Palantír: Towards Efficient Super Resolution for Ultra-high-definition Live Streaming

Xinqi Jin*
School of Software & BNRist, Tsinghua University
jinxq21@mails.tsinghua.edu.cn

Zhui Zhu*
Department of Automation & BNRist, Tsinghua University
z-zhu22@mails.tsinghua.edu.cn

Xikai Sun
Department of Automation & BNRist, Tsinghua University
sxk23@mails.tsinghua.edu.cn

Fan Dang
School of Software Engineering, Beijing Jiaotong University
dangfan@bjtu.edu.cn

Jiangchuan Liu
School of Computing Science, Simon Fraser University
jcliu@sfu.ca

Jingao Xu
Computer Science Department, Carnegie Mellon University
jingaox@andrew.cmu.edu

Kebin Liu
Global Innovation Exchange, Tsinghua University
kebinliu2021@tsinghua.edu.cn

Xinlei Chen
Shenzhen International Graduate School, Tsinghua University
chen.xinlei@sz.tsinghua.edu.cn

Yunhao Liu†
Global Innovation Exchange & Department of Automation & BNRist, Tsinghua University
yunhao@tsinghua.edu.cn

Abstract

Neural enhancement through super-resolution (SR) deep neural networks (DNNs) opens up new possibilities for ultra-high-definition (UHD) live streaming. Yet, the heavy SR DNN inference overhead leads to severe deployment challenges. To reduce the overhead, existing systems propose to apply DNN-based SR only on carefully selected anchor frames while upscaling non-anchor frames via the lightweight reusing-based SR approach. However, frame-level scheduling is coarse-grained and fails to deliver optimal efficiency. In this work, we propose Palantír, the first neural-enhanced UHD live streaming system with fine-grained patch-level scheduling.

At the core of Palantír is its SR video quality estimation strategy which guides the low-delay selection of the most beneficial anchor patches. Although existing systems propose estimation strategies for anchor frame selection, these strategies heavily rely on empirical insights that cannot be transferred to our context, making fine-grained scheduling a challenging problem that requires a fundamentally new solution. Facing the challenge, we follow the first-principles approach and derive a directed acyclic graph (DAG) model to address the problem. The model can also be generalized to various settings due to its first-principles nature. Compared to the state-of-the-art real-time frame-level scheduling strategy for live streaming, Palantír reduces the anchor size by 80.1% at most and 38.4% on average without compromising the quality gain. Furthermore, Palantír incurs a scheduling latency accounting for only 0.6-3.9% of the end-to-end latency requirement for UHD live streaming.

CCS Concepts

• **Information systems** → **Multimedia streaming**; • **Computing methodologies** → *Computer vision.*

Keywords

Video streaming, super-resolution, video codec

ACM Reference Format:
Xinqi Jin, Zhui Zhu, Xikai Sun, Fan Dang, Jiangchuan Liu, Jingao Xu, Kebin Liu, Xinlei Chen, and Yunhao Liu. 2025. Palantír: Towards Efficient Super Resolution for Ultra-high-definition Live Streaming. In *ACM Multimedia Systems Conference 2025 (MMSys '25), March 31-April 4, 2025, Stellenbosch, South Africa.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3712676.3714434

1 Introduction

UHD videos such as 4K and 8K videos are expected to form a huge market worth more than $1 trillion in the following few years [17]. More and more UHD live-streaming applications [14, 16, 24, 25] are deployed to revolutionize many aspects of our society. However, the bitrates of 4K videos (as large as 45Mbps [20]) can be significantly larger than the worldwide average uplink bandwidth of mobile broadband networks (about 11Mbps [36]), which poses great deployment challenges. Although the latest codecs such as AV1 and VVC can provide a lower bitrate, they can be rather computation-intensive and typically require specific hardware encoders to achieve real-time UHD encoding [38], which are still unavailable on many commercial devices [23]. Using fixed broadband networks for the uplink is an alternative solution, but it inevitably affects mobility and prohibits many mobile applications [10–13, 37].

Recently, neural enhancement has been proposed [8, 46, 48] and deployed [15, 27, 34, 50] to improve video streaming. It can potentially boost the broad deployment of UHD live streaming by

*Co-primary authors.
†Corresponding author.
Project repository: **https://palantir-sr.github.io**

MMSys '25, Stellenbosch, South Africa

ACM ISBN 979-8-4007-1467-2/2025/03
https://doi.org/10.1145/3712676.3714434

allowing the streaming source to stream only a low-bitrate low-resolution (LR) video over the bandwidth-limited uplink and using a super-resolution (SR) deep neural network (DNN) to upscale the LR stream to its SR counterpart later. However, as detailed in Sec. 2.1, SR DNN inference incurs heavy computation overhead and deployment challenges. Therefore, many research efforts have been aimed at optimizing the overhead. NEMO [46] and NeuroScaler [48] achieve this by categorizing frames into anchor frames and non-anchor frames: only anchor frames undergo computation-intensive DNN-based SR, while non-anchor frames are reconstructed via reusing the SR results of reference frames. Scheduling has been a central aspect of designing such systems - the most beneficial anchor frames must be carefully selected to deliver a large quality gain with a low inference overhead. Nevertheless, these systems are inherently sub-optimal due to their coarse scheduling granularity. A beneficial anchor frame may contain some existing objects that can be well reconstructed by reusing-based SR, and vice versa.

In this paper, we propose Palantír, **the first patch-level neural-enhanced UHD live streaming system** that selects the appropriate type — anchor or non-anchor — for each patch (part of a single frame). Palantír aims to meet two design goals. The first is to accurately pinpoint the most beneficial anchor patches so that a smaller overhead can be achieved without sacrificing the quality gains. The second is to minimize the scheduling latency to better support many latency-sensitive live streaming applications [1, 16, 28, 41, 44].

Limitations of existing methods (Sec. 3). A natural idea of fulfilling fine-grained scheduling is to adapt existing frame-level solutions. However, as shown in Sec. 3, **existing frame-level solutions all lack a deep understanding of how DNN-based and reusing-based SR affect the SR quality.** Their success in frame-level scheduling heavily relies on empirical insights that either require non-realtime operations or only make sense in the limited context of frame-level scheduling. Therefore, an entirely different scheduling method is required to meet our two design goals.

Palantír: a paradigm shift. (Sec. 5). Instead of using trial and error to seek another insight suitable for the fine-grained context, which may again face the problem of limited generalizability in other contexts, we **adopt a first-principles approach**. A first-principles approach involves breaking down complex problems into their most fundamental elements and building solutions from the ground up, rather than relying on empirical insights. In our case, we systematically analyze the SR process of every basic coding block to understand how coding types and anchor configurations affect the SR error (which is inversely related to the quality gains of SR). We then naturally conclude from our pioneering analysis that the process can be simulated accurately and quickly through a DAG structure at the patch-level granularity. This analytical model allows us to efficiently and accurately estimate the SR error for any given anchor patch set and lays a rigorous foundation for the design of Palantír towards the two aforementioned goals. As a final note, **the first-principles DAG structure essentially reflects the inter-frame coding strategy, which is at the core of all mainstream codecs, and thus shows unprecedented generalizability to various codecs, coding settings, and video types.**

In addition to the paradigm shift, Palantír further incorporates two novel techniques to tackle some specific challenges and ultimately meet the two design goals (**Sec. 6**). *First*, strictly following our analytical framework to construct the DAG requires information from the high-resolution (HR) video. We expect the scheduling algorithm of Palantír to be executed at the media server for easy deployment, yet the media server often does not have access to the HR video. We overcome this challenge by approximating missing HR information with the available LR data. *Second*, we find that a naive implementation of DAG-based estimation still fails the second goal. Alternatively, we utilize a universal characteristic of all mainstream codecs to introduce two parallelism mechanisms in the DAG-based computation process, which reduce the latency by hundreds of times without changing the anchor selection results.

Results. We conduct extensive evaluations to demonstrate that Palantír meets the two design goals. When compared to the naive method of applying DNN-based SR on all the frames, Palantír can **reduce the SR DNN inference overhead by 20 times (or 60 times) while preserving 54.0-82.6% (or 32.8-64.0%) of the quality gain**. When compared to the state-of-the-art real-time frame-level scheduling strategy, Palantír can **reduce the SR DNN inference overhead by 80.1% at most (and 38.4% on average) without sacrificing the video quality.** Palantír incurs a negligible scheduling latency accounting for only 0.6-3.9% of the end-to-end latency requirement.

The rest of the paper is structured as follows. Sec. 2 reviews preliminary knowledge. We detail the core limitations of existing methods in Sec. 3. A high-level overview of Palantír is outlined in Sec. 4. We introduce our first-principles approach in Sec. 5. The two practical techniques used to build Palantír are presented in Sec. 6. We evaluate Palantír in Sec. 7. The paper is concluded in Sec. 8. Discussions on future work, related work, and some technical details can be found in the Appendix.

2 Background

2.1 Primer on SR Streaming

Currently, there are two common deployment models for SR enhancement. One is to execute the SR DNN inference on the mobile streaming clients [15, 27, 34, 50], and the other is to execute the SR DNN on a cloud server to benefit many audiences for the same video [48]. Yet, SR DNNs are of high computation complexity and lead to severe deployment challenges in both deployment models. In the first model, the battery life of mobile clients can be easily drained. Consequently, Microsoft Edge VSR disables its SR feature when the device is not being charged [34]. As for the second model, the heavy inference overhead appears in the form of the high monetary cost of using cloud servers (estimated to be at least $1.690 per hour per 4K stream [48]). Lowering the overhead is essential to a broader application of SR enhancement.

2.2 Reusing-based SR

To enable low-cost SR enhancement, researchers have built the NEMO [46] system, where an SR-enhanced decoder is adopted to reduce the cost by using video temporal redundancy. In the SR decoder, video frames are categorically divided: anchor frames are upscaled via DNN-based SR, while non-anchor frames are quickly upscaled via reusing-based SR. To appreciate the intricacies of reusing, a basic understanding of video codecs is essential. Video coding predominantly encodes a block through inter-coding; it

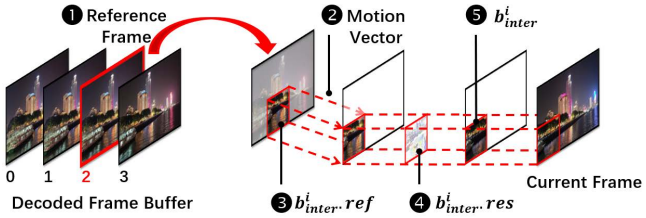


Figure 1: Video decoding pipeline.

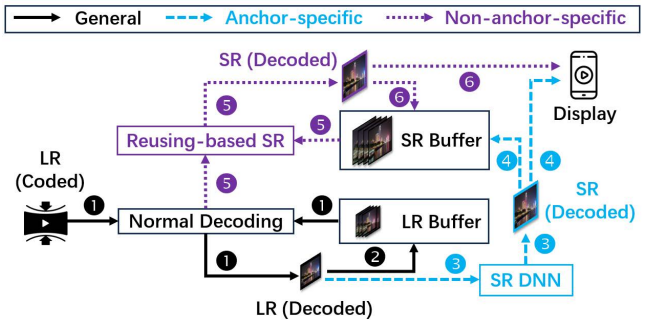


Figure 2: SR-integrated decoder overview.

identifies a similar reference block from a prior frame and only stores the subtle residual between the two blocks. A reference index is retained in the coded video to identify the frame containing the reference block or the reference frame, while a motion vector is stored to represent the potential spatial offset between the current and reference blocks. To decode an inter-coded block b^i_{inter} (*i.e.,* ❺ in Fig. 1), the decoder first parses the reference index to determine its reference frame (❶) and then parses the motion vector (❷) to determine its reference block $b^i_{inter}.ref$ (❸). The residual $b^i_{inter}.res$ (❹) is added to $b^i_{inter}.ref$ to obtain b^i_{inter}, *i.e.,*

$$b^i_{inter} = b^i_{inter}.ref + b^i_{inter}.res. \tag{1}$$

An alternative to inter-coding is intra-coding, which, while similar, encodes an intra-coded block b^i_{intra} by storing the intra-frame residual. We refer the readers to the technical specifications [21] for further details. Note that patches are not the same as encoding blocks in this paper: patches are scheduling units while blocks are encoding units.

The SR decoder in NEMO [46] is developed based on the open-source Google libvpx VP9 decoder [4]. As shown in Fig. 2, the SR decoder first decodes a frame into its LR version by referring to decoded frames in the LR buffer (❶) and can insert it into the LR buffer for future frames (❷), just as a standard decoder. Along with the LR video, a cache profile is also downloaded, each bit of which indicates whether a frame is an anchor or non-anchor. If the current frame is an anchor, the LR version will be fed into the SR DNN to obtain the SR version (❸), which may be inserted into the SR buffer for future reuse (❹). Otherwise, the SR version is obtained via reusing-based SR (❺). The reusing-based SR uses a process similar to that in Fig. 1 to decode every inter-coded block b^i_{inter} to its SR version, except that the motion vector is scaled (*e.g.,* by 4 times, when the LR and SR frames are 240p and 960p, respectively), the residual is upscaled by bilinear interpolation to match the resolution of the SR block ($b^i_{inter}.SR$), and the same reference index is used to fetch cached frames from the SR buffer rather than the LR buffer. We can formulate the process as

$$b^i_{inter}.SR = b^i_{inter}.ref.SR + interp(b^i_{inter}.res, scale), \tag{2}$$

where *scale* is the SR ratio and *interp* is the bilinear interpolation operation. We also rewrite Eq. (1) as

$$b^i_{inter}.LR = b^i_{inter}.ref.LR + b^i_{inter}.res \tag{3}$$

to distinguish between the LR and SR versions of the same block in the SR decoder. As for any intra-coded block b^i_{intra} in the non-anchor frame, it is upscaled by applying bilinear interpolation on its decoded LR version, *i.e.,*

$$b^i_{intra}.SR = interp(b^i_{intra}.LR, scale). \tag{4}$$

After upscaling every block to their SR versions, the SR version of the non-anchor frame may be inserted into the SR buffer for future reuse (❻). More details are available in [46]. Based on the open-source SR decoder in NEMO, we develop a fine-grained SR decoder, where a larger cache profile indicates the type (*i.e.,* anchor or non-anchor) of every patch, and, based on their types, patches are upscaled via either DNN-based or reusing-based SR.

3 Limitations of existing solutions

As shown in the equation (2) and (4), reusing-based SR involves only lightweight operations such as bilinear interpolation of residuals rather than heavy inference of DNNs. Despite its advantage in computation costs, bilinear interpolation may not reconstruct missing details in the LR video well, so reusing-based SR may lead to suboptimal SR qualities when compared to DNN-based SR. To trade off the SR quality and the computation overhead, existing systems like NEMO and NeuroScaler carefully select the most beneficial anchor frames. To limit the scheduling complexity, both NEMO and NeuroScaler perform independent scheduling for every LR video segment whose time duration equals the pre-defined scheduling interval. Furthermore, NEMO and NeuroScaler approximate the video qualities with estimation values to avoid the heavy measurement of video qualities and accelerate scheduling.

Palantír also bases its scheduling on quality estimations and performs independent scheduling for every LR video segment. And the design goals of Palantír are two-fold: (1) selecting the most beneficial anchor patches to further improve the quality gain without increasing the total size of all anchors; and (2) low-latency selection to support UHD live streaming. A natural way to design Palantír is to adapt existing quality estimation techniques in NEMO and NeuroScaler to the patch-level granularity. However, as revealed later in this section, **existing SR quality estimation strategies lack a thorough understanding of how DNN-based and reusing-based SR affect the quality of the resulting SR video. Alternatively, they both heavily rely on empirical insights that either incur time-consuming operations or are only valid at the frame-level granularity.** Consequently, we can't simply extend existing solutions to meet the two design goals.

Observation #1: The insight in NEMO inevitably requires a

heavy initial measurement phase and thus violates the second design goal.

The core insight in NEMO is that the quality gain of a frame is mostly determined by the most relevant anchor frame, making it reasonable to estimate the quality under a given anchor frame set by combining the actual quality measurements under relevant anchor frame sets. Specifically, NEMO first measures $FQ\left(i\middle|f\right|)$, the quality of every frame i under every single anchor frame set $|f|$. Then, NEMO estimates the quality under any anchor frame set AF as $FQ\left(i\middle|AF\right) \approx max_{f\in AF} FQ\left(i\middle|f\right|)$. Capitalized on the heuristics, NEMO requires a heavy measurement phase in nature and incurs a dramatically high latency not suitable for live streaming. The scheme can be easily extended to the patch-level granularity by firstly conducting measurements for all single anchor patch sets, but the latency of initial measurements can be further increased.

Observation #2: The linked list model in NeuroScaler is highly inaccurate and the type-based prioritization contributes the most to the success of NeuroScaler. Yet, the type-based prioritization in NeuroScaler is only valid at the frame-level granularity, while its patch-level counterpart is invalid and cannot lead us toward the first design goal.

The method in NeuroScaler [48] can support real-time scheduling with its simplified modeling of the problem. It models the super-resolution error propagation process as a linked list, where each node corresponds to a frame and each pair of temporally consecutive frames is linked. The SR error of an anchor frame is assumed to be 0, while the error of a non-anchor frame equals that of its preceding frame node plus the residual between the two frames. The quality is estimated as the inverse of the sum of the errors over all frame nodes. In addition to the linked list model, NeuroScaler further incorporates the insight that some types of frames (*i.e.,* keyframes and alternative reference frames in the VP9 codec [21]) may have a high degree of reference (*i.e.,* be directly referred to by many subsequent frames) and thus lead to a large quality gain when determined as anchors. Therefore, NeuroScaler proposes to prioritize these types of frames as anchor frames.

To summarize, two core parts of NeuroScaler are linked list-based modeling and frame type-based prioritization. We quantify the importance of the two parts by comparing NeuroScaler with two variants. The first variant only uses frame type-based prioritization, while the second variant only uses linked list-based modeling. A detailed description of the two variants is available in Appendix § A.1. For a comprehensive comparison, we also use the solution in Palantír for frame-level scheduling by setting the patch size equal to the frame size, although we will not go into details about the solution in Palantír for now. The preliminary experiment is based on the first benchmark video used in the evaluation part (Sec. 7). As shown in Fig. 3, the second variant achieves the worst SR quality (computed as the PSNR between the SR video and the original HR video). Therefore, the linked list model is highly inaccurate, and frame type-based prioritization is indispensable to NeuroScaler.

Considering the above result, it seems that adapting NeuroScaler to fine-grained scheduling requires using the fine-grained counterpart of type-based prioritization, *i.e.,* the types of small encoding

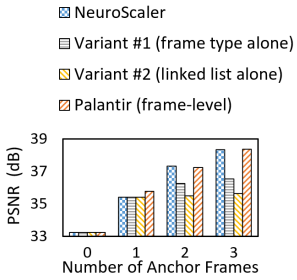


Figure 3: The SR qualities of NeuroScaler, two variants of NeuroScaler, and frame-level Palantír.

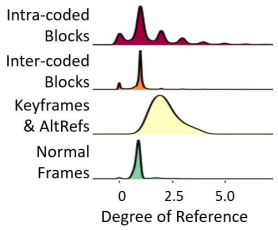


Figure 4: The distribution of degrees of reference for different coding units.

units such as blocks could implicitly imply their degrees of references and consequently their benefits as anchors. We validate such a counterpart using the 480p version of the first benchmark video. We separately measure the distribution of degrees of reference among different types of frames and blocks (including prioritized frame types, unprioritized normal frames, intra-coded blocks, and inter-coded blocks). The degree of reference for a given frame (or block) is quantitatively defined as $\frac{num\_references}{resolution}$, where $num\_references$ denotes the number of pixels that refer to the given frame (or block) for inter coding and $resolution$ denotes the number of pixels in the given frame (or block). As shown in Fig. 4, the degrees of references of keyframes and AltRefs are significantly greater than those of normal frames, while the distribution range of degrees of references for intra-coded blocks overlaps heavily with that for inter-coded blocks. We further use the common language effect size (CLES) [33] to quantify the results. The CLES is defined as the probability that a value randomly sampled from one distribution will be greater than that from another distribution. The CLES reaches 97.4% (close to the best case of 100%) between prioritized frame types and normal frames but is as low as 58.9% (close to the worst case of 50%) between different block types. Therefore, type-based prioritization is only helpful in the limited context of frame-level scheduling but hardly applies to patch-level scheduling.

4 Palantír Overview

Core methodology. As existing methods heavily rely on empirical insights that either require heavy operations or make sense in a limited context, we take an entirely new and first-principles approach. Our approach involves decomposing the SR video quality into the SR quality of every basic coding block and analyzing how the SR error propagates and accumulates at every basic coding block. Our first-principles reasoning suggests a DAG structure to represent the SR error propagation process. We discuss the approach later in Sec. 5.

Practical techniques. We also introduce two practical techniques so that we can construct the DAG by only using the LR video and improve the scheduling latency. We discuss them in Sec. 6.

Workflow. The workflow of Palantír is shown in Fig. 5. The streamer only uploads the LR video to the media server when it detects a limited uplink bandwidth. The server generates an SR error

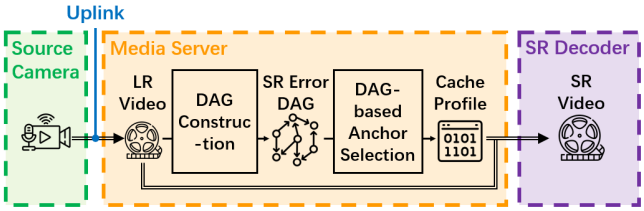


Figure 5: Palantír overview.

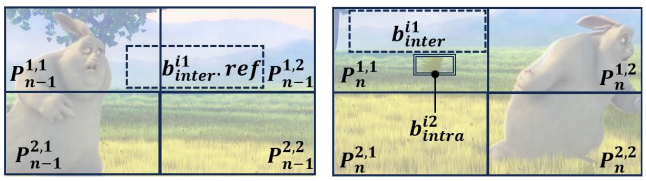


Figure 6: An example of SR error propagation.

DAG for every LR video segment whose time duration equals the pre-defined scheduling interval. The SR quality under any possible anchor patch set can be quickly estimated using the DAG, and beneficial anchor patches are greedily searched via DAG-based quality estimation. A cache profile is created, every bit of which indicates whether a patch in the LR segment is an anchor or not. In the existence of a powerful cloud server [48], both the LR segment and its corresponding cache profile are streamed to the server and then processed by the SR decoder on the server. Or, alternatively, the SR decoder can be executed in the streaming client.

5 First Principles Modeling

We take a first-principles approach to build Palantír. A first-principles approach involves breaking down complex problems into their fundamental parts and building up the solution from there. In neural-enhanced streaming, the anchor selection problem can be solved by estimating the video qualities under different anchor sets. The overall SR video quality is the average of the patch qualities, and each patch quality is fundamentally determined by the average quality of its most basic components - the blocks. In Sec. 5.1, we will follow this way of thinking to deduce our SR quality estimation strategy. In Sec. 5.2, we introduce our DAG model, which derives naturally from our analysis in Sec. 5.1 and allows us to fulfill lightweight yet reliable SR quality estimation.

5.1 Analysis of SR Error

We now separately discuss the SR errors (which are inversely related to the qualities) of non-anchor patches and anchor patches.

- **Case #1: non-anchor patches**.

  Analysis #1.1: every non-anchor patch P may consist of multiple inter-coded blocks ($b_{inter}^1, ..., b_{inter}^{n1}$) and intra-coded blocks ($b_{intra}^1, ..., b_{intra}^{n2}$). For analysis purposes, we split those blocks spanning multiple patches into multiple sub-blocks, each within a single patch. The SR error of P is:

$$\begin{aligned} &P.error \\ =&||P.SR - P.HR||_2^2 \\ =&\sum_{i=1}^{n1} ||b_{inter}^i.SR - b_{inter}^i.HR||_2^2 + \sum_{i=1}^{n2} ||b_{intra}^i.SR - b_{intra}^i.HR||_2^2 \\ =&\sum_{i=1}^{n1} b_{inter}^i.error + \sum_{i=1}^{n2} b_{intra}^i.error. \end{aligned} \tag{5}$$

  Example #1.1: as shown in Fig. 6, $P_n^{1,1}.error$ equals the sum of $b_{inter}^{i1}.error$, $b_{intra}^{i2}.error$, and the errors of many other blocks within $P_n^{1,1}$ (not marked due to the limited space).

Conclusion #1.1: the SR error of a non-anchor patch is the sum of the SR errors of all inter-coded and intra-coded blocks within the patch.

Analysis #1.2: According to the equation (2), we can further write the SR error of an inter-coded block b_{inter}^i as

$$\begin{aligned} &b_{inter}^i.error \\ =&||b_{inter}^i.SR - b_{inter}^i.HR||_2^2 \\ =&||b_{inter}^i.ref.SR + interp(b_{inter}^i.res, scale) - b_{inter}^i.HR||_2^2 \\ \approx&b_{inter}^i.ref.error + b_{inter}^i.res.complexity. \end{aligned} \tag{6}$$

where

$$b_{inter}^i.ref.error = ||b_{inter}^i.ref.SR - b_{inter}^i.ref.HR||_2^2 \tag{7}$$

and

$$\begin{aligned} &b_{inter}^i.res.complexity \\ =&||interp(interp(b_{inter}^i.HR - b_{inter}^i.ref.HR, scale^{-1}), scale) \\ &- (b_{inter}^i.HR - b_{inter}^i.ref.HR)||_2^2. \end{aligned} \tag{8}$$

A more detailed derivation of the equation (6)-(8) is in Appendix § A.2. Here, $b_{inter}^i.res.complexity$ relates to the **texture complexity** of the HR residual (*i.e.,* $b_{inter}^i.HR - b_{inter}^i.ref.HR$), since an HR residual with more complex texture details will experience a more significant deviation after the process of downscaling and re-upsampling, *i.e.,* $interp(interp(\cdot, scale^{-1}), scale)$.

Similarly, according to the equation (4), we have

$$b_{intra}^i.error = b_{intra}^i.complexity, \tag{9}$$

where

$$\begin{aligned} &b_{intra}^i.complexity \\ =&||interp(interp(b_{intra}^i.HR, scale^{-1}), scale) - b_{intra}^i.HR||_2^2 \end{aligned} \tag{10}$$

relates to the **texture complexity** of the HR content $b_{intra}^i.HR$.

Example #1.2: for b_{inter}^{i1} in the frame F_n (see Fig. 6), its SR error equals $b_{inter}^{i1}.res.complexity + b_{inter}^{i1}.ref.error$, where $b_{inter}^{i1}.ref$ is its reference block in F_{n-1}, a reference frame of F_n; for b_{intra}^{i2}, its SR error equals $b_{intra}^{i2}.complexity$.

Conclusion #1.2: the SR error of an inter-coded block depends on both the texture complexity of its HR residual

and the SR error of its reference block; while the SR error of an intra-coded block depends on the texture complexity of its HR content.

Analysis #1.3: combining the equation (5), (6), and (9), we have

$$P.error \approx P.TC + P.AE, \tag{11}$$

where

$$P.TC = \sum_{i=1}^{n1} b^i_{inter}.res.complexity + \sum_{i=1}^{n2} b^i_{intra}.complexity \tag{12}$$

indicates the **texture complexity** of the HR content or the HR residual, and

$$P.AE = \sum_{i=1}^{n1} b^i_{inter}.ref.error \tag{13}$$

is the **accumulated error** from depending blocks.

To reformulate (13) and simplify the modeling of patch-level SR error propagation, we make the following assumption (further discussed in Appendix § A.9):

Assumption #1.3: the per-pixel SR error within a patch is uniform - every pixel in the same patch shares exactly the same amount of error. Or, formally speaking, we assume that

$$p.error = ||p.SR - p.HR||_2^2 = \frac{P.error}{patch\_size} \tag{14}$$

holds for every pixel p in some patch P.

Denoting the set of reference patches of P as P^1, ..., P^{n3} and according to the **Assumption #1.3**, we can reformulate the equation (13) as

$$\begin{aligned} P.AE &= \sum_{i=1}^{n1} b^i_{inter}.ref.error \\ &= \sum_{i=1}^{n1} \sum_{p \in b^i_{inter}.ref} p.error \\ &= \sum_{i=1}^{n3} W^i \cdot P^i.error, \end{aligned} \tag{15}$$

where the weight coefficient W^i indicates the ratio of the number of referenced pixels in P^i to the patch size.

Example #1.3: for convenience, we assume that b^{i1}_{inter} is the only inter-coded block in $P^{1,1}_n$ (see Fig. 6). The weight between $P^{1,1}_n$ and $P^{1,1}_{n-1}$ equals 0.105, as the size of the intersecting region between $b^{i1}_{inter}.ref$ and $P^{1,1}_{n-1}$ is 0.105 of the patch size; similarly, the weight between $P^{1,1}_n$ and $P^{1,2}_{n-1}$ equals 0.323. According to the equation (11) and (15), $P^{1,1}_n.error$ can be approximated as $P^{1,1}_n.TC + 0.105 \cdot P^{1,1}_{n-1}.error + 0.323 \cdot P^{1,2}_{n-1}.error$.

From the equation (11), (12), and (15) we can make the final conclusion:

Conclusion #1.3: under the Assumption #1.3, the SR error of a non-anchor patch equals the weighted sum of the SR errors of its depending patches plus the texture complexities of its inter-coded HR residuals and its intra-coded HR contents.

- **Case #2: anchor patches**. In many neural-enhanced live streaming systems [30, 48], it is common to train or fine-tune the SR DNNs based on the current video content. Furthermore, it has been demonstrated to be feasible to share the same model across similar videos to reduce the overhead of training or fine-tuning [32]. Therefore, we make the following assumption (further discussed in Appendix § A.9):

  Assumption #2: the SR errors of anchor patches are 0.

5.2 DAG Modeling

A patch P may depend on another patch P^i for inter-coding only if the frame containing P^i is a reference frame of that containing P. Since the frame-level reference relationship is directed and acyclic, the patch-level dependency is also directed and acyclic. Therefore, we propose to use a directed acyclic graph (DAG) to represent the dependency, where every node corresponds to a patch and every edge indicates an inter-coding dependency.

A static ***weight*** attribute is associated with every edge to reflect the inter-node coding dependency level (*i.e.,* the ratio of pixels in the destination patch node that refer to the source patch node for coding), corresponding to W^i in the equation (15). Three attributes are associated with each node P: the static ***P.TC*** attribute, representing the texture complexity defined in the equation (12); the ***P.is_anchor*** attribute, indicating whether the patch node is an anchor or non-anchor under a given anchor patch set; and the ***P.error*** attribute, representing the SR error. When *P.is_anchor* equals 1, *P.error* equals 0 (according to the **Assumption #2**); otherwise, *P.error* equals the weighted sum of the *error* attributes of the predecessor nodes plus $P.TC$ (according to the **Conclusion #1.3**).

Formulation of the first design goal. As introduced in Sec. 4, the first design goal is to pinpoint the anchor patches that can result in a high-quality SR video. With our DAG-based modeling, the quality can be estimated as a proxy variable $-\sum_P P.error$. Furthermore, the inference overhead is affected by the number of anchor patches (*i.e.,* $\sum_P P.is\_anchor$). The first goal can then be reformulated as determining the optimal value of $P.is\_anchor$ for each node P in order to maximize $-\sum_P P.error$, while ensuring that the value of $\sum_P P.is\_anchor$ does not surpass some limit.

Comparison with the linked list model in NeuroScaler. We demonstrate before in Sec. 3 that our DAG model outperforms the linked list model even in the case of frame-level scheduling (by setting the DAG node size equal to the frame size). Here we give a brief discussion on the architectural improvements of the DAG model over the linked list model from two perspectives, and a more detailed discussion can be found in Appendix §A.3. *First*, the DAG model enables more accurate dependency representation. Although both the two models use edges to represent paths along which SR error propagates, our DAG model creates edges only where

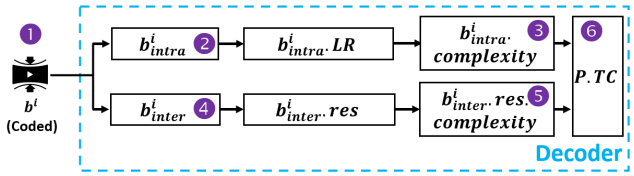


Figure 7: Determining the value of the static *TC* attribute.

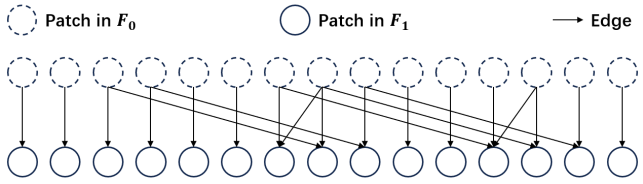


Figure 8: The part of a DAG, with 15 patches in each frame and F_0 being a reference frame of F_1.

actual coding dependencies exist, unlike the linked list model's fixed sequential connections. This is crucial for capturing the actual dependency in modern video codecs, including scenarios such as hierarchical B-picture coding structure [39, 40] and multi-reference video encoding [22]. *Second*, the DAG model adopts a dynamic ***weight*** attribute based on actual pixel reference ratios, whereas the linked list model assumes a fixed ***weight*** of 1. The adaptive attribute allows the DAG model to accurately model cases such as scene changes dominated by intra-frame coding, and multiple reference scenarios where dependency is distributed across frames.

Discussion of generalizability. We also discuss the generalizability of the DAG model in Appendix §A.4.

6 Practical Techniques

6.1 DAG Construction

At first glance of the equation (8) and (10), we need data from the HR video to compute the block-level texture complexities and then aggregate the block-level results to obtain the static $P.TC$ attribute according to (12). However, we find that using data from the HR video faces several challenges: (1) Commodity cameras can capture the HR data in the place, but they are typically not programmable [49] and thus not suitable for processing the task of DAG construction. Furthermore, video encoding and transmission alone are already computation-intensive and can cause severe problems such as device overheating [29]. Further running the scheduling task at the sender device can even make such problems more severe. (2) We choose to deploy the scheduling algorithm on the media server considering the above challenges of sender-side deployment. However, the media server only has access to the LR video due to the limited uplink bandwidth.

To enable scheduling on the media server, we propose to approximate the complexity of an inter-coded residual via

$$\begin{aligned} & b^i_{inter}.res.complexity \\ =&||interp(b^i_{inter}.res, scale) - (b^i_{inter}.HR - b^i_{inter}.ref.HR)||_2^2 \\ \approx& C||interp(interp(b^i_{inter}.res, 0.5), 2) - b^i_{inter}.res||_2^2 \end{aligned} \tag{16}$$

and approximate the complexity of an intra-coded block via

$$\begin{aligned} & b^i_{intra}.complexity \\ =&||interp(b^i_{intra}.LR, scale) - b^i_{intra}.HR||_2^2 \\ \approx& C||interp(interp(b^i_{intra}.LR, 0.5), 2) - b^i_{intra}.LR||_2^2, \end{aligned} \tag{17}$$

with C being a constant coefficient. We base our approximation on the following observation: if some content (*i.e.,* $b^i_{intra}.HR$) or some residual (*i.e.,* $b^i_{inter}.HR - b^i_{inter}.ref.HR$) is of high texture complexity, its downsampled version (*i.e.,* $b^i_{intra}.LR$ or $b^i_{inter}.res$) also tends to have high texture complexity. In other words, the texture complexity of a content or a residual is linearly related to that of its downsampled version. We validate the observation through a preliminary experiment and find that the Pearson correlation coefficient between the approximated texture complexity and the actual texture complexity is as high as 0.96.

With the above approximations, we can determine the $P.TC$ attribute by only resorting to data in the LR video. We slightly modify the decoder to fulfill the computation process. The workflow is shown in Fig. 7. While decoding a coded block b^i (❶), it computes either the complexity of its content or its residual, based on its coding type, and then adds the value to $P.TC$ (❻), where P is the patch containing the block. In the case of intra-coding (❷), $b^i_{intra}.complexity$ (❸) is computed from the decoded $b^i_{intra}.LR$, following the equation (17). In the case of inter-coding (❹), $b^i_{inter}.res.complexity$ (❺) is computed from the parsed $b^i_{inter}.res$, following the equation (16).

6.2 Parallel Selection

We employ a greedy searching algorithm to iteratively select the new anchor patch based on the estimated quality. We refer to the sequential implementation of this method as the vanilla Palantír. Specifically, the estimation processes for different anchor patch sets are executed sequentially, and the *error* attributes for different patch nodes under a given anchor patch set are also computed sequentially. As demonstrated later in our ablation experiment (see Sec. 7.4), the vanilla Palantír fails our second design goal of low-latency anchor selection. To address the issue, we propose a novel strategy to enable parallel and low-latency anchor selection in Palantír. The parallelization does not alter the selection results, so its benefits in latency do not come at the cost of compromising the energy efficiency of on-device neural enhancement or the monetary cost of cloud-based neural enhancement.

For clarity, we use an example to introduce our parallelism mechanism. As shown in Fig. 8, edges always start from some patch in F_0 and point to some patch in F_1. There exist neither edges along the opposite direction nor edges connecting patches within the same frame. We will utilize this phenomenon to optimize Palantír via both intra-set and inter-set parallelism. Note that the observed phenomenon is not accidental: edges indicate inter-frame coding references, and the coding reference relationship among frames is directed and acyclic in all mainstream codecs. Therefore, our optimization should be widely applicable.

Intra-set parallelism. Based on the above phenomenon, we can follow the frame decoding order to enumerate the DAG for quality estimation, *i.e.,* firstly compute the *error* attributes for the nodes in

F_0 and then deal with the nodes in F_1. The TC attributes of nodes in F_0 (or F_1) can be denoted as a vector TC_0 (or TC_1). Similarly, we use the notation $Error_i$ for the *error* attributes and Is_anchor_i for the *is_anchor* attributes ($i = 0, 1$). The *weight* attributes of the edges can be denoted by a sparse $Weight$ matrix. Note that the $Weight$ matrix is sparse since each patch in F_1 only refers to a limited set of patches in F_0 due to the limited range of motion vectors. For simplicity of discussion we assume that F_0 is the only reference frame of F_1, and the computation process of $Error_1$ can be formalized as $Error_1 = (TC_1 + Weight \cdot Error_0) \circ Is\_anchor_1$, where $\circ$ indicates the element-wise multiplication and $Weight \cdot Error_0$ is a parallelizable sparse matrix-vector multiplication (SpMV) operation. Since Is_anchor_1 is a binary vector, we implement the element-wise multiplication by using Is_anchor_1 as a mask to parallelly set the corresponding entries in $TC_1 + Weight \cdot Error_0$ to zero. As SpMV is a common operation in many applications and has already attracted many optimization efforts [18, 31, 45], parallelized SpMV can be achieved by using a mainstream matrix-related computation package such as PyTorch.

Inter-set parallelism. Batching is widely used to improve DNN inference throughput [6] due to the effect of the data dimension on parallelism opportunities [5, 35]. Therefore, we execute the quality estimation under several searched anchor sets in parallel by adding a batch dimension to both $Error_i$ and Is_anchor_i. The same $Weight$ matrix and TC_i vectors are shared among different samples in the batch. With the inter-set parallelism, the SpMV operations are converted into the sparse matrix-matrix (SpMM) operations, which are also well studied and supported for parallel implementation.

7 Evaluation

We evaluate Palantír by answering three questions:

- Does Palantír achieve the first design goal of selecting a beneficial anchor patch set?
- Does Palantír achieve the second design goal of incurring a negligible latency overhead for UHD live streaming?
- How does each component of Palantír contribute to its overall performance?

7.1 Experimental Setup

Implementation. We develop our decoder based on the open-source SR codec in NEMO [46]. We incorporate two novel modes into the decoder. The first is to obtain the data required for graph construction (as introduced in Sec. 6.1). The second is to take both an LR video and a corresponding cache profile as input, and then upscale patches by DNN-based or reusing-based SR. About 1500 lines of code are added to the open-source codec to support the two modes, accounting for less than 0.5% of the total lines of code in the entire codec. **The source code is available at https://palantir-sr.github.io.**

Video. We download six popular 4k@30fps videos from YouTube. To demonstrate the universality of Palantír, the videos contain six distinct categories, including `makeup review`, `computer gaming`, `skit`, `shopping`, `car review`, and `unboxing`. We use FFmpeg (v3.4) [3] to transcode the HR video into the 480p (854×480) LR version in real time. We follow encoding guidelines to set the bitrate to `1800` `kbps`, the encoding speed to `5` [2], and the group of pictures (GoP) to `60` frames (*i.e.,* 2 seconds) [43]. We use the `-auto-alt-ref` option in FFmpeg to enable the alternative reference frame feature required by the anchor selection algorithm in NeuroScaler. Unless noted otherwise, we use the first five minutes of each video.

SR DNN. We adopt the DNN model of NAS [47]. We empirically set the number of residual blocks to 8 and the number of filters to 48. The DNN upscales the resolution of the LR video by 4 times. As the feasibility of online training for live streaming has been demonstrated [30], we train the DNN model for each benchmark video. When comparing the performance of different methods on the same video, the same DNN model is used for fairness.

Anchor patch size. We use a patch size of 170×160 to trade off anchor effectiveness and scheduling latency. Consequently, each LR frame consists of 15 patches.

Baselines. We use four baselines in this part. The first is the Per-frame baseline, which applies DNN-based SR on all the frames. The second is the NeuroScaler baseline, which uses the algorithm in NeuroScaler [48] to select the anchor frame set. The third is the NEMO [46] baseline. The fourth is the Key+Uniform baseline, which selects all the patches in the keyframe and equally spaced patches in the remaining frames as anchor patches.

Scheduling interval. Unless otherwise noted, we use a scheduling interval equal to the GoP (*i.e.,* 2 seconds).

Parallelism. The parallelized Palantír and the vanilla Palantír are two different implementations of the same selection method and lead to the same anchor patch set, so the results in Sec. 7.2 apply to both implementations. The latency results in Sec. 7.3 are obtained using the parallelized Palantír. Finally, the two implementations are compared in Sec. 7.4.

Hardware. We use a server with a 16-core AMD Ryzen processor and an NVIDIA A10 GPU as our media server, where graph construction and anchor selection are performed. The scheduling latency is measured on the server. Considering the deployment cost constraint in real-world applications, we only use CPU for Palantír, the NeuroScaler baseline, and the Key+Uniform baseline. However, to speed up our experiment, we further use the NVIDIA A10 GPU for the NEMO baseline, as the NEMO baseline requires numerous times of SR DNN inferences for anchor frame selection. We also use a Xiaomi 12S smartphone, which was announced in July 2022 and equipped with the Qualcomm Snapdragon 8+ Gen 1 Mobile Platform, to measure the energy efficiency when running SR DNN inference on mobile receiver devices.

7.2 Anchor Effectiveness

Quality Gain. We compute the peak-signal-to-noise-ration (PSNR) between the SR video and the original HR video to quantify the effectiveness of an anchor set. To make a fair comparison, we keep the total sizes of the anchor regions the same, *i.e.,* compare the quality gain under the m-element anchor frame set selected by some frame-level baseline with that under the $(15 \cdot m)$-element anchor patch set selected by Palantír or the Key+Uniform baseline. The only exception here is the Per-frame baseline, where all the frames are always treated as anchors and the size of the anchor

Table 1: A comparison of anchor effectiveness.

Video	Per-frame (dB)	DNN-based SR + Reusing-based SR (dB)			
		NeuroScaler	NEMO	Key+Uniform	Palantír
1	10.5	(2.2, 4.1, 5.1)	(2.8, 4.4, 5.4)	(2.1, 2.8, 3.2)	**(3.8, 5.4, 6.4)**
2	5.7	(0.5, 2.0, 2.6)	(1.7, 2.4, 3.0)	(0.5, 1.4, 2.0)	**(1.9, 2.8, 3.3)**
3	6.5	(1.1, 2.3, 3.1)	**(2.3, 3.2, 3.7)**	(1.1, 1.6, 1.9)	(2.1, 3.0, 3.5)
4	11.6	(4.4, 6.4, 7.4)	(4.7, 6.6, 7.6)	(4.4, 5.1, 5.6)	**(5.4, 7.1, 8.0)**
5	6.6	(3.8, 4.8, 5.2)	(4.2, 5.0, 5.3)	(3.8, 4.3, 4.5)	**(4.2, 5.1, 5.5)**
6	8.9	(3.0, 4.4, 5.1)	(3.5, 4.7, 5.3)	(3.0, 3.3, 3.6)	**(4.0, 5.3, 5.9)**
Average	8.3	(2.5, 4.0, 4.7)	(3.2, 4.4, 5.1)	(2.5, 3.1, 3.5)	**(3.6, 4.8, 5.4)**

Annotations: For any triple located in the last three columns, it is composed of the quality gain under $m = 1$, that under $m = 2$, and that under $m = 3$.

Table 2: A comparison of energy overhead.

Video	NeuroScaler (mAh)	Palantír (mAh)
1	(27.63, 41.33, 63.40)	**(21.21, 30.73, 41.72)**
2	(28.94, 42,27, 56,21)	**(18.06, 26.17, 40.82)**
3	(26.92, 41.48, 63.94)	**(21.07, 34.90, 47.91)**
4	(28.75, 41.71, 63.40)	**(26.08, 37.77, 47.05)**
5	(26.98, 41.18, 62.79)	**(26.37, 35.88, 45.47)**
6	(27.15, 40.64, 63.05)	**(23.98, 32.10, 40.85)**

Annotations: For any triple located in the second column, it is composed of the energy overhead under $|AF| = 1$, that under $|AF| = 2$, and that under $|AF| = 3$. As for any triple located in the last column, each of its elements is the result under some AP corresponding to the AF.

regions is thus always larger than other methods. Furthermore, we empirically limit m to not be greater than 3 since: (1) $m = 3$ can deliver quality gains that are comparable to the setting of applying DNN-based SR on all frames; (2) further increasing the value of m leads to a significant overhead.

The results are shown in Table. 1, from which we have several observations: (1) Palantír outperforms all the baselines with its ability to identify beneficial patches. (2) When compared with the NeuroScaler baseline, the state-of-the-art real-time scheduling method, Palantír boosts the quality gain of neural enhancement by 3.7 times at most and 1.4 times on average. (3) Compared to the non-realtime NEMO baseline, which requires significant time for initial measurements and does not support live streaming, Palantír still achieves a quality enhancement of up to 1.4 times in optimal conditions and 1.1 times on average. However, Palantír exhibits slightly inferior performance to NEMO in the worst-case scenario, specifically on the third benchmark video. A comprehensive analysis of this performance decline is provided in Appendix § A.5. (4) The fine-grained scheduling-based Key+Uniform baseline even falls behind the coarse-grained scheduling-based NeuroScaler baseline, so the effect of fine-grained scheduling heavily depends on the anchor selection method. (5) Palantír reduces the SR DNN inference overhead by 20 times with $m = 3$ (or 60 times with $m = 1$) while compared to the Per-frame baseline. With such a remarkable overhead reduction, Palantír still preserves 54.0-82.6% (or 32.8-64.0%) of the quality gain of the Per-frame baseline.

Energy Efficiency. We now examine how the anchor efficiency of Palantír transfers to energy efficiency when running the SR decoder on mobile devices. The detailed energy consumption measurement method is presented in Appendix § A.6.

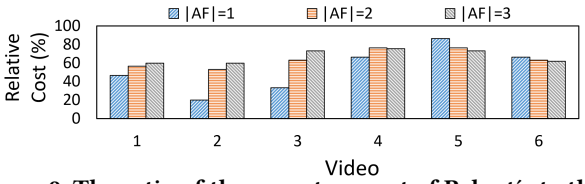


Figure 9: The ratio of the monetary cost of Palantír to that of the NeuroScaler baseline.

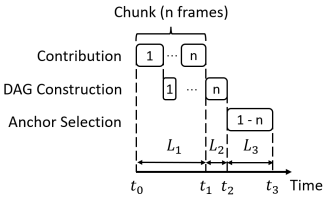


Figure 10: The timeline of Palantír. The numbers within the blocks represent the frame indices.

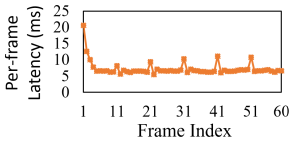


Figure 11: Per-frame latency of decoding for DAG construction during a GoP.

For each anchor frame set AF selected by the NeuroScaler baseline, we find the minimal anchor patch set AP which is selected by Palantír and achieves an equivalent or higher PSNR than AF. We compare the energy overhead under AF with that under AP. As shown in Table. 2, Palantír reduces the energy overhead over all cases. The reduction ratio is 38.1% at most and 22.4% on average.

Monetary cost reduction. We now present how Palantír reduces the monetary cost when running the SR decoder on cloud servers. We use the same method as measuring energy efficiency to find the corresponding AF for every AP. The monetary cost is estimated to be linear to the DNN computation complexity under the cache profile (AF or AP), and the keras-flops package [42] is used to measure the computation complexity. The ratio of the monetary cost incurred by Palantír to that incurred by the NeuroScaler baseline is presented in Fig. 9. Compared to NeuroScaler, Palantír reduces the monetary cost by 80.1% at most and 38.4% on average.

7.3 Scheduling Latency

End-to-end (E2E) latency is an important metric in live streaming [1, 7, 19, 41]. To ensure that the live streaming latency can be lower than the GoP, modern streaming standards such as CMAF [26] allow a chunk (which can be part of a GoP) to be immediately packaged (*i.e.,* chunked packaging [9]) and delivered (*i.e.,* chunked delivery [9]) when ready. Here we denote the chunk length as n frames and assume the scheduling interval of Palantír to be equal to n for simplicity. As shown in Fig. 10, the streamer contributes new video frames at a constant rate. Every new chunk of n frames is contributed per time duration of $L_1 = \frac{n}{frame\_rate}$. In a traditional streaming pipeline without neural enhancement, the new chunk can be immediately packaged and delivered at t_1. However, two additional latency sources are presented in Palantír, *i.e.,* the DAG construction latency L_2 and the DAG-based anchor selection L_3. We evaluate whether $L_2 + L_3$ is small enough to well support latency-sensitive UHD live streaming applications.

We first examine the value of L_2. Note that we can directly feed a newly contributed frame to the decoder (working in the first mode introduced in Sec. 7.1) for DAG construction, so L_2 should be equal to the processing latency of the last frame in the scheduling interval if the decoder runs above 30fps. As shown in Fig. 11, the measured per-frame decoding latency for DAG construction is indeed always below 33 ms, so we estimate L_2 to be the average of the measured per-frame decoding latencies in Fig. 11, *i.e.,* 7.2ms.

The DAG-based anchor selection latency L_3 depends on the scheduling interval and the ratio of anchor patches. For ULL UHD live streaming applications [41] requiring an E2E latency below 200ms, we set the scheduling interval to be 66.67ms (*i.e.,* 2 frames in the 30-fps evaluation videos). As for LL live streaming applications [1, 7, 19] whose E2E latency requirements range from 2 seconds to 10 seconds, we consider five different settings (with the latency requirement being 2s, 4s, 6s, 8s, and 10s, respectively) and set the scheduling interval to be one-fifth of the latency requirement under each setting. As illustrated in Sec. 7.2, using only an anchor ratio of 5% can lead to large quality gains, so we set the ratio to be 5% for latency measurement. Under the above settings, the relationship between the overall scheduling latency $L_2 + L_3$ (with L_2 fixed to 7.2ms) and the E2E latency requirement is measured and plotted in Table. 3. We have two observations from the results: (1) Although the scheduling latency of Palantír is a little higher than that of NeuroScaler, it only accounts for a negligible portion (*i.e.,* 0.6-3.9%) of the E2E latency requirement. (2) As for the NEMO baseline, its scheduling latency is 88.2-225.6 times higher than that of Palantír and fails to support live streaming. A detailed discussion of NEMO's high latency is available in Appendix §A.7.

As a final note, the scheduling latency of Palantír can be even further optimized if necessary. We demonstrate in Appendix §A.8 that a simple modification can further reduce the latency by 3.1 times while incurring a quality degradation of no more than 0.1dB.

7.4 Ablation Study

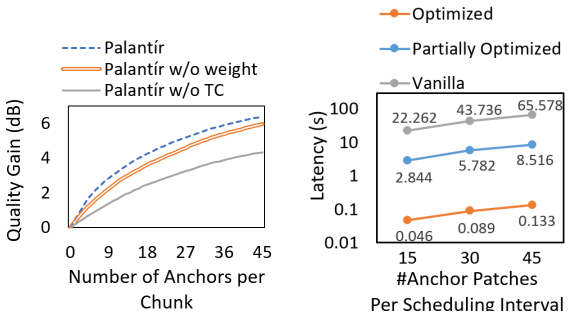
SR error DAG. The key to selecting a beneficial anchor patch set is our DAG-based modeling. We use a theoretical analysis to determine the values of the static *weight* attributes of the edges and the static *TC* attributes of patch nodes (see Sec. 5 and 6.1). To quantify the importance of setting appropriate values, we evaluate with the `makeup review` video and compare Palantír with two variants. In the first variant (Palantír w/o *weight*), the only predecessor node of the patch node $P_n^{i,j}$ (located at the i-th row and j-th column of the patch grid of the n-th frame) is $P_{n-1}^{i,j}$ and the weight of the edge connecting them equals 1. However, the *TC* attributes in the first variant are kept the same as in Palantír. Note that the first variant resembles the NeuroSclaer baseline when the patch size equals the frame resolution. In the second variant (Palantír w/o *TC*), the *weight* attributes are kept the same as in Palantír, but the *TC* attributes of all nodes are set to 1. As shown in Fig. 12(a), Palantír consistently outperforms the two variants.

Parallel Searching. We have introduced intra-set and inter-set parallelism (see Sec. 6.2) to speed up the quality estimation process in our greedy searching algorithm. We measure the DAG-based anchor selection latency (*i.e., L_3*) under three different settings: (1) the vanilla Palantír- nodes in the original DAG are processed serially for estimation; (2) the partially optimized Palantír setting - only the intra-set parallelism is enabled; (3) the optimized Palantír- both the two parallelism mechanisms are enabled. The measured DAG-based selection latencies (*i.e., L_3*) are shown in Fig. 12(b), showing that the optimized Palantír speeds up selection by 493 times at most.

Table 3: The relationship between the overall scheduling latency and the E2E latency requirement.

Setting	E2E Latency Requirement (ms)	Scheduling Latency (ms)		
		NeuroScaler	Palantír	NEMO
ULL	200	0.5	7.7	678.9
LL: Setting 1	2000	1.4	12.6	2842.2
LL: Setting 2	4000	2.0	27.0	5975.5
LL: Setting 3	6000	2.6	56.9	10509.3
LL: Setting 4	8000	5.0	92.4	16656.3
LL: Setting 5	10000	6.5	140.6	23126.7

Annotations: The presented result is the average latency over all the videos as the latencies for different videos are fairly close.



(a) Analysis on the DAG's attributes.

(b) Analysis on the parallelism mechanisms.

Figure 12: Ablation study.

8 Conclusion

In this work, we propose Palantír, the first neural-enhanced UHD live streaming system with fine-grained scheduling. Palantír seeks to improve efficiency via reasonable scheduling while minimizing the scheduling latency. Based on our first-principles reasoning, Palantír adopts DAG-based quality estimation to select a beneficial anchor patch set with low computation cost. Palantír further integrates two practical challenges to improve its feasibility and latency. The evaluation results demonstrate the superiority of Palantír.

Acknowledgments

This work is supported in part by the National Key Research and Development Program of China under grant No. 2024YFC2607400, the National Natural Science Foundation of China under grant No. 62302259, 62432008, 62472248, 62371269, and the Guangdong Innovative and Entrepreneurial Research Team Program under grant No. 2021ZT09L197.

References

[1] 2022. GB/T 28181-2022 PDF English. https://www.chinesestandard.net/PDF.aspx/GBT28181-2022, last accessed on Feb. 18, 2025.

[2] 2023. Live encoding with VP9 using FFmpeg. https://developers.google.com/media/vp9/live-encoding, last accessed on Mar. 15, 2024.

[3] 2024. FFmpeg. https://ffmpeg.org/, last accessed on Mar. 15, 2024.

[4] 2024. webm/libvpx - Git at Google. https://chromium.googlesource.com/webm/libvpx, last accessed on Mar. 15, 2024.

[5] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Savannah, GA, 265–283.

[6] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2020. Batch: machine learning inference serving on serverless platforms with adaptive batching. In *Proceedings of the 2020 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Article 69, 15 pages.

[7] Amazon Web Services, Inc. 2025. Video Latency in Live Streaming. https://aws.amazon.com/media/tech/video-latency-in-live-streaming, last accessed on Feb. 18, 2025.

[8] Duin Baek, Mallesham Dasari, Samir R. Das, and Jihoon Ryoo. 2021. DcSR: Practical Video Quality Enhancement Using Data-Centric Super Resolution. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*. Association for Computing Machinery, New York, NY, USA, 336–343.

[9] Abdelhak Bentaleb, May Lim, Mehmet N. Akcay, Ali C. Begen, Sarra Hammoudi, and Roger Zimmermann. 2023. Toward One-Second Latency: Evolution of Live Media Streaming. (2023). arXiv preprint arXiv:2310.03256.

[10] Xuecheng Chen, Haoyang Wang, Yuhan Cheng, Haohao Fu, Yuxuan Liu, Fan Dang, Yunhao Liu, Jinqiang Cui, and Xinlei Chen. 2024. DDL: Empowering Delivery Drones With Large-Scale Urban Sensing Capability. *IEEE Journal of Selected Topics in Signal Processing* 18, 3 (2024), 502–515.

[11] Xuecheng Chen, Haoyang Wang, Zuxin Li, Wenbo Ding, Fan Dang, Chengye Wu, and Xinlei Chen. 2023. DeliverSense: Efficient Delivery Drone Scheduling for Crowdsensing with Deep Reinforcement Learning. In *Adjunct Proceedings of the 2022 ACM International Joint Conference on Pervasive and Ubiquitous Computing and the 2022 ACM International Symposium on Wearable Computers*. Association for Computing Machinery, New York, NY, USA, 403–408.

[12] Xuecheng Chen, Zijian Xiao, Yuhan Cheng, Chen-Chun Hsia, Haoyang Wang, Jingao Xu, Susu Xu, Fan Dang, Xiao-Ping Zhang, Yunhao Liu, and Xinlei Chen. 2024. SOScheduler: Toward Proactive and Adaptive Wildfire Suppression via Multi-UAV Collaborative Scheduling. *IEEE Internet of Things Journal* 11, 14 (2024), 24858–24871.

[13] Yuhan Cheng, Jirong Zha, Renjue Yang, Zhi Sun, Susu Xu, and Xinlei Chen. 2024. Multi-Agent Target Pursuit Using Perception Uncertainty-Aware Reinforcement Learning. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. Association for Computing Machinery, New York, NY, USA, 1992–1997.

[14] Coastal Safety Group. 2021. Beach Cameras and Image Analytics——Coastal Safety Group. https://coastalsafetygroup.com.au/news/beach-cameras-and-image-analytics, last accessed on Jun. 26, 2024.

[15] CommsEase. 2022. AI Super-Resolution. https://doc.commsease.com/en/nertc/guide/zYzMjc0NTA?platform=android, last accessed on Jun. 26, 2024.

[16] Flyability. 2024. Ultimate Guide to Wind Turbine Inspection Techniques. https://www.flyability.com/blog/wind-turbine-inspection, last accessed on Jun. 26, 2024.

[17] Global Market Insights Inc. 2024. 4K Technology Market, Share & Analysis Report, 2024-2032. https://www.gminsights.com/industry-analysis/4k-technology-market, last accessed on Jun. 26, 2024.

[18] Constantino Gómez, Filippo Mantovani, Erich Focht, and Marc Casas. 2021. Efficiently running SpMV on long vector architectures. In *Proceedings of the 26th ACM Symposium on Principles and Practice of Parallel Programming*. Association for Computing Machinery, New York, NY, USA, 292–303.

[19] Google. 2025. Live streaming latency - YouTube Help. https://support.google.com/youtube/answer/7444635?hl=en, last accessed on Feb. 18, 2025.

[20] Google. 2025. YouTube recommended upload encoding settings. https://support.google.com/youtube/answer/1722171?hl=en, last accessed on Jun. 18, 2025.

[21] Adrian Grange, Peter de Rivaz, and Jonathan Hunt. 2016. VP9 Bitstream & Decoding Process Specification v0.6. https://storage.googleapis.com/downloads.webmproject.org/docs/vp9/vp9-bitstream-specification-v0.6-20160331-draft.pdf, last accessed on Mar. 15, 2024.

[22] Yu-Wen Huang, Bing-Yu Hsieh, Shao-Yi Chien, Shyh-Yih Ma, and Liang-Gee Chen. 2006. Analysis and complexity reduction of multiple reference frames motion estimation in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology* 16, 4 (2006), 507–522.

[23] Indrajit Ghosh. 2024. List of the Best AV1 Hardware Support in 2024 – Encode & Decode. https://www.faceofit.com/av1-hardware-support-in-2024/, last accessed on Feb. 18, 2025.

[24] International Olympic Committee. 2022. Beijing 2022 set to be the most immersive Olympic Winter Games yet. https://olympics.com/ioc/news/beijing-2022-set-to-be-the-most-immersive-olympic-winter-games-yet, last accessed on Jun. 26, 2024.

[25] International Olympic Committee. 2024. Intel unveils AI-Platform Innovation for Paris 2024. https://olympics.com/ioc/news/intel-unveils-ai-platform-innovation-for-paris-2024, last accessed on Jun. 26, 2024.

[26] ISOIEC JTC 1SC 29. 2024. ISOIEC 23000-19:2024 — Multimedia application format (MPEG-A) — Part 19: Common media application format (CMAF) for segmented media. https://www.iso.org/standard/85623.html, last accessed on Feb. 18, 2025.

[27] JDT Developer. 2022. The practice and application of video super-resolution technology. https://developer.jdcloud.com/en/article/2267, last accessed on Jun. 26, 2024.

[28] Zhuozhu Jian, Qixuan Li, Shengtao Zheng, Xueqian Wang, and Xinlei Chen. 2024. LVCP: LiDAR-Vision Tightly Coupled Collaborative Real-time Relative Positioning. arXiv:2407.10782 [cs.RO] https://arxiv.org/abs/2407.10782

[29] Soowon Kang, Hyeonwoo Choi, Sooyoung Park, Chunjong Park, Jemin Lee, Uichin Lee, and Sung-Ju Lee. 2019. Fire in Your Hands: Understanding Thermal Behavior of Smartphones. In *Proceedings of the 25th Annual International Conference on Mobile Computing and Networking*. Association for Computing Machinery, New York, NY, USA, Article 13, 16 pages.

[30] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. 2020. Neural-Enhanced Live Streaming: Improving Live Video Ingest via Online Learning. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*. Association for Computing Machinery, New York, NY, USA, 107–125.

[31] Kenli Li, Wangdong Yang, and Keqin Li. 2015. Performance Analysis and Optimization for SpMV on GPU Using Probabilistic Modeling. *IEEE Transactions on Parallel and Distributed Systems* 26, 1 (2015), 196–205.

[32] Jinyeong Lim, Juncheol Ye, Jaehong Kim, Hwijoon Lim, Hyunho Yeo, and Dongsu Han. 2023. Neural Cloud Storage: Innovative Cloud Storage Solution for Cold Video. In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems*. Association for Computing Machinery, New York, NY, USA, 1–7.

[33] K. O. McGraw and S. P. Wong. 1992. A common language effect size statistic. *Psychological Bulletin* 111, 2 (1992), 361–365.

[34] Microsoft Edge Team. 2023. Video super resolution in Microsoft Edge. https://blogs.windows.com/msedgedev/2023/03/08/video-super-resolution-in-microsoft-edge/, last accessed on Jun. 26, 2024.

[35] Graham Neubig, Yoav Goldberg, and Chris Dyer. 2017. On-the-fly Operation Batching in Dynamic Computation Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc.

[36] Ookla, LLC. 2024. Speedtest Global Index - Internet Speed around the world. https://www.speedtest.net/global-index, last accessed on Jun. 26, 2024.

[37] Jiyuan Ren, Yanggang Xu, Zuxin Li, Chaopeng Hong, Xiao-Ping Zhang, and Xinlei Chen. 2023. Scheduling UAV Swarm with Attention-based Graph Reinforcement Learning for Ground-to-air Heterogeneous Data Communication. In *Adjunct Proceedings of the 2023 ACM International Joint Conference on Pervasive and Ubiquitous Computing & the 2023 ACM International Symposium on Wearable Computing*. Association for Computing Machinery, New York, NY, USA, 670–675.

[38] Mário Saldanha, Marcel Corrêa, Guilherme Corrêa, Daniel Palomino, Marcelo Porto, Bruno Zatt, and Luciano Agostini. 2020. An Overview of Dedicated Hardware Designs for State-of-the-Art AV1 and H.266/VVC Video Codecs. In *Proceedings of the 27th IEEE International Conference on Electronics, Circuits and Systems*. IEEE, 1–4.

[39] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. 2006. Analysis of Hierarchical B Pictures and MCTF. In *Proceedings of the 2006 IEEE International Conference on Multimedia and Expo*. IEEE, 1929–1932.

[40] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. 2007. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology* 17, 9 (2007), 1103–1120.

[41] Soliton Systems. 2025. Beyond Line of Sight Drones with Ultra Low Latency. https://www.solitonsystems.com/low-latency-video/remote-operation/beyond-line-of-sight-command-and-control-of-drones, last accessed on Feb. 18, 2025.

[42] Tokusumi. 2020. keras-flops · pypi. https://pypi.org/project/keras-flops/, last accessed on Feb. 18, 2025.

[43] Twitch. 2024. Broadcast Guidelines. https://help.twitch.tv/s/article/broadcast-guidelines?language=en_US, last accessed on Mar. 15, 2024.

[44] Haoyang Wang, Xuecheng Chen, Yuhan Cheng, Chenye Wu, Fan Dang, and Xinlei Chen. 2023. H-SwarmLoc: Efficient Scheduling for Localization of Heterogeneous MAV Swarm with Deep Reinforcement Learning. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*. Association for Computing Machinery, New York, NY, USA, 1148–1154.

[45] Guoqing Xiao, Kenli Li, Yuedan Chen, Wangquan He, Albert Y. Zomaya, and Tao Li. 2021. CASpMV: A Customized and Accelerative SpMV Framework for the Sunway TaihuLight. *IEEE Transactions on Parallel and Distributed Systems* 32, 1 (2021), 131–146.

[46] Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2020. NEMO: Enabling Neural-Enhanced Video Streaming on Commodity Mobile Devices. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. Association for Computing Machinery, New York, NY, USA, Article 28, 14 pages.

[47] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. 2018. Neural adaptive content-aware internet video delivery. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*. 645–661.

[48] Hyunho Yeo, Hwijoon Lim, Jaehong Kim, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2022. NeuroScaler: Neural Video Enhancement at Scale. In *Proceedings of the ACM SIGCOMM 2022 Conference*. Association for Computing Machinery, New York, NY, USA, 795–811.

[49] Mu Yuan, Lan Zhang, Xuanke You, and Xiang-Yang Li. 2023. PacketGame: Multi-Stream Packet Gating for Concurrent Video Inference at Scale. In *Proceedings of the ACM SIGCOMM 2023 Conference*. Association for Computing Machinery, New York, NY, USA, 724–737.

[50] ZEGOCLOUD. 2025. ZegoExpressEngine. https://docs.zegocloud.com/article/api?doc=express_video_sdk_API~java_android~class~ZegoExpressEngine#enable-video-super-resolution, last accessed on Feb. 18, 2025.