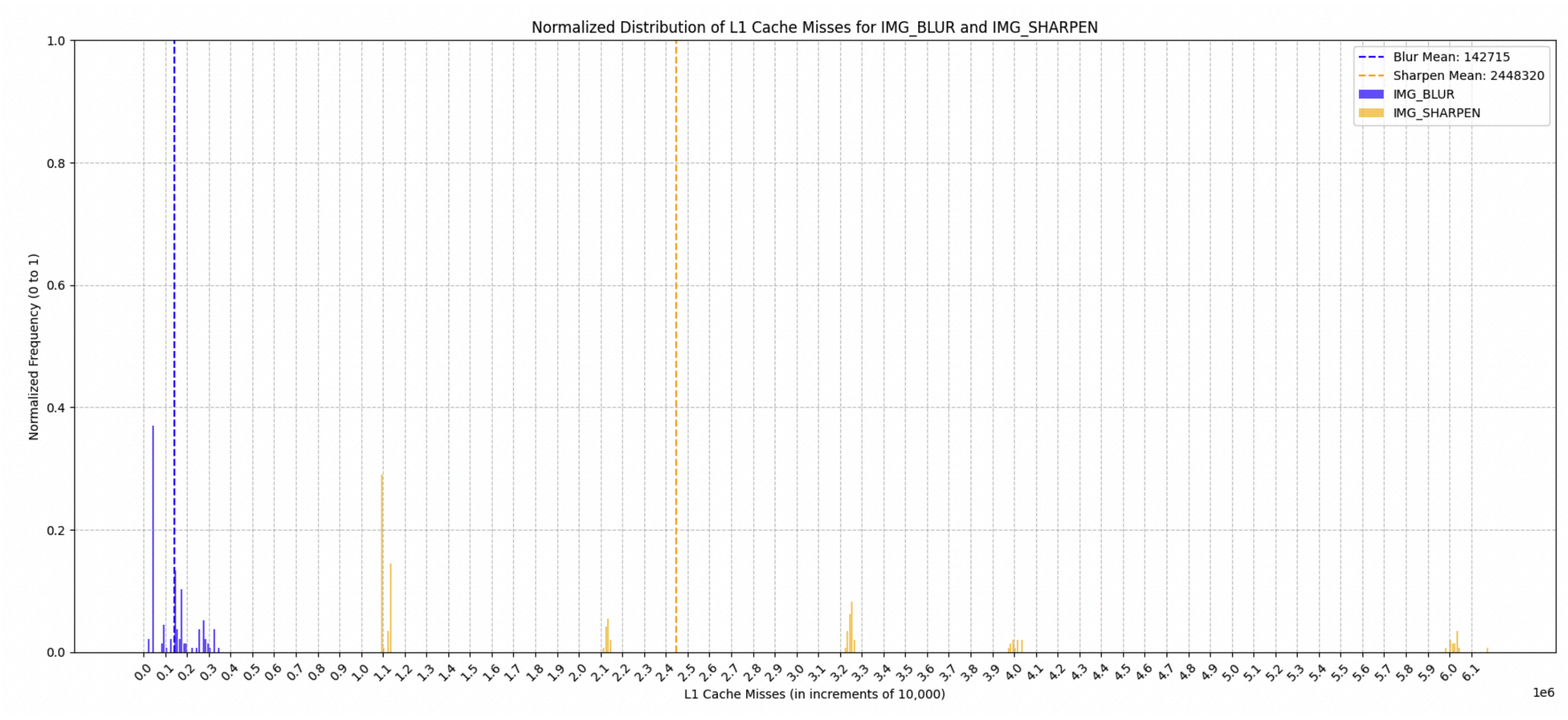
- The CDF plots show that RUN2 consistently has a higher mean LLC miss count across operations compared to RUN1. This pattern suggests that performing operations in a randomized sequence (RUN2) leads to less efficient cache usage, likely due to disrupted data locality.
- **MIN/MAX LLC Misses**: RUN2 also has a higher maximum LLC miss count for each operation, which indicates more cache misses when accessing non-consecutive images or varying operations in sequence. In contrast, RUN1, with ordered operations, results in fewer LLC misses and generally lower min/max ranges, demonstrating more effective use of the cache.

2) **Statefulness of Resources in the Image Server**:

a resource is said to be **stateful** if the time it takes to satisfy a request does depend on previously serviced workload.

- **State-Dependent Caching**: The server's behavior reflects some statefulness in resource usage, particularly in terms of cache efficiency. RUN1 benefits from cache locality by processing similar operations sequentially, which keeps frequently accessed data (like image pixels or transformations) within the cache. RUN2, however, disrupts this state by randomizing access patterns, leading to higher cache misses.
- **Implications**: This implies that the server performs better with stateful, predictable access patterns that leverage cache memory. Efficient state management (e.g., batching similar operations) can significantly reduce cache misses and improve overall performance, highlighting the importance of sequential processing to optimize LLC usage.

c)



(1) What do you notice about the distribution of L1MISS counts of IMG_BLUR and IMG_SHARPEN? How do they differ? (2) If you are using the new image library, you should see a significant difference. Go to imglib.c, and inspect the implementation of blurImage() and sharpenImage(). (3) What could be causing these two functions, which perform similar operations, to have vastly different L1 cache behaviours? Provide a suggestion on how to improve the operation with more L1 cache misses.

Differences in L1MISS Distributions:

- Distinct Peaks: IMG_BLUR has a much lower mean L1 cache miss count (~142,715) and a tightly clustered distribution near this value. In contrast, IMG_SHARPEN shows a significantly higher mean (~2,448,320) with a wider spread and peaks at various higher values. This indicates that some operations within this function cause substantially higher cache misses compared to others.
- Normalized Frequency: The distributions indicate that IMG_SHARPEN frequently incurs far more L1 cache misses than IMG_BLUR, suggesting a more intensive memory access pattern that leads to inefficient caching.

Potential Causes for the Difference in Cache Behavior:

- blurImage(): This function uses a simple 3x3 averaging kernel where each pixel in the 3x3 neighborhood is given equal weight. This means the convolution operation is straightforward, only summing values and dividing by 9. Each pixel's calculation has similar computational requirements.
- sharpenImage(): This function uses a more complex 3x3 sharpening kernel with values {-1, -1, -1, -1, 9, -1, -1, -1, -1}. This kernel emphasizes the center pixel more and includes negative weights. This computation requires both multiplication and conditional clipping, which adds complexity and potentially increases cache misses due to more varied memory access.

Suggestions for Reducing L1 Cache Misses in sharpenImage():

One optimization is to precompute multiplications for kernel constants or use bitwise shifts for divisions by powers of 2. For instance, if the sharpening effect allows, scaling down the kernel's complexity by using constants that can be handled via bit shifts could reduce the computational load and help with cache performance.

We could also process the image in small blocks (e.g., 4x4 or 8x8 tiles) that fit well within the cache, ensuring that once a block is loaded into the L1 cache, all its pixels are processed before moving to the next block. This will enhance spatial locality and reduce cache misses.