Alice: Low-latency Image Live Co-editing via Adaptation

Anlan Zhang[†], Stefano Petrangeli[‡], Haoliang Wang[‡], Yu Shen[‡], Feng Qian[†]
[†]University of Southern California [‡]Adobe Research

Abstract

Image live co-editing (LCE), which allows users to edit a shared image concurrently and remotely, is rising in popularity. However, fluctuating resources (*i.e.,* bandwidth and computation), as well as varying degrees of edit complexity, make it challenging to achieve low-latency image live co-editing, which drastically degrades the user experience. To address this issue, we propose Alice, a cross-platform compression adaptation framework that incorporates three core designs. First, Alice leverages both *data-based* (*i.e.,* sending compressed pixels) and *operation-based* (*i.e.,* sending image editing operation APIs and corresponding parameters) approaches for image edit transmission. Second, Alice combines diverse modern lossless compression techniques and their various configurations to enhance the adaptability of *data-based* transmission. Third, Alice features a lookup table (LUT)-based decision framework to determine the best transmission strategy for image edits in real time. We implement Alice and integrate it into our image LCE testbed. Our extensive evaluation shows that, compared to the baselines using a fixed transmission strategy, Alice achieves up to 95% latency reduction with negligible overhead.

CCS Concepts

• **Information systems** → **Multimedia streaming**.

Keywords

Image Live Co-editing, Low Latency, Adaptation, Compression

ACM Reference Format:
Anlan Zhang[†], Stefano Petrangeli[‡], Haoliang Wang[‡], Yu Shen[‡], Feng Qian[†]. 2025. Alice: Low-latency Image Live Co-editing via Adaptation. In *The 35th edition of the Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '25), March 31-April 4, 2025, Stellenbosch, South Africa.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3712678.3721883

1 Introduction

Live co-editing (LCE) applications [14–17] have become crucial for boosting workplace productivity. Emerging image LCE tools [9], which are newer and less studied compared to their text-based counterparts, extend these capabilities to image editing. Ideally, image LCE tools should possess three essential features to enhance work flexibility and enable efficient collaboration from anywhere at any time: *lossless* information transmission, *low-latency* interactions, and *scalability*. However, image LCE systems are still in their early stages and lack substantial research. This work conducts a latency-focused study of cloud-based image LCE systems. We focus on achieving a low-latency user experience, where latency is defined as *the elapsed time between when a user makes an edit to a shared image on their local machine and when the edit appears on the devices of other users.*

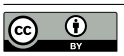
Many networked multimedia applications, such as real-time communication and video streaming, face similar challenges as those mentioned above. Existing approaches to reducing end-to-end multimedia transmission latency can be categorized into three main strategies: (1) *Reducing in-network queuing delay.* Numerous studies [31, 41, 43] focus on minimizing queuing delays on the network side; (2) *Adaptive streaming.* Adaptive bitrate (ABR) algorithms are widely used to ensure timely video delivery under fluctuating bandwidth conditions [25, 34, 40]; and (3) *Accelerating compression.* Some studies [38] reduce image compression latency by leveraging high-performance computing units, such as GPUs.

The above approaches provide benefits but have limitations in the context of image live co-editing. *Reducing in-network queuing delay* can be helpful, but overall latency may still be constrained by network limitations due to the low compression ratio of lossless compression. *Adaptive streaming* relies on lossy compression, which is incompatible with image LCE's requirement for lossless compression, limiting its applicability. *Accelerating compression* through extensive parallelization demands high on-device computational power, which may not always be available. Additionally, network constraints may still impose latency limits due to the restricted compression ratio of lossless methods.

In this paper, inspired by *adaptive video streaming*, we achieve low-latency image live co-editing by incorporating extensive adaptation capabilities into the image LCE system. These capabilities are enabled by various image edit compression and transmission strategies. The key challenge we face, as mentioned above, is that the commonly used *data-based* transmission – where the (compressed) pixel data of an edit is sent to other users – has limited adaptability due to the mathematical constraints of lossless compression [37]. We address this challenge through the following novel **designs**.

• Joint Use of *Data-based* and *Operation-based* Strategies (§4, §6.1). Our pilot study on real user image editing patterns (§4) reveals opportunities to leverage an alternative strategy, referred to as *operation-based*, for image edit transmission, where the editing operation's metadata (*e.g.,* API and parameters) is sent to other users, who then "replay" the edit on their local copies. To optimize image edit transmission, we strategically switch between *data-based* and *operation-based* approaches based on available bandwidth and computational resources.

• Leveraging Multiple Lossless Compression Techniques with Various Configurations (§5, §6.2). Numerous lossless compression techniques have been developed, each exhibiting significant variability in both compression ratio and efficiency within the context of image LCE (§5). To enhance the adaptability of *data-based* transmission, we strategically combine these techniques and their configurations.



NOSSDAV '25, Stellenbosch, South Africa

ACM ISBN 979-8-4007-1469-6/2025/03
https://doi.org/10.1145/3712678.3721883

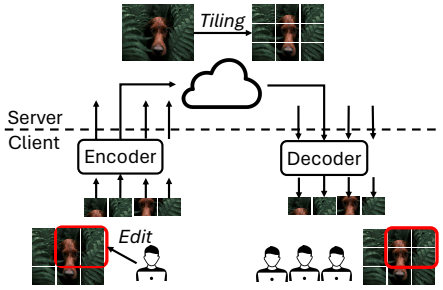


Figure 1: Cloud-based image LCE system.

• Real-time Strategy Selection (§6.2). We develop a unified lookup table (LUT)-based approach for rapid selection of the optimal transmission strategy, built through extensive offline profiling.

We **implement** the above components into Alice, a latency-aware, cross-platform image edit transmission framework. To evaluate Alice, we integrate it into our cloud-based image LCE testbed. Our extensive trace-driven evaluation (§7) demonstrates that: (1) Compared to baselines with fixed transmission strategies, Alice achieves up to 95% per-tile latency reduction, with an average (median) per-tile latency of 160ms (85ms); (2) Across diverse system setups (e.g., varying tile resolutions and user counts), Alice consistently outperforms baseline strategies, delivering 28%–85% latency reduction; and (3) Alice incurs negligible overhead (< 1 ms) on commodity devices (Ubuntu desktop and MacBook Air 2020).

Our **contributions** include: (1) To the best of our knowledge, this is the first study to focus on low-latency image live co-editing, addressing the problem from a system perspective; (2) The design of Alice, including its hybrid transmission strategy and the LUT-based adaptation algorithm; and (3) The implementation, integration, and thorough evaluation of Alice on our self-developed testbed.

2 Background and Motivation

2.1 Cloud-based Image LCE System

We begin by introducing the cloud-based image LCE system, which serves as the foundation of this paper. As shown in Figure 1, the system consists of a server and multiple clients. All shared images are initially stored on the LCE server. When a new image LCE session starts, the server distributes the shared image to all participating clients. Once users have local copies, they can begin editing. LCE clients capture user edits on the local image, send these edits to the server, and apply edits received from the server to their local copies. The LCE server manages conflicts arising from concurrent edits by different clients and distributes valid edits to all users. [1]

The image LCE system transmits image edits by sending their pixel data over the network, optionally compressed using lossless techniques. We refer to this approach as *data-based* transmission. Specifically, a shared image is spatially segmented into smaller, non-overlapping tiles, each with a lower resolution (*e.g.,* 512×512, 1024×1024, *etc.*), and a single channel, unlike the multi-channel full image. Compression and transmission of image edits occur on a per-tile basis, involving only the tiles affected by the edit (*e.g.,* the red area in Figure 1). This design is inspired by viewport-adaptive streaming in immersive video systems [25, 34], which transmits only the content within the viewer's viewport to conserve bandwidth. In the context of image LCE, the area affected by the edit serves as a conceptual "viewport" for transmission. Our user study results (§4) further validate this design choice.

[1]We assume that conflict management is efficient and does not introduce a significant end-to-end latency bottleneck. Therefore, we do not discuss its details in this paper.

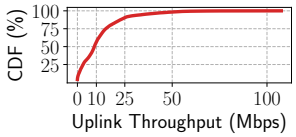


Figure 2: CDF of FCC mobile uplink throughput in Jan. 2023 [13].

2.2 Resource Uncertainty In Image LCE

Achieving consistently low-latency performance in image LCE systems is challenging, especially in mobile use cases, due to the following factors: (1) Unreliable wireless communication: Mobile networks are inherently unstable, with frequent oscillations and reliability issues across all available infrastructures [26, 27, 30, 45]. For example, Figure 2 presents the FCC's mobile broadband uplink throughput distribution in 10 major U.S. cities (January 2023) [13]. The average and median throughputs are 11.41 Mbps and 8.95 Mbps, respectively, with a standard deviation of 11.54 Mbps. For 90% of the time, uplink throughput remains below 25 Mbps; (2) Heterogeneous computational resources: Image LCE clients run on diverse platforms (*e.g.,* laptops, tablets, and web), leading to significant variations in computational capabilities; and (3) Varying complexities of image editing operations: Image edits differ widely in computational intensity, making it difficult to design a one-size-fits-all solution. These *motivates us to enhance the adaptability of image LCE systems to better cope with varying resource constraints.*

3 Alice Oveview

Alice is a cross-platform adaptation framework designed for low-latency image live co-editing. It offers two key features: (1) *a combined use of data-based and operation-based image edit transmission approaches* (§4, §6.1), and (2) *real-time selection of the transmission strategy* (§6.2). Figure 3 illustrates the workflow of Alice, with most components operating on the client side. In an image LCE session, user edits are encoded using Alice 's image edit codec before being transmitted to other users, who then decode and apply them to the shared image. The strategy scheduler determines the compression configuration based on resource estimates (*i.e.,* bandwidth and computation) provided by the resource monitor. The Alice server forwards valid image edits (after conflict management) to users.

4 Understanding Real User's Image Editing Pattern: A Pilot Study

We conduct a user study with three objectives: (1) understand real user image editing patterns, (2) validate the tile-based image edit transmission design outlined in §2.1, and (3) provide additional insights for image LCE system design.

4.1 Methodology

The high-level methodology of our user study is to have participants complete a series of pre-defined image editing tasks on various pre-defined images. We use a professional image editing tool to log users' operations and edits. Due to the complexity of implementing

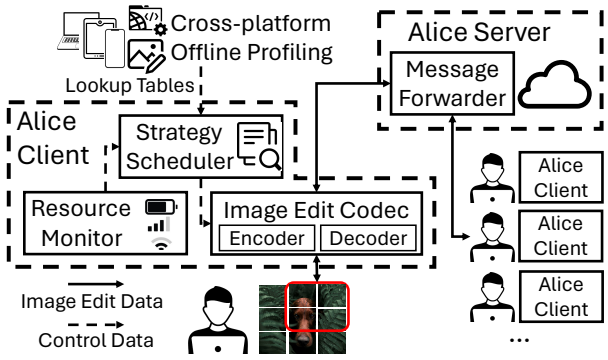


Figure 3: The system architecture of Alice.

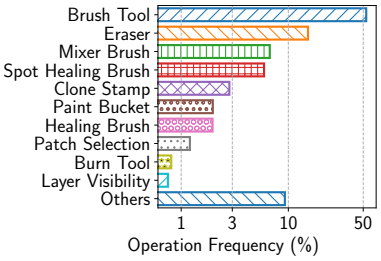


Figure 4: The operation frequency in our user study.

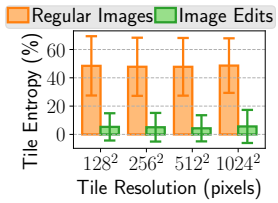


Figure 5: The tile entropy of regular image and image edits.

a professional-grade image live co-editing tool, this study focuses on analyzing the editing patterns of individual users instead. As shown in the previous section, these results remain valuable as they provide insights into the editing behavior of professional creators, which, in turn, inform the design of our Alice system. Additionally, we exclude generative AI-based image editing operations [33].

4.2 Results & Insights

We collect a total of 19,819 image edits performed by seven participants using 143 distinct image editing operations with diverse parameters. We then present our analysis results and insights.

Frequencies of Image Editing Operations. Figure 4 presents the frequencies of image editing operations, highlighting the 10 most common actions. We derive two key insights from these results. First, a significant portion of image editing operations impose minimal computational overhead on the host machine, such as Layer Visibility. This suggests that instead of relying on *data-based* transmission strategy, image LCE systems can *transmit only the operation API and parameters*, which typically require far less bandwidth than pixel data. Other users can then replicate the edit by replaying the operation locally with the received parameters – an approach we refer to as *operation-based* transmission. Second, certain image editing operations, such as Patch Selection, do not alter image pixels and therefore do not require transmission.

Resolutions of Image Edits. To analyze the area sizes affected by image edits, we define a set $\mathcal{S}$ consisting of 7 resolutions: $\mathcal{S} = \{128^2, 256^2, 384^2, 512^2, 1024^2, 2048^2, 4096^2\}$. The resolution of an image edit is determined as the smallest resolution in $\mathcal{S}$ that fully encompasses the edit. Our findings show that the two most common resolutions among collected image edits are 1024^2 and 384^2. In addition, the average height and width of these edits are 660×660 pixels. *These results validate the tile-based image edit transmission design in basic image LCE systems (see §2.1).*

Complexity of Image Edit Tiles. We analyze the complexity of collected image edit tiles and find that they are mathematically "simpler" than regular image tiles of the same resolution. The complexity of an image edit can be quantified by its entropy [37], measured as the proportion of data size reduction after lossless compression. To investigate this, we segment image edits into tiles as defined in §2.1, focusing on four resolutions: $\{128^2, 256^2, 512^2, 1024^2\}$. We randomly sample 1,000 image edit tiles at each resolution and compare their complexity to an equal number of regular image tiles, sampled from our custom dataset (discussed in §5.1). We use PNG [35] for lossless compression. As shown in Figure 5, the average entropy of image edit tiles at 128^2, 256^2, 512^2, and 1024^2 is lower than that of regular image tiles at the same resolutions by 43.13%, 42.74%, 43.46%, and 43.01%, respectively. These results confirm the simplicity of image edit tiles. Our key insight is that, *given the inherent simplicity of image edit tiles, there are ample opportunities to reduce end-to-end latency by employing less complex compression techniques or configurations. This can significantly accelerate lossless compression while maintaining the compression ratio.*

5 Lossless Compression For Image LCE

Many lossless compression techniques [18, 19, 22, 35, 36, 39] have been developed, yet none has been specifically investigated in the context of image live co-editing. We evaluate their performance using a comprehensive dataset we collected. We collect a separate dataset instead of reusing the one from §4 because we aim to investigate a broader range of image types.

5.1 Frameworks, Dataset & Methodology

Lossless Compression Frameworks. We explore three frameworks: a general-purpose lossless data compression framework, zlib [36], and two dedicated image compression frameworks, PNG [35] and JPEG XL [18]. The reference implementations used are zlib-1.3.1 [12], libpng-1.6.43 [5], and libjxl-0.10.0 [4]. These frameworks offer 10, 2, and 10 configurations, respectively, to balance compression ratio and efficiency. In total, we examine 20 configurations: 9 from zlib, 2 from PNG, and 9 from JPEG XL. We exclude one zlib configuration that performs no compression and one JPEG XL configuration due to excessively long compression times.

Image Tile Dataset. We construct a comprehensive image tile dataset by randomly sampling a subset from multiple popular public image datasets [2, 3, 6, 10, 11, 20, 21, 23, 29]. This results in 881 images with diverse resolutions, categories, and complexities, including photographic images, photorealistic images, artistic images, AI-generated images, and simple binary masks. Each image is then divided into tiles according to the definition in §2.1. We consider four tile resolutions: $\{128^2, 256^2, 512^2, 1024^2\}$. For each resolution, we get 183K+, 47K+, 12K+, and 3K+ tiles, respectively.

Methodology. We evaluate the compression ratio and compression latency (encoding + decoding) of the above frameworks with our dataset, using a single CPU thread on a MacBook Air M1 2020 [7]. We consider both native and web (WebAssembly (Wasm) [24]) applications. Internal parallelism in compression engines is disabled, if available.

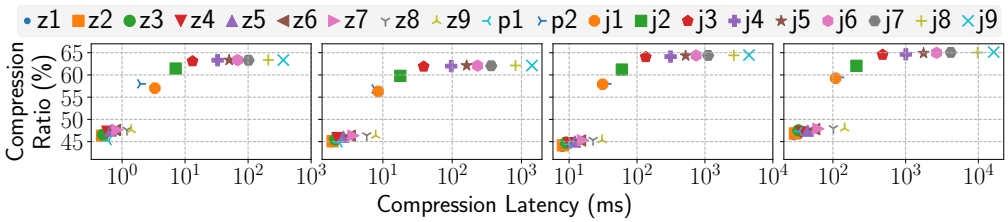


Figure 6: Average compression (encoding + decoding) ratio/latency of zlib (z1-9), PNG (p1-2), and JPEG XL (j1-9) under various image tile resolutions (left to right): 128^2, 256^2, 512^2, 1024^2.

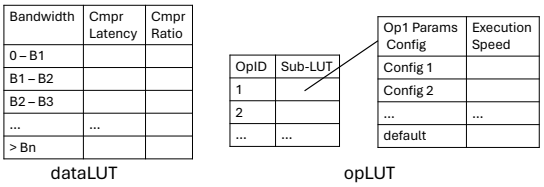


Figure 7: An example of the dataLUT and opLUT of Alice.

5.2 Compression Performance

Figure 6 presents the benchmark results across diverse tile resolutions. We derive four key takeaways: (1) zlib-{1-6} demonstrate a good trade-off between compression ratio and latency. However, from zlib-6 to zlib-9, improvements in compression ratio become marginal, while compression latency increases significantly; (2) The two PNG configurations exhibit notable differences in both compression ratio and latency; (3) JPEG XL-{1-3} provide a reasonable trade-off between compression ratio and latency. In contrast, JPGE XL-{4-9} incur a substantial increase in compression latency with only minimal gains in compression ratio; and (4) More importantly, *the significant heterogeneity among all configurations presents opportunities for image LCE systems to select compression configurations at runtime to adapt to varying resource constraints.*

6 System Design

We now present the system design of Alice, as illustrated in Figure 3.

6.1 Hybrid Transmission Strategies

Alice combines *data-based* and *operation-based* approaches for image edit transmission. In *data-based* transmission, pixel data from an image edit is sent from one user to others, with or without lossless compression. In *operation-based* transmission, Alice transmits the metadata of the operation (*e.g.,* API and parameters) from one user to others, who then replicate the edit by replaying the operation locally with the received parameters. This hybrid approach introduces a system compatibility challenge for Alice, which stems from two key issues. First, clients in a live co-editing session may have different sets of image operations due to varying application versions (*e.g.,* across operating systems) or platform resources (*e.g.,* native vs. web applications, laptop vs. tablet). Second, many image editing operations rely on specific graphic assets or presets [1], which may not be publicly available. As a result, blindly adopting the hybrid transmission strategy without precautions may lead to transmission failures due to operation incompatibility or missing graphic assets. To address this challenge, Alice enforces two policies: (1) *operation-based* transmission is restricted to common operations that are supported across all variants of the image LCE application. These operations can be negotiated at the start of an editing session between remote clients; and (2) the Alice client monitors the graphic assets used in each editing operation and switches to data-based transmission whenever private assets are involved.

6.2 Real-time Strategy Selection

Alice must solve a discrete optimization problem in real time to determine the optimal transmission strategy for image edits. This problem involves a large search space, including: (1) selecting between the *operation-based* and *data-based* approaches, and (2) choosing from all available configurations of the *data-based* approach. There are three additional challenges: (1) measuring the exact end-to-end latency of each configuration through actual execution is impractical, as it would require redundant transmissions of the same image edit; (2) the selection algorithm must be fast to prevent additional latency overhead; and (3) each LCE client must be aware of available resources on other clients to optimally solve the problem.

Inspired by FastMPC [46], Alice overcomes these challenges by efficiently selecting the optimal compression strategy at runtime using an LUT-based approach. Each Alice client maintains two LUTs: dataLUT for the *data-based* approach and opLUT for the *operation-based* approach. As shown in Figure 7, dataLUT is indexed by network bandwidth and provides the predicted optimal data-based compression configuration along with its associated compression latency and ratio. opLUT is a two-level LUT: the first level is indexed by the operation ID, which accesses a sub-LUT for each image operation. The sub-LUT is indexed by specific parameters to predict execution speed based on those parameters. At runtime, for each image edit, Alice: (1) predicts *data-based* transmission latency by querying compression latency and ratio from the dataLUT, using the latest bandwidth estimate, (2) forecasts *operation-based* transmission latency by combining operation execution speed from the opLUT, size of the operation parameters, and estimated bandwidth, and (3) selects the strategy with the lowest predicted latency for the image edit. Each {platform, tile resolution} combination has its own dataLUT and opLUT, which are constructed offline through extensive performance profiling and enumeration, making this a one-time setup. Next, we detail the LUT construction process.

Constructing the dataLUTs. Alice builds a dataLUT in two steps. First, it profiles the compression ratio and latency of all considered compression techniques across various configurations. This profiling follows the methodology in §5, using a comprehensive image tile dataset. Next, Alice enumerates possible bandwidth values and selects the compression configuration that minimizes latency for each bandwidth. The latency of each configuration is estimated using back-of-the-envelope calculations based on the average compression ratio and latency obtained in the first step. To optimize the dataLUTs, Alice applies several trimming strategies: (1) Initially,

it profiles all configurations on a dataset subset before constructing the full dataLUT. It discards configurations with high compression latency and limited compression ratio improvement, such as zlib-{7-9} and JPEG XL-{7-9}, as shown in Figure 6; (2) Configurations that consistently perform worse than others are removed; and (3) Instead of enumerating all possible bandwidth values, Alice identifies a bandwidth range where each configuration performs optimally. Higher bandwidth requires less compression effort for image edits, making this range easily to be identified.

Building the opLUTs. The opLUTs are constructed through extensive offline measurements of various image editing operations, with two key differences from dataLUTs. First, Alice does not enumerate bandwidth values for opLUTs. Instead, it predicts *operation-based* transmission latency at runtime. Second, image editing operations can have infinite possible parameter values, unlike the finite configurations in data-based transmission. Profiling every parameter set is impractical. To address this, Alice profiles a subset of frequently used parameters, which may include the top-k most common parameters identified from large-scale user studies. This subset is dynamically updated as Alice collects more user operation data. In addition, each operation's LUT includes a *default* entry, which provides the average execution speed across all profiled parameter sets. If a query does not match a specific entry, Alice returns this default value. We empirically set k to 20 to ensure fast table lookup, resulting in an opLUT size of ~2MB (assuming 1,000 operations and each entry takes 100 Bytes).

7 Implementation & Evaluation

Our **implementation** consists of: (1) an image LCE system following the architecture in Figure 1, comprising 2,182 lines of code (LoC), and (2) the Alice framework (Figure 3) with 2,777 LoC, integrated into the LCE system. Both components are implemented in C/C++. We develop a simple message-oriented protocol over TCP for communication between the server and clients. For *data-based* transmission, Alice applies zlib-1.3.1 [12], libpng-1.6.43 [5], and libjxl-0.10.0 [4] with configurations 1-6, 1-2, and 1-3, respectively.

7.1 Experimental Setup

Controlled Experiments. We conduct trace-driven evaluations, where each image LCE client is assigned an image editing trace, an uplink throughput trace, and a downlink throughput trace.

Image Editing Traces. Each data point in an editing trace includes operation metadata (API and parameters), capturing time, and affected pixels and their values. To generate image editing traces, we first create an operation set with 20 image processing operations from OpenCV-4.9.0 [8]. We consider 4 trace durations: 60, 120, 180, and 240 seconds, and synthesize 20 traces per duration: For a t-second trace, we first determine the total number of operations n ($n < t$), randomly sampled from our operation set. The capturing time for the n operations are uniformly sampled from $[0, t]$. Then, for each operation, we randomly select an area from an image in our dataset, apply the operation to that area, and record the modified pixels along with their values. The operation parameters are randomly generated.

Network Traces. We sample network traces from the FCC mobile broadband dataset (January 2023) [13] and replay them using Mahimahi [32]. We use 20 uplink traces and 20 downlink traces. The average uplink throughput ranges from 10.21 to 12.34 Mbps, with a standard deviation of 10.25 to 13.02 Mbps. The average downlink throughput ranges from 64.47 to 73.51 Mbps, with a standard deviation of 59.38 to 76.15 Mbps.

Baseline	Transmission	Compression
op	*operation-based*	N/A
data	*data-based*	Dynamic configuration
raw	*data-based*	No compression
zlib3	*data-based*	zlib-3 in §5.1
png1	*data-based*	PNG-1 in §5.1
jxl1	*data-based*	JPEG XL-1 in §5.1

Table 1: Comparison baselines in our experiments.

Devices, Dataset, and Other Configurations. We use an Ubuntu 18.04 desktop (64-GB memory, Intel Core i9-10900X CPU @ 3.70GHz) as the LCE server. LCE clients run on two difference platforms, evenly distributed: an Ubuntu 20.04 desktop (32-GB memory, Intel Core i7-9700K CPU @ 3.60GHz) and a MacBook Air 2020 laptop. We randomly select 100 images from our dataset (§5.1) for evaluation and consider diverse *tile resolutions* (512^2 and 1024^2) and *client scales* (2, 4, 6, 8, and 10 clients).

Baselines. We consider 6 baselines, summarized in Table 1: (1) *op* applies only the *operation-based* approach; (2) *data* applies only the *data-based* approach with online configuration selection; and (3) 4 *data-based* baselines with fixed configuration: *raw* applies no compression, *zlib3* uses zlib configuration 3, *png1* uses PNG configuration 1, and *jxl1* uses JPEG XL configuration 1.

Metrics. We evaluate Alice in terms of the per-tile end-to-end latency and its overhead. Per-tile latency is defined as the elapsed time between when an image edit on a tile is captured and when the tile is displayed on the other client's device. We focus on per-tile latency because, in our prototype, tiles are compressed and transmitted sequentially, making per-tile latency a reasonable approximation of per-edit latency. All results are reported across all images, traces, network traces, and clients.

7.2 End-to-end Performance of Alice

Alice **vs. Baselines.** We first compare Alice with the baselines in Table 1, considering 2 LCE clients with a tile resolution of 1024^2. Figure 8 shows our result. We have three takeaways. First, compared to *raw*, *zlib3*, *png1*, and *jxl1*, *data* reduces the average (median) per-tile latency by 90.44% (49.91%), 78.16% (9.62%), 78.30% (11.27%), and 75.31% (20.42%), respectively, thanks to the dynamic compression configuration selection at runtime, which adapts to the varying bandwidth better, compared to using a fixed configuration. Second, compared to *data* and *op*, Alice further reduces the average (median) latency by 62.03% (28.34%) and 66.72% (79.09%), respectively. This confirms the effectiveness of jointly using *data-based* and *operation-based* approaches. Third, the average (median) per-tile latency of Alice is 160.18ms (85.01ms), which is a 95.48% (70.18%), 91.85% (60.55%), 91.49% (62.21%), and 93.17% (71.65%) reduction compared to *raw*, *zlib3*, *png1*, and *jxl1*, respectively.

Various Tile Resolution. We repeat the evaluation with a tile resolution of 512^2. As shown in Figure 9: (1) compared to *raw*, *zlib3*, *png1*, and *jxl1*, *data* reduces the average (median) per-tile latency by 90.44% (49.91%), 78.16% (9.62%), 78.30% (11.27%) and 75.31% (20.42%), respectively; (2) compared to *data* or *op*, Alice further reduces the

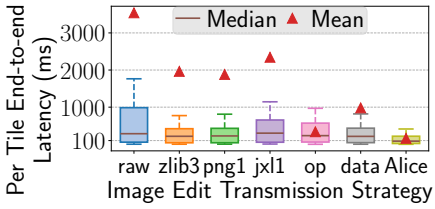


Figure 8: Per-tile latency of different strategies. Tile resolution is 1024^2. # of clients is 2.

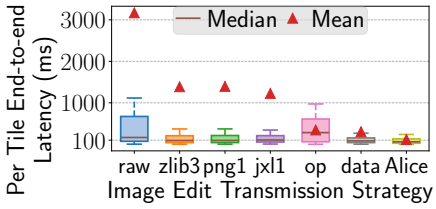


Figure 9: Per-tile latency of different strategies. Tile resolution is 512^2. # of clients is 2.

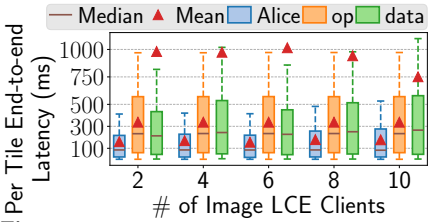


Figure 10: Per-tile latency with various # of clients. Tile resolution is 1024^2.

average (median) per-tile latency by 62.03% (28.34%) and 66.72% (79.02%), respectively; and (3) the average (median) per-tile latency of Alice is 115.06ms (58.87ms). This result confirms Alice's effectiveness under various tile resolutions.

Diverse Client Scales. We vary the number of LCE clients from 2 to 10, and compare Alice with *op* and *data* in Table 1. Figure 10 shows that when the number of clients is 2, 4, 6, 8, and 10, Alice consistently outperforms the baselines: (1) the average (median) per-tile latency of Alice is 160.18ms (85.01ms), 169.62ms (85.02ms), 155.36ms (84.52ms), 178.99ms (85.00ms), and 177.35ms (82.57ms), respectively; (2) Compared to *data*, Alice reduces the average (median) latency by 83.68% (60.13%), 82.60% (65.07%), 84.70% (62.79%), 81.01% (66.08%), and 76.40% (68.88%), respectively; (3) Compared to *op*, Alice reduces the average (median) latency by 52.59% (63.49%), 49.80% (63.71%), 54.00% (63.90%), 47.04% (63.72%), and 47.50% (64.64%), respectively. This result confirms the scalability of Alice.

7.3 Micro-benchmarks

LUT-based vs. ML-based Selection. We compare the LUT-based strategy selection with a machine learning-based approach (Alice-ML). Specifically, we formulate strategy selection as a classification problem, where a machine learning model predicts the optimal strategy based on: device type ID, tile resolution, operation ID, and bandwidth estimation. Using an offline-collected dataset, we investigate 4 lightweight ML models: *Logistic Regression* (LR), *Random Forest* (RF), *Gradient Boosting Decision Tree* (GBDT), and *Multi-layer Perceptron* (MLP). We implement these models using Python's *scikit-learn* package with default parameters. Table 2 shows their prediction accuracy through 10-fold cross-validation, where GBDT achieves the best performance. We then integrate the pre-trained GDBT model into Alice-ML and compare its performance with Alice. We set the tile resolution to 1024^2 and the number of clients to 2. Figure 11 shows that Alice outperforms Alice-ML, reducing average (median) latency by 81.83% (56.91%). The likely reason for Alice's superior performance is the limited scale of the training dataset and the lack of fine-tuning in the ML model. Nevertheless, this result demonstrates the feasibility of ML-based solutions as an alternative approach for dynamic configuration selection, which we plan to further explore in future work.

Alice **Overhead.** We confirm that Alice incurs negligible overhead (<1ms) for transmission strategy selection on our test devices.

Model	Accuracy
LR	51.64
RF	55.91
GBDT	**61.76**
MLP	52.13

Table 2: Prediction accuracy of explored ML models.

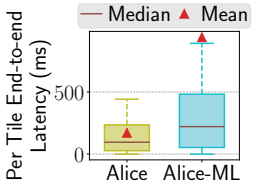


Figure 11: Alice vs. Alice-ML. The tile resolution is set to 1024^2.

8 Related Work

Reducing Network-side Delay. Extensive research has focused on reducing network-side latency for various applications [31, 41, 43]. Zhuge [31] reduces tail latency in wireless real-time communication (RTC) applications by proactively predicting network delay for each RTC packet and adjusting the packet sending rate accordingly. TwinStar [43] achieves low-latency video delivery by leveraging multiple network paths simultaneously, mitigating network jitter over a single path. Tan et al. [41] propose a data-driven LTE latency reduction framework for latency-sensitive mobile applications by analyzing operational LTE traces.

Adaptive Streaming. Adaptive bitrate (ABR) streaming is widely adopted in modern multimedia systems [25, 34, 40, 42, 47, 48], from 2D videos to immersive content. The core concept is to encode multimedia content at multiple quality levels and stream the highest possible quality that matches the current network bandwidth [40, 47], ensuring timely delivery before the playback deadline. Some immersive video streaming systems [25, 34] adopt visibility-adaptive streaming, transmitting only content visible to the viewer. Machine-oriented systems, such as connected autonomous vehicles [49] and video analyst systems [28], dynamically adjust encoding parameters to adapt to fluctuating network conditions.

Optimizing Lossless Compression. A wide range of lossless compression techniques [18, 19, 22, 35, 36, 39] have been developed over the past decades. In addition, research has focused on accelerating existing frameworks [38, 44]. Wu et al. [44] propose an entropy-guided, content-aware pixel prioritization strategy to enhance the progressive compression performance of FLIF [39]. Shen et al. [38] optimize FLIF by leveraging GPU parallelism to accelerate compression. These studies are orthogonal to our work.

9 Concluding Remarks

This paper focuses on achieving low-latency image live co-editing. We demonstrate that by jointly utilizing *data-based* and *operation-based* strategies and integrating diverse lossless compression techniques and configurations, an image LCE system can achieve up to 95% latency reduction compared to baselines. We believe that our high-level design concept can also benefit human-machine co-editing applications in the Generative AI era.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work was supported in part by NSF Award CNS-2402991, CNS-2409267, and CNS-2409269.

References

[1] Create or delete presets. https://helpx.adobe.com/photoshop-elements/prereleas e/create-save-edit-delete-presets.html.

[2] Ieee mmsp 2020 challenge: Learning-based image coding challenge test images. https://jpegai.github.io/test_images/.

[3] Kodak lossless true color image suite. https://r0k.us/graphics/kodak/.

[4] libjxl v0.10.0. https://github.com/libjxl/libjxl/archive/refs/tags/v0.10.0.zip.

[5] libpng v1.6.43. https://github.com/pnggroup/libpng/releases/tag/v1.6.43.

[6] Lossless compression software for camera raw images. https://imagecompressi on.info/.

[7] Macbook air (m1, 2020) - technical specifications. https://support.apple.com/en-us/111883.

[8] Opencv 4.9.0. https://github.com/opencv/opencv/releases/tag/4.9.0.

[9] Photoshop Unlocks Creative Collaboration with Live Co-Editing. https://blog.adobe.com/en/publish/2025/01/14/photoshop-unlocks-creative-collaboration-with-live-co-editing-join-private-beta.

[10] Unsplash dataset. unsplash.com/data.

[11] Wikiart dataset. https://huggingface.co/datasets/huggan/wikiart/.

[12] zlib 1.3.1 release. https://github.com/madler/zlib/archive/refs/tags/v1.3.1.zip.

[13] Measuring broadband america mobile data. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-broadband-america-mobile-data, 2023.

[14] Facet: Photo editor. https://facet.ai/photo-editor, 2024.

[15] Figma: The collaborative interface design tool. https://www.figma.com/, 2024.

[16] Google docs: Online document editor. https://www.google.com/docs/about/, 2024.

[17] Overleaf, online latex editor. https://www.overleaf.com/, 2024.

[18] J. Alakuijala, R. Van Asseldonk, S. Boukortt, M. Bruse, I.-M. Comșa, M. Firsching, T. Fischbacher, E. Kliuchnikov, S. Gomez, R. Obryk, et al. Jpeg xl next-generation image compression architecture and coding tools. In *Applications of digital image processing XLII*, volume 11137, pages 112–124. SPIE, 2019.

[19] U. Albalawi, S. P. Mohanty, and E. Kougianos. A hardware architecture for better portable graphics (bpg) compression encoder. In *2015 IEEE international Symposium on Nanoelectronic and information systems*, pages 291–296. IEEE, 2015.

[20] N. Asuni and A. Giachetti. Testimages: A large data archive for display and algorithm testing. *Journal of Graphics Tools*, 17(4):113–125, 2013.

[21] N. Asuni and A. Giachetti. Testimages: a large-scale archive for testing visual devices and basic image processing algorithms. In *STAG*, pages 63–70, 2014.

[22] M. Calore. Meet webp, google's new image format, 2010.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[24] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien. Bringing the web up to speed with webassembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 185–200, 2017.

[25] B. Han, Y. Liu, and F. Qian. Vivo: Visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th annual international conference on mobile computing and networking*, pages 1–13, 2020.

[26] A. Hassan, A. Narayanan, A. Zhang, W. Ye, R. Zhu, S. Jin, J. Carpenter, Z. M. Mao, F. Qian, and Z.-L. Zhang. Vivisecting mobility management in 5g cellular networks. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 86–100, 2022.

[27] B. Hu, X. Zhang, Q. Zhang, N. Varyani, Z. M. Mao, F. Qian, and Z.-L. Zhang. Leo satellite vs. cellular networks: Exploring the potential for synergistic integration. In *Companion of the 19th International Conference on emerging Networking EXperiments and Technologies*, pages 45–51, 2023.

[28] S. Kim, K. Bin, D. Yang, S. Ha, S. Chong, and K. Lee. Entro: Tackling the encoding and networking trade-off in offloaded video analytics. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9115–9123, 2023.

[29] M. Ku, T. Li, K. Zhang, Y. Lu, X. Fu, W. Zhuang, and W. Chen. Imagenhub: Standardizing the evaluation of conditional image generation models. *arXiv preprint arXiv:2310.01596*, 2023.

[30] Y. Li, H. Lin, Z. Li, Y. Liu, F. Qian, L. Gong, X. Xin, and T. Xu. A nationwide study on cellular reliability: Measurement, analysis, and enhancements. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 597–609, 2021.

[31] Z. Meng, Y. Guo, C. Sun, B. Wang, J. Sherry, H. H. Liu, and M. Xu. Achieving consistent low latency for wireless real-time communications with the shortest control loop. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 193–206, 2022.

[32] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. Mahimahi: accurate {Record-and-Replay} for {HTTP}. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, 2015.

[33] D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

[34] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 99–114, 2018.

[35] G. Roelofs. *PNG: the definitive guide*. O'Reilly & Associates, Inc., 1999.

[36] G. Roelofs. zlib: A massively spiffy yet delicately unobtrusive compression library. *http://www. zlib. net/*, 2017.

[37] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[38] Y. Shen, G. Wu, V. Swaminathan, H. Wang, S. Petrangeli, and T. Yu. Gpu-accelerated lossless image compression with massive parallelization. In *2023 IEEE International Symposium on Multimedia (ISM)*, pages 321–324. IEEE, 2023.

[39] J. Sneyers and P. Wuille. Flif: Free lossless image format based on maniac compression. In *2016 IEEE international conference on image processing (ICIP)*, pages 66–70. IEEE, 2016.

[40] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE multimedia*, 18(4):62–67, 2011.

[41] Z. Tan, J. Zhao, Y. Li, Y. Xu, and S. Lu. {Device-Based}{LTE} latency reduction at the application layer. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 471–486, 2021.

[42] C. Wang, A. Zhang, Y. Yang, L. Qiu, Y. Yang, X. Jiang, F. Qian, and S. Banerjee. Volut: Efficient volumetric streaming enhanced by lut-based super-resolution. *arXiv preprint arXiv:2502.12151*, 2025.

[43] H. Wang, Z. Yu, R. Zhang, S. Tao, H. Yu, and S. Shi. Twinstar: A practical multi-path transmission framework for ultra-low latency video delivery. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9234–9242, 2023.

[44] J. Wu, H. Wang, T. Yu, G. Wu, S. Petrangeli, H. Zhao, S. Kim, and V. Swaminathan. Content-aware progressive image compression and syncing. In *2023 IEEE International Symposium on Multimedia (ISM)*, pages 221–224. IEEE, 2023.

[45] R. Xing, M. Xu, A. Zhou, Q. Li, Y. Zhang, F. Qian, and S. Wang. Deciphering the enigma of satellite computing with cots devices: Measurement and analysis. *arXiv preprint arXiv:2401.03435*, 2024.

[46] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, 2015.

[47] A. Zhang, C. Wang, B. Han, and F. Qian. {YuZu}:{Neural-Enhanced} volumetric video streaming. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 137–154, 2022.

[48] A. Zhang, C. Wang, Y. Hu, A. Hassan, Z. Zhang, B. Han, F. Qian, and S. Xu. Habitus: boosting mobile immersive content delivery through full-body pose tracking and multipath networking. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1677–1695, 2024.

[49] X. Zhang, A. Zhang, J. Sun, X. Zhu, Y. E. Guo, F. Qian, and Z. M. Mao. Emp: Edge-assisted multi-vehicle perception. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 545–558, 2021.