X4Rate: A Scene-Adaptive Extrapolation Framework for 4× Frame Rate Upsampling

Akanksha Dixit[a] and Smruti R. Sarangi[b]
Department of Electrical Engineering, Indian Institute of Technology Delhi, New Delhi, India
Akanksha.Dixit@ee.iitd.ac.in, srsarangi@cse.iitd.ac.in

Keywords: Extrapolation, Real-Time Rendering, LinUCB.

Abstract: High-frequency displays are widely used in gaming and VR; sadly, GPUs often fail to sustain high frame rates, leading to visual artifacts like judder and motion blur. State-of-the-art approaches to enhance frame rates include interpolation and extrapolation. Interpolation typically yields higher-quality results because it leverages both past and future frames. However, it is more suitable for doubling frame rates, as further upsampling requires multiple passes, necessitating repeated forward and backward frame references, which introduces unwanted latency and complexity. Extrapolation, on the other hand, can continuously generate new frames at high rates. In this work, we aim to increase the frame rate by 4×, making extrapolation the preferred choice. However, no single extrapolation algorithm performs consistently well across all scene types. Each one is tailored to specific scenarios and exhibits strengths and weaknesses depending on the context. We propose *X4Rate*, a frame generation framework that combines multiple existing methods to deliver results superior to any individual algorithm by selecting the best algorithm for the current scene. It is basically a decision predictor (meta-learner) that dynamically selects the best-performing frame extrapolation algorithm at runtime. This adaptive selection ensures high visual quality consistently because when one method underperforms, others compensate. *X4Rate* achieves 4× the rendering rate while delivering high-quality frames, outperforming the nearest competitor, ExtraNet, with a 22.13% improvement in the PSNR.

1 INTRODUCTION

The global gaming market is rapidly growing, with revenues expected to reach 691.3 billion USD by 2029 (Insights, 2025). While modern displays now support refresh rates up to 240 Hz, GPUs often fail to render frames consistently at such rates for high resolutions like 1080p or 1440p. Hence, we need advanced frame generation techniques. This work proposes such a framework that generates frames at 4× the rendering rate (ex: raises the frame rate of 60 fps to up to 240 fps). This process of creating new frames using already rendered frames is known as *frame generation* or *temporal supersampling* (Yang et al., 2024).

Two popular frame generation methods are *Interpolation* (Niklaus and Liu, 2020; Zhang et al., 2023) and *Extrapolation* (Guo et al., 2021). Interpolation methods such as NVIDIA's Deep Learning Super-Sampling (DLSS 3) (Burnes and C Lin, 2023) are more suitable for doubling frame rates, as further upsampling requires multiple passes. For instance, to create $F_{t+0.25}$, one must first generate $F_{t+0.5}$ using F_t and F_{t+1}, then use F_t and $F_{t+0.5}$ to produce $F_{t+0.25}$. This iterative process not only increases latency but also adds computational complexity. Consequently, interpolation has a naturally inherent latency component (Guo et al., 2021). Hence, it is impractical to use interpolation for 4× temporal supersampling. Extrapolation, in contrast, predicts future frames using only past frames (e.g., generating $F_{t+0.25}$ or $F_{t+0.5}$ directly from F_t), avoiding the need for future inputs and simplifying the process. While this may reduce perceptual quality slightly, it is more suitable for low-latency, high-rate frame synthesis. Our goal in this work is to realize 4× frame rate upsampling using extrapolation, while maximizing visual quality.

There are several state-of-the-art frame extrapolation algorithms, such as ExtraNet (Guo et al., 2021), ExtraSS (Wu et al., 2023), Mob-FGSR (Yang et al., 2024), and GFFE (Wu et al., 2024), each designed with specific scene characteristics in mind and exhibiting unique strengths and limitations. ExtraNet, for instance, effectively handles dynamic lighting and

[a] https://orcid.org/0009-0009-1565-7199
[b] https://orcid.org/0000-0002-1657-8523

shadows using a history encoder that incorporates information from multiple past frames, but suffers from ghosting in fast-moving foregrounds (Wu et al., 2023). ExtraSS mitigates this ghosting using G-buffer-guided warping, which improves motion estimation and edge preservation. GFFE specializes in addressing out-of-screen disocclusions but struggles with unseen disocclusions and dynamic lighting due to its limited access to past frames and G-buffer data, unlike ExtraNet and ExtraSS. Mob-FGSR, while lightweight and efficient for mobile platforms, omits shading corrections, leading to artifacts in complex lighting scenarios (unlike ExtraSS, which has a separate shading correction network). Given this diversity, no method performs best across all conditions, underscoring the need for an adaptive selection strategy.

This is aligned with the No Free Lunch (NFL) theorem (Adam et al., 2019; Wolpert et al., 1995), which states that no algorithm is universally superior across all test cases. Each algorithm embodies a particular inductive bias that makes it more or less effective depending on the characteristics of the data. Hence, it is necessary to dynamically select the most suitable algorithm for each frame. This adaptive algorithm selection approach ensures consistently high-quality output by dynamically choosing the most suitable method per frame because when one underperforms, others compensate. In computer vision, several tasks such as semantic segmentation (Aguiar et al., 2019), plant species identification (Verma et al., 2023), and multi-target regression (Aguiar et al., 2022) have adopted meta-learning (MtL) (Rice, 1976; Brazdil et al., 2008) to guide algorithm selection. MtL leverages performance information from prior tasks to infer the most appropriate algorithm for new, unseen tasks using meta-features that describe input characteristics. In this work, we propose an adaptive algorithm selection strategy that combines MtL with the Contextual Multi-Armed Bandit (CMAB) framework (Lu et al., 2010). While MtL and CMAB both utilize contextual information under uncertainty, they follow different paradigms. MtL is typically employed offline; it learns from historical performance across multiple datasets to generalize to new ones. In contrast, CMAB operates online, making sequential decisions at inference time by balancing the exploration-exploitation trade-off. Our approach bridges these paradigms by using meta-features extracted from the current frame (Bensusan and Kalousis, 2001; Bilalli et al., 2017; Rivolli et al., 2021) as contextual signals in an online CMAB framework.

This work proposes a novel meta-learner *X4Rate* for selecting the optimal frame extrapolation algorithm in real time. Specifically, this paper makes two key contributions. First, we perform a comprehensive characterization of scene properties and extract meta-features that are correlated with the choice of the algorithm. Then, we design a meta-learner to dynamically select the best-performing frame extrapolation algorithm. We designed *X4Rate* as a plugin and attached it with Unreal Engine (UE) v5.1 such that it is accessible to game developers. We chose this engine because of its popularity and due to the fact that the software's source code is freely available at this link (ueg, 2025). It is easy to customize the plugin for other gaming engines. Our primary contributions are thus as follows:

❑ We identify meta-features that define the current state (context) of the scene (the presence of dynamic objects and camera movement) and the system's usage (hardware performance counters) that affect the performance of extrapolation algorithms.

❑ We propose a meta-learner (decision predictor) that uses these identified features to intelligently choose between three frame extrapolation algorithms to increase the frame rate without affecting the perceived image quality adversely.

❑ We were able to supersample the frame rate by $4\times$. We record a 22.13%, 20.13%, 18.17% and 13.64% increase in the PSNR as compared to ExtraNet, ExtraSS, Mob-FGSR and GFFE, respectively.

❑ *X4Rate* consistently outperforms interpolation-based algorithms across most scenes. On average, *X4Rate* achieves a PSNR gain of 10.65% over EMA-VFI (Zhang et al., 2023) and 11.98% over softmax-splatting (Niklaus and Liu, 2020).

The paper is organized as follows. Section 2 provides the background and related work in the domain of frame generation. Section 3 shows the characterization of the benchmarks. Section 4 presents the methodology and the implementation details. Section 5 shows the experimental results, and we finally conclude in Section 6. We intend to open-source our project via GitHub in order to encourage its usage and extensions.

2 BACKGROUND AND RELATED WORK

2.1 Interpolation vs Extrapolation

Recent works primarily focus on generating new frames using ① interpolation (Nehab et al., 2007; Herzog et al., 2010; Andreev, 2010) and ② extrapolation (Guo et al., 2021) to increase the frame rate. *In-*

terpolation predicts frames between two already rendered neighboring frames, while *extrapolation* predicts frames solely based on past frames. Both processes double the frame rate by generating a new frame after each rendered frame and display the frame in the following order $0, 0.5, 1, 1.5, 2, ...$ However, interpolation introduces an additional display latency because it buffers an already rendered frame let's say F_1, starts the interpolation, displays the interpolated frame $F_{0.5}$, and then displays frame F_1.

2.2 Meta Learning (MtL) and the Multi-Arm Bandit (MAB) Problem

As mentioned in Section 1, there is no single ML algorithm that can perform well for all the test cases due to inductive bias i.e., the assumptions each model makes to generalize from finite training data. MtL (Pfahringer et al., 2000) addresses this by mapping input characteristics (meta-features) to algorithm performance, and training a meta-learner to recommend suitable algorithms for new inputs. As mentioned in Section 1, our problem directly cannot be considered as a meta-learning problem since along with learning from previous experience (MtL), we need to make decisions in real time under uncertainty (MAB) by using contextual information to choose the best extrapolation algorithm on the fly. The MAB problem involves an agent (decision maker) that iteratively selects one out of multiple available choices (i.e., arms or actions). This happens in an uncertain environment, where the properties of each choice are not fully known at the time of selection and may become better understood as time progresses (Lu et al., 2010). Hence, our problem of selecting the best frame extrapolation algorithm can be modeled as an MAB problem, where each arm represents a different extrapolation algorithm. A variant of MAB, known as Contextual MAB (CMAB), incorporates additional contextual information (e.g., scene properties and Hardware Performance Counters (HPCs)) into the decision-making process (Li et al., 2010). This context informs the agent about the current environment, allowing it to infer which algorithm is likely to perform best under the given conditions. The fundamental challenge in bandit problems is balancing exploration (acquiring new knowledge about the arms) and exploitation (optimizing decisions based on existing knowledge). This trade-off is central to both MAB and broader reinforcement learning paradigms. To address this, various algorithms have been developed, including LinUCB (Li et al., 2010) and epoch-greedy (Langford and Zhang, 2007), which offer heuristic-based approaches to solving the CMAB problem by leveraging contextual information to guide the exploration-exploitation balance effectively (Li et al., 2010; Vermorel and Mohri, 2005).

2.3 State-of-the-Art Frame Extrapolation Algorithms

We present a brief comparison of related work in Table 1. This section discusses the available frame extrapolation algorithms in the literature among which the best one is chosen by the meta-learner. In recent years, frame extrapolation has seen an upsurge since this is the most popular technique used for temporal supersampling when we try to generate frames in real-time (Wu et al., 2024; Yang et al., 2024; Guo et al., 2021; Wu et al., 2023; He et al., 2024; Shimizu et al., 2022; Lee et al., 2018). The prominent works in this area are ExtraNet (Guo et al., 2021), ExtraSS (Wu et al., 2023), Mob-FGSR (Yang et al., 2024) and GFFE (Wu et al., 2024), all of which are considered by our meta-learner when selecting the best extrapolation method.

Each extrapolation method has distinct strengths and limitations (Shimizu et al., 2022; Lee et al., 2018). ExtraNet uses traditional motion-vector (MV)-based warping followed by an inpainting network and auxiliary G-buffers (e.g., depth, normals) to correct shading and fill disoccluded regions. While effective, it often suffers from ghosting artifacts due to inaccuracies in motion vectors. To reduce ghosting, ExtraSS introduces G-buffer-guided warping using a joint bilateral filter similar to the Gaussian filter that improves motion tracking and edge preservation. However, both methods rely on G-buffers, which are expensive to compute. Hence, Mob-FGSR and GFFE avoid UE-generated motion vectors. Mob-FGSR instead uses a lightweight MV reconstruction technique. Though efficient, it ignores shading variations, leading to artifacts in shadows, reflections, and transparent objects. Unlike others, GFFE handles out-of-screen disocclusions using a background-collection module but struggles with novel disocclusions and dynamic lighting changes due to the lack of future frame access and G-buffer information . ExtraNet addresses these issues with a history encoder. No method provides a complete solution; this highlights the need for adaptive algorithm selection. Approaches such as GFFE (Wu et al., 2024) and Mob-FGSR (Yang et al., 2024) are lightweight and can support up to $4\times$ frame rate upsampling. In contrast, ML-based approaches such as ExtraNet (Guo et al., 2021), ExtraSS (Wu et al., 2023) and STSS (He et al., 2024) use heavy neural networks (NNs) hence, their high computational cost and latency make them less

practical for real-time use.

3 CHARACTERIZATION

It is necessary to first analyze the conditions under which each algorithm performs best in terms of visual quality and runtime efficiency. This section provides that analysis, helping us understand the key trade-offs between different approaches. Based on these insights, we then identify the scene-level features that effectively represent these conditions, enabling informed decision-making during algorithm selection.

3.1 Overview of the Benchmarks

Similar to prior work (Guo et al., 2021; Wu et al., 2023), we use ten different applications from the UE (Unreal Engine) marketplace (Mar, 2024) with varying artistic backgrounds and complexities (refer to Table 2). These applications feature scenes with movable objects, dynamic cameras, and in some cases, moving light sources to effectively evaluate the extrapolation of lighting reflections. The experiments are run on an NVIDIA RTX series GPU (Ampere architecture). The detailed configuration is shown in Table 3.

3.2 Performance Trade-Offs in Extrapolation Algorithms

State-of-the-art extrapolation algorithms that we consider in this work are ExtraNet, ExtraSS, Mob-FGSR and GFFE. This section evaluates their performance in terms of quality and runtime, highlighting the key trade-offs between them.

3.2.1 Quality

To evaluate quality, we identify regions where each algorithm struggles or excels and then find the scene attributes that influence this behavior. Our analysis reveals that no single algorithm consistently outperforms the others in all scenarios. The key observations are as follows:

❑ Mob-FGSR struggles with extrapolating shading effects such as shadows, reflections and lighting (refer to Fig.1).

❑ GFFE and Mob-FGSR perform equally well for slow-moving objects, but GFFE handles fast-moving objects better (refer to row 3 in Fig. 1). Therefore, the number of dynamic objects (lights and skeleton meshes) casting shadows should be considered along with their velocities when selecting the appropriate extrapolation algorithm.

❑ Now, if we look at extrapolating the light reflections. It depends upon the type of material and the type of environment (light intensity value). ExtraSS produces proper light reflections for non-metal surfaces (such as wood) whereas ExtraNet works well for extrapolating the specular reflections for metallic surfaces (refer to rows 5-7 in Fig. 1).

❑ GFFE performs better than the other two algorithms when extrapolating areas outside the screen. (refer to row 1 in Fig. 1 where the camera is moving in the left direction). This is because it can fill in large masked regions. It also works well for objects with complex shapes in the Forest (RF) scene since its slight blurriness creates more plausible results (refer to rows 1 and 4 in Fig. 1). These situations can be identified during runtime by observing the camera's velocity (tracking its position over time) and checking for the presence of complex geometries (identified by the number of triangles).

3.2.2 Runtime

Next, we compare the runtime performance. Table 4 shows the runtime breakdown of these extrapolation algorithms. To achieve $4\times$ frame rate upsampling (i.e., 240 fps), each extrapolated frame must be generated within 4.16 ms. Based on this constraint, only Mob-FGSR and GFFE satisfy the latency requirement. In contrast, ExtraSS and ExtraNet, with runtimes exceeding this threshold, are suitable for up to $2\times$ upsampling (120 fps) but fall short for real-time 240 fps generation.

4 METHODOLOGY AND IMPLEMENTATION

4.1 Problem Formulation

In this paper, we aim to select the most suitable frame extrapolation algorithm among ExtraNet, ExtraSS, Mob-FGSR and GFFE, one that provides the highest visual quality at each time instant t. However, our goal is not only to maintain quality but also to increase the frame rate by $4\times$ (from 60 fps to 240 fps). This requires generating three intermediate frames at $t+0.25$, $t+0.5$, and $t+0.75$ between input frames F_t and F_{t+1}. While algorithms such as Mob-FGSR (M) and GFFE (G) can generate all three intermediate frames in time due to their lower runtime, others like ExtraNet (EN) and ExtraSS (ES) are unable to do so (refer to Section 3.2.2). As a result, when the predicted algorithm is EN or ES, we adopt frame du-

Table 1: A comparison of related work.

Year	Work	Type	ML-based	Real-time	*US* Domain	*US* Factor
2020	SS (Niklaus and Liu, 2020)	*I*	✓	×	Temporal	up to 2×
2021	ExtraNet (Guo et al., 2021)	*E*	✓	✓	Temporal	up to 2×
2022	DLSS 3 (Burnes and C Lin, 2023)	*I*	✓	✓	Temporal	2×
2023	EMA-VFI (Zhang et al., 2023)	*I*	✓	×	Temporal	2×
2023	ExtraSS (Wu et al., 2023)	*E*	✓	✓	Spatio-temporal	up to 2×
2024	STSS (He et al., 2024)	*E*	✓	✓	Spatio-temporal	2×
2024	Mob-FGSR (Yang et al., 2024)	*E*	×	✓	Spatio-temporal	2×
2024	GFFE (Wu et al., 2024)	*E*	✓	✓	Spatio-temporal	2×
2025	***X4Rate***	***E***	✓	✓	**Temporal**	**4**×

I: Interpolation, *E*: Extrapolation and *US*: Upsampling

Table 2: List of UE scenes used for all the experiments. The input resolution for all the scenes is 720p.

Abbr.	Name	Abbr.	Name
BK	Bunker	CM	Cemetery
RF	Redwood Forest	BR	Bridge
WT	Western Town	TC	Tennis Court
PR	City Park	TN	Town
DW	Downtown West	VL	Village

Table 3: Platform configuration.

Parameter	Type/Value
CPU	Intel®Xeon®Gold 6226R @ 2.90GHz
GPU	NVIDIA RTX™4080
Game engine	Unreal Engine v5.1

Table 4: Runtime (ms) of state-of-the-art extrapolation methods at 720p resolution.

Methods	ExtraNet	ExtraSS	Mob-FGSR	GFFE
Runtime	8.06	5.63	1.92	3.66

plication. We reuse the last available frame to meet the frame rate constraint (see Table 5).

Table 5: Possible scenarios based on the predicted algorithm at time instant t.

S.	$\mathbf{F_{t+0.25}}$	$\mathbf{F_{t+0.5}}$	$\mathbf{F_{t+0.75}}$
M	$M(F_t)$	$M(M(F_t))$	$M(M(M(F_t)))$
G	$G(F_t)$	$G(G(F_t))$	$G(G(G(F_t)))$
EN	F_t (old frame)	$EN(F_t)$	$EN(F_t)$ (old frame)
ES	F_t (old frame)	$ES(F_t)$	$ES(F_t)$ (old frame)

We broadly divide our implementation into two sub-components. We first identify the input features (meta-features) for the prediction model. We then design multiple predictors and compare their performance for the test data.

4.2 Meta-Features

Extracting meta-features is a crucial and complex part of any meta-learning (MtL) pipeline, as the quality of these features directly impacts the model's accuracy and reliability. Recent papers have identified over 100 meta-features, organized into structured taxonomies (Brazdil et al., 2008; Pfahringer et al., 2000; Rivolli et al., 2022; Bilalli et al., 2017). In this work, we adopt a taxonomy inspired by these surveys and tailor it for real-time frame extrapolation. From a large pool of meta-features, we select a representative subset most relevant to our task (see Table 6) and enhance it with domain-specific features based on empirical insights from Section 3.2.1. Our selected meta-features span six broad categories:

Simple: These features are computationally inexpensive and easily available through APIs. They offer a quick overview of scene complexity. Examples include the number of dynamic objects (N_d), draw calls (N_{dc}), camera velocity (V_c) and shadow-casting lights (N_s). These features significantly affect motion blur and ghosting artifacts.

Statistical: This category includes features that capture statistical properties of scene dynamics and materials. Examples are object motion variance (V) and the average material metallic value (M), which influence perceptual motion complexity and lighting response.

Information-Theoretic: While not computing entropy directly, we use illumination intensity (I) that indirectly reflects pixel unpredictability, aiding in assessing extrapolation difficulty under varying lighting.

Hardware-Based Landmarking: These low-cost proxies estimate scene or system complexity. Features like PCIe and VRAM read/write bandwidth (P_r, P_w, V_r, V_w) and PCIe-to-BAR requests (R) reflect data transfer and CPU-GPU sync overhead. GPU stats like SM warp occupancy (S_o) and compute instructions (C_i) indicate runtime load and extrapolation cost.

Model-Based: To enforce our target of achieving a 4× frame rate boost, we implement latency-aware meta-learner; hence we include the runtime latency (T) of each algorithm in the feature list.

Hence, we represent the meta-features for a frame

Figure 1: Performance of various extrapolation algorithms based on the scene attributes.

F_t using a vector $\mathbf{x}_t$ as follows:

$$\mathbf{x}_t = [N_d, N_s, V, V_c, N_t, M, I, V_r, V_w, P_r, P_w, S_o, C_i, R, T] \quad (1)$$

4.3 Meta-Learner Architecture

Our meta-learning framework implements a Contextual Multi-Armed Bandit (CMAB)-based model to select the optimal frame extrapolation technique in real-time. CMAB-based models are online learners that

Table 6: Features defining the scenes' characteristics.

Feature	Description	Type	Source/API
N_d	#dynamic objects	SP	`TActorIterator<AActor> with Mobility == EComponentMobility::Movable`
N_s	#movable shadow-casting lights	SP	`TActorIterator<ALight> with Mobility == EComponentMobility::Movable and GetLightComponent()->CastShadows`
V	Variance in motion of dynamic objects	ST	`Use GetActorLocation() over time for dynamic actors`
V_c	Camera velocity	SP	`APlayerController::GetPawn()->GetVelocity()`
N_{dc}	#draw calls	SP	`GNumDrawCallsRHI`
M	Average metallic value	ST	`UMaterialInstanceDynamic with GetScalarParameterValue("Metallic")`
I	Total intensity of the Directional Light and Sky Light	IT	`TActorIterator<ADirectionalLight> + AActor::GetLightComponent()->Intensity`
V_r and V_w	VRAM read and write bandwidth (Throughput%)	HL	`NG`
P_r and P_w	PCIe read and write bandwidth (Throughput%)	HL	`NG`
S_o	SM warp occupancy (Throughput%)	HL	`NG`
C_i	Compute (#instructions) in flight (Throughput%)	HL	`NG`
R	PCIe to BAR requests	HL	`NG`
T	Runtime latency	ML	`NG`

NG: Nsight Graphics, **SP**: Simple, **ST**: Statistical, **IT**: Information-theoretic, **HL**: Hardware-based Landmarking, **ML**: Model-based

adapt to runtime scene dynamics through interaction and reward feedback. Specifically, our meta-learner is adapted from the LinUCB algorithm, a contextual bandit method that maintains linear reward estimations for each candidate extrapolation algorithm. Algorithm 1 outlines the working of our decision predictor. At each timestep t, given the current context vector $\mathbf{x}_t$ (Eq. 1), the algorithm evaluates all available extrapolation methods (arms) by computing an upper confidence bound (UCB) for each and provides an estimate of the potential reward. This bound combines the predicted reward, based on historical data, and an uncertainty term $\varepsilon(= 0.05)$ that promotes exploration. Then, the arm with the highest UCB is selected for frame generation. After the extrapolated frame is generated, the actual reward is observed, and the model parameters are updated accordingly to improve future predictions. This adaptive learning loop enables the system to balance quality and latency efficiently by continuously learning which algorithm performs best under varying scene dynamics.

Algorithm 1: Workflow of *X4Rate*.

Input: Context vector $\mathbf{x}_t \in \mathbb{R}^d$, set of available algorithms $\mathcal{A}$, and exploration parameter ε
Output: Selected arm a_t
Initialization:
foreach *arm* a_i **do**
  Initialize $\mathbf{A}_i = \mathbf{I}$ (identity matrix of size $d \times d$) ;
  Initialize $\mathbf{b}_i = \mathbf{0}$ (zero vector of size $d \times 1$) ;
end
for *each time step t* **do**
  foreach *arm* $a_i \in \mathcal{A}$ **do**
    Compute $\hat{\theta}_i = \mathbf{A}_i^{-1}\mathbf{b}_i$;
    Compute predicted reward $\hat{r}_{i,t} = \mathbf{x}_t^\top \hat{\theta}_i$;
    /* No need to run each algorithm to compute the reward */
    Compute confidence bound $\mathrm{CB}_{i,t} = \varepsilon\sqrt{\mathbf{x}_t^\top \mathbf{A}_i^{-1}\mathbf{x}_t}$;
    Compute UCB: $\mathrm{UCB}_{i,t} = \hat{r}_{i,t} + \mathrm{CB}_{i,t}$;
  end
  Select arm $a_t = \arg\max_{a_i \in \mathcal{A}} \mathrm{UCB}_{i,t}$;
  Observe reward r_t for the selected arm a_t;
  Update parameters for the selected arm a_t:

$$\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_t\mathbf{x}_t^\top$$
$$\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t\mathbf{x}_t$$

end

4.3.1 Reward Function

Our reward function is the sum of the following expressions at each decision point. Here, *MSE* refers to the mean square error. We also adopt a latency-aware reward-shaping mechanism in our predictor. This ensures that selections not only maximize visual quality but also adhere to strict real-time performance budgets. Algorithms that exceed the latency threshold are penalized during the decision process, leading to stable and responsive behavior under diverse runtime conditions.

$$\begin{aligned} R &= PSNR + SSIM - \alpha \\ PSNR &= 10 \times log_{10}\left(255^2/MSE\right) \\ SSIM &= \text{Structural similarity between two images} \end{aligned} \tag{2}$$

- The PSNR and SSIM (Niklaus and Liu, 2020) achieved for the chosen *action*.
- The parameter α is the penalty for latency. If the observed latency (t_o) is greater than the predefined latency budget (t_b = 1/240 fps = 4.16ms), then α = t_o - t_b, else there is no penalty.

Table 7: Statistics of the training and testing dataset.

Scenes	Training Sequences	Training Frames	Testing Sequences	Testing Frames
BK	6	3000	2	1000
RF	6	3000	2	1000
WT	6	3000	2	1000
PR	6	3000	2	1000
DW	6	3000	2	1000
CM	6	3000	2	1000
BR	6	3000	2	1000
TC	0	0	2	1000
TN	0	0	2	1000
VL	0	0	2	1000

The action that gives the maximum reward is chosen finally by the predictor.

4.3.2 Implementation Details

All prediction models, including state-of-the-art baselines (ExtraNet, ExtraSS, Mob-FGSR, GFFE), are implemented using the PyTorch framework (v1.13) and trained from scratch. We use the Adam optimizer with standard parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e^{-8}$), a learning rate of 1e-4, batch size of 16, and train for 100 epochs. The total number of frames for training and testing for each scene are specified in Table 7. To ensure fair and robust evaluation across all benchmarks, we adopt the Leave-One-Out Cross-Validation (LOOCV) strategy, where models are trained on all but one benchmark and evaluated on the held-out benchmark, iteratively.

5 RESULTS AND ANALYSIS

① First, we compare the performance of our method *X4Rate* against individual state-of-the-art extrapolation methods. We also evaluate the runtime performance of *X4Rate*.
② Next, we perform an ablation study to analyze the contribution of individual features to the overall effectiveness of *X4Rate*.
③ Subsequently, we compare the prediction accuracy and output quality of various baseline meta-learners with our meta-learner used in *X4Rate*.
④ Finally, to comprehensively explore the solution space for temporal supersampling, we compare *X4Rate* against interpolation-based frame generation methods as well.

For all these experiments, we use the same system configuration (refer to Table 3). Unless otherwise specified, all comparisons are performed at a default spatial resolution of 720p, which is consistent across all experiments. The video results for *X4Rate* are available at the following link (Authors, 2024).

5.1 Comparison with State-of-the-Art Extrapolation Algorithms

In this section, we compare the dynamic algorithm selection capability of *X4Rate* against individual state-of-the-art extrapolation algorithms: ExtraNet (Guo et al., 2021), ExtraSS (Wu et al., 2023), GFFE (Wu et al., 2024) and Mob-FGSR (Yang et al., 2024). Each algorithm is evaluated in isolation using the same benchmark suite. For all the approaches that rely on machine learning, we fine-tune their respective neural networks on our dataset before assessing their performance. We observe that while ML-based methods like GFFE, ExtraNet and ExtraSS achieve high visual fidelity in static or moderately dynamic scenes, they suffer from increased latency in fast-moving scenes or on constrained hardware. The non-ML method Mob-FGSR is computationally lightweight and better suited for real-time execution, but its quality degrades significantly in complex scenarios with disocclusions and shading inconsistencies. In contrast, our method dynamically selects the optimal algorithm for each frame, leading to a better trade-off between quality and efficiency. We make the following observations from the final results (see Table 8).
① Our method *X4Rate* performs better than all existing state-of-the-art extrapolation algorithms across scenes.
② We record a 22.13%, 20.13%, 18.17% and 13.64% increase in the PSNR as compared to ExtraNet, ExtraSS, Mob-FGSR and GFFE, respectively.

5.2 Breakdown of the Predictions

As mentioned earlier, our proposed model, *X4Rate*, predicts the best frame extrapolation method depending upon the current scene attributes. In this section, we show the prediction pattern of our model. The results are shown in Table 9. GFFE emerges as the most frequently selected method overall, followed by Mob-FGSR as the second-best performer.

5.3 Runtime Performance

Table 10 displays the latency overheads for *X4Rate*, which significantly influence the frame rate (FPS) achieved by *X4Rate*. We report the latency for an NVIDIA RTX GPU (please see Table 3 for system details). Given that *X4Rate* dynamically switches between extrapolation models based on scene context, one might expect context switching to add latency. However, we observe that the switching overhead is minimal. This efficiency is due to two factors: (1) all models share similar input formats, and (2) all pre-

Table 8: Quantitative comparison of various state-of-the-art extrapolation methods against *X4Rate* in terms of PSNR (dB) and SSIM. Throughout this paper, the best and second-best results of each test setting are highlighted in bold red and underlined blue, respectively.

Metrics	Scenes	Mob-FGSR	ExtraSS	GFFE	ExtraNet	*X4Rate*
PSNR (dB) ↑	PR	20.30	23.32	24.80	25.82	**27.01**
	BK	21.53	24.43	26.25	20.14	**29.91**
	WT	25.23	27.97	29.25	29.14	**31.24**
	RF	21.15	18.38	18.98	17.47	**21.24**
	CM	28.37	27.21	28.99	26.58	**34.89**
	BR	30.93	24.24	28.02	28.29	**33.25**
	DW	28.01	26.37	28.42	26.78	**32.04**
	TC	23.09	24.17	25.09	21.04	**27.36**
	VL	29.65	25.21	27.38	22.25	**31.07**
	TN	27.65	30.53	29.01	30.14	**34.48**
SSIM ↑	PR	0.852	0.838	0.845	0.854	**0.895**
	BK	0.898	0.876	0.883	0.891	**0.906**
	WT	0.906	0.899	0.901	0.903	**0.921**
	RF	0.801	0.805	0.799	0.811	**0.834**
	CM	0.904	0.907	0.914	0.899	**0.954**
	BR	0.916	0.926	0.940	0.907	**0.952**
	DW	0.898	**0.902**	0.898	0.879	0.901
	TC	**0.945**	0.876	0.885	0.894	0.911
	VL	**0.974**	0.902	0.907	0.896	0.919
	TN	0.943	0.975	0.927	0.955	**0.995**

Table 9: Breakup of the predictions made by the predictor (in %).

Algo.	Mob-FGSR	ExtraSS	GFFE	ExtraNet
PR	05.1	26.5	28.2	**40.2**
BK	18.1	35.2	**36.3**	10.4
WT	12.7	22.0	**35.5**	29.8
RF	**50.4**	18.2	21.1	10.3
CM	24.3	22.0	**28.7**	25.0
BR	**36.4**	15.4	19.1	29.1
DW	33.5	16.7	**36.5**	13.3
TC	27.2	21.7	**34.8**	16.3
VL	**34.5**	23.1	23.6	15.8
TN	07.1	**44.1**	10.6	38.2

trained models (in ONNX format) are pre-loaded in memory. Consequently, context switching contributes negligibly to overall runtime. Additionally, algorithm selection is lightweight, involving only simple matrix operations over the meta-feature vector. Feature extraction is also optimized for speed, further minimizing latency.

Table 10: Average latency (in *ms*) of *X4Rate* at 720p resolution.

Step	Feature collection	Context switching	Algo. selection	Total
Time	1.05	0.05	0.18	1.28

5.4 Ablation Study

In Section 4.2, we introduced the categories of features used in our model. Here, we evaluate the contribution of each category to the predictor's performance. We experiment with three feature combinations: simple scene features (SP), statistical + information-theoretic features (SI), and hardware-level landmarking features (HL). The prediction accuracy for each combination is shown in Figure 2. The average prediction accuracy when considering all the features is 92.42%, which is decreased by 61.69%, 9.48%, and 28.09% when considering only HL, SP,and SI categories, respectively. This demonstrates the importance of incorporating all features for optimal prediction accuracy.

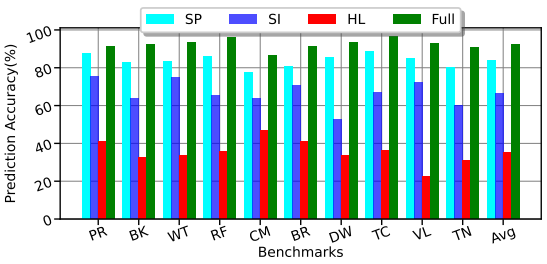

Figure 2: Effect of various feature combinations on the prediction accuracy.

5.5 Performance Comparison against Other Meta-Learners

To comprehensively explore the domain of algorithm selection, we implement three additional meta-learners: an RL model (Sutton et al., 1999), and two supervised models- Random Forest (RF) (Breiman,

2001) and Support Vector Machine (SVM) (Vapnik, 2013). We evaluate their performances and compare against *X4Rate* using four quality metrics: PSNR, SSIM (Hore and Ziou, 2010), LPIPS (Belhe et al., 2023), and accuracy (Aguiar et al., 2019).

In the RL model, we design an agent (neural network: Figure.3) that selects actions based on current features $\mathbf{x_t}$ and a five-dimensional vector $\mathbf{p_t}$ encoding the last five decisions. The combined state $\mathbf{s_t} = (\mathbf{x_t}, \mathbf{p_t})$ is mapped to a four-dimensional reward vector:

$$EW_{\text{net}} : \mathbb{R}^{15} \rightarrow \mathbb{R}^4$$

corresponding to four candidate algorithms: ExtraNet, ExtraSS, Mob-FGSR, and GFFE. The agent chooses the action yielding the highest reward:

$$a_t = \arg \max_{i \in \{1,2,3,4\}} (EW_{\text{net}}(s_t)_i) \quad (3)$$

The loss function we use for the RL agent measures the difference between the predicted reward at time t, $R(\mathbf{s_t}, a_t; \theta_i)$, and the target value, which is the sum of the immediate ground-truth reward r_t and the discounted estimated reward for the next best action, $R(\mathbf{s_{t+1}}, a_{t+1}; \theta_{i-1})$.

$$L_i = \mathbb{E}\left[\left(r_t + \gamma \max_{a_{t+1}} R(\mathbf{s_{t+1}}, a_{t+1}; \theta_{i-1}) - R(\mathbf{s_t}, a_t; \theta_i)\right)^2\right] \quad (4)$$

Where $\gamma = 0.95$ is the discount factor and θ_i are network parameters. The RL network consists of three fully connected ReLU layers (19×128, 128×256, 256×128) followed by an output layer (128×4), as shown in Fig. 3.

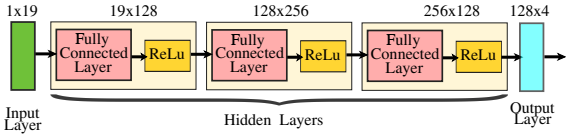

Figure 3: Network architecture of the RL model.

The supervised learning-based models we use are RF and SVM. These models follow different learning biases and have been used with success in multiple predictive tasks. Although RF and SVM are relatively simple models, they serve as strong classical baselines in meta-learning work such as (Verma et al., 2023; Aguiar et al., 2022) due to their robustness and interpretability. Each of these supervised models is trained on a dataset containing meta-features and the ground-truth best algorithm labels. To capture the effect of previous actions, we add one feature to the feature list: the previous decision. We implement a classifier that chooses the best algorithm for each performance metric: PSNR and SSIM, and then use an ensemble model (majority voting) to get the final output.

To enable fair and direct comparison across metrics, we normalize all scores to the range [0,1], ensuring that higher values consistently indicate better performance. Metrics such as PSNR, SSIM and accuracy naturally follow this interpretation, as higher values correspond to higher quality. In contrast, for LPIPS lower values are better; thus, we invert its normalized scores by subtracting from 1. We use a radar chart to present the predictive performances of the meta-learners (refer to Fig. 4). In the radar chart, each line represents a meta-model, and each vertex accounts for a different performance measure. The larger the area in the radar chart, the better the meta-model. From the radar plot, we make the following observations.

① *X4Rate* has the highest accuracy (0.92) hence consistently outperforms the other meta-learners across all other evaluation metrics such as PSNR, SSIM, LPIPS.

② The second best is RL with an accuracy of 0.79. In contrast, SVM and RF (both supervised and offline models) show comparatively lower performance. Their shapes are significantly smaller on the radar, reflecting limited adaptability and suboptimal predictions in dynamic scenarios.

③ Hence, we conclude that our online and contextual model, *X4Rate* is more effective for real-time frame extrapolation tasks, as they adapt quickly to scene variations.

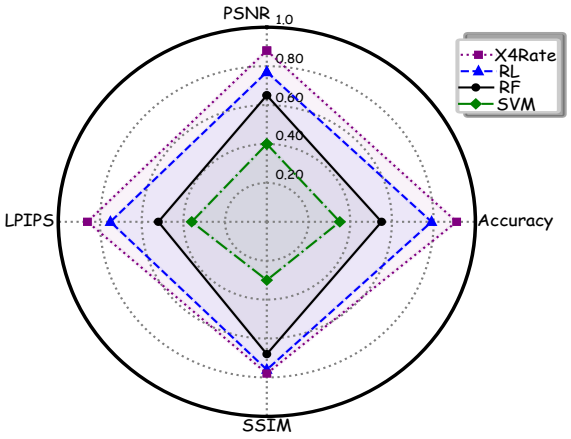

Figure 4: Performance comparison of various meta-learners.

5.6 Comparison with State-of-the-Art Interpolation Algorithms

This section compares the performance of *X4Rate* with two state-of-the-art interpolation-based methods: Softmax Splatting (SS) (Niklaus and Liu, 2020) and EMA-VFI (Zhang et al., 2023). The average interpolation latencies are 8.36 *ms* and 8.12 *ms* for SS

and EMA-VFI, respectively; this does not include the display latency part. Both the solutions upsample the frame rate by 2, hence, to compare their performance with *X4Rate* (4× upsampling), we use the same frame duplication strategy used for ExtraNet and ExtraSS. Due to which, the overall quality is lower that *X4Rate* (refer to Table 11). We record a 10.65%, and 11.98% increase in the PSNR as compared to EMA-VFI and SS, respectively.

Table 11: Quantitative comparison of various interpolation methods against *X4Rate* in terms of PSNR (dB) and SSIM.

Scenes	PSNR (dB) ↑			SSIM ↑		
	EMA-VFI	SS	*X4Rate*	EMA-VFI	SS	*X4Rate*
PR	25.85	24.99	**27.01**	0.857	0.834	**0.895**
BK	27.06	27.21	**29.91**	0.904	0.901	**0.906**
WT	27.58	27.21	**31.24**	0.898	0.895	**0.921**
RF	20.35	21.01	**21.24**	**0.857**	0.842	0.834
CM	29.99	29.01	**34.89**	0.913	0.925	**0.954**
BR	30.55	26.21	**33.25**	0.906	0.911	**0.952**
DW	28.96	27.99	**32.04**	0.900	**0.924**	0.901
TC	26.01	26.54	**27.36**	0.893	**0.901**	0.894
VL	29.25	28.32	**31.07**	**0.924**	0.910	0.919
TN	32.36	31.54	**34.48**	0.991	0.989	**0.995**

5.7 Frame Rate (FPS)

In Table 12, we report the final FPS values achieved by *X4Rate* that gives how many new (extrapolated) frames, we inserted in the original rendering frame sequence. As mentioned previously, in our Unreal Engine (UE) setup, the base rendering rate is capped at 60 FPS via VSync. Moreover, we do not include results from interpolation-based methods, as they operate in an offline setting (where they give 2× upsampling) and are not integrated into UE, making a direct comparison inappropriate. The final FPS correlates with the prediction breakdown across sequences. Higher FPS is achieved when Mob-FGSR or GFFE are selected more frequently, as they can generate all intermediate frames in real time, whereas increased selection of ExtraNet or ExtraSS leads to lower FPS due to frame duplication. However, *X4Rate* consistently achieves a substantial increase in effective frame rate across all sequences, approaching the target 4× upsampling (240 FPS) in several cases.

6 CONCLUSION

With high-frequency displays becoming increasingly popular, generating frames for real-time applications at higher rates is necessary. Since applications are very demanding in terms of processing power, even most GPUs cannot provide a consistently high frame rate at an HD/4k resolution. This work illustrates a temporal supersampling method *X4Rate* which maximizes the quality. Several existing methods use extrapolation to increase the frame rate, but we observed that none of the existing extrapolation algorithms provide high visual quality consistently; consequently, we need to make a dynamic, run-time to choose the best possible algorithm for the current frame. Hence, we designed a predictor to take this decision. We successfully achieved four times the frame rate (≈ 240 Hz) while providing best-in-class visual quality.

Table 12: Final FPS achieved by *X4Rate*.

Algo.	Original FPS	Final FPS
PR	60	199.9
BK	60	205.3
WT	60	198.8
RF	60	232.8
CM	60	200.6
BR	60	200.6
DW	60	226.0
TC	60	214.4
VL	60	211.9
TN	60	191.2

REFERENCES

(2024). Marketplace. https://www.unrealengine.com/marketplace/en-US/store. [Online; accessed 2024-09-23].

(2025). Accessing Unreal Engine source code on GitHub. https://www.unrealengine.com/en-US/ue-on-github. [Online; accessed 2025-03-23].

Adam, S. P., Alexandropoulos, S.-A. N., Pardalos, P. M., and Vrahatis, M. N. (2019). No free lunch theorem: A review. *Approximation and optimization: Algorithms, complexity and applications*, pages 57–82.

Aguiar, G. J., Mantovani, R. G., Mastelini, S. M., De Carvalho, A. C., Campos, G. F., and Junior, S. B. (2019). A meta-learning approach for selecting image segmentation algorithm. *Pattern Recognition Letters*, 128:480–487.

Aguiar, G. J., Santana, E. J., de Carvalho, A. C., and Junior, S. B. (2022). Using meta-learning for multi-target regression. *Information Sciences*, 584:665–684.

Andreev, D. (2010). Real-time frame rate up-conversion for video games: or how to get from 30 to 60 fps for" free". In *SIGGRAPH*, pages 1–1.

Authors, A. (2024). Video results. https://drive.google.com/drive/folders/1CmRhLAqR-wb2b49JTyj4hyvG9m3jcQrA?usp=sharing. [Online; accessed 2024-06-12].

Belhe, Y., Gharbi, M., Fisher, M., Georgiev, I., Ramamoorthi, R., and Li, T.-M. (2023). Discontinuity-aware 2d neural fields. *ACM TOG*, 42(6):1–11.

Bensusan, H. and Kalousis, A. (2001). Estimating the predictive accuracy of a classifier. In *European Conference on Machine Learning*, pages 25–36. Springer.

Bilalli, B., Abelló Gamazo, A., and Aluja Banet, T. (2017). On the predictive power of meta-features in openml. *International Journal of Applied Mathematics and Computer Science*, 27(4):697–712.

Brazdil, P., Carrier, C. G., Soares, C., and Vilalta, R. (2008). *Metalearning: Applications to data mining*. Springer science & business media.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Burnes, A. and C Lin, H. (2023). Nvidia ADA Science. https://images.nvidia.com/aem-dam/Solutions/geforce/ada/ada-lovelace-architecture/nvidia-ada-gpu-science.pdf. [Online; accessed 2023-05-09].

Guo, J., Fu, X., Lin, L., Ma, H., Guo, Y., Liu, S., and Yan, L.-Q. (2021). Extranet: real-time extrapolated rendering for low-latency temporal supersampling. *TOG*, 40(6):1–16.

He, R., Zhou, S., Sun, Y., Cheng, R., Tan, W., and Yan, B. (2024). Low-latency space-time supersampling for real-time rendering. In *Proceedings of the AAAI Conference*, volume 38, pages 2103–2111.

Herzog, R., Eisemann, E., Myszkowski, K., and Seidel, H.-P. (2010). Spatio-temporal upsampling on the gpu. In *I3D*, pages 91–98.

Hore, A. and Ziou, D. (2010). Image quality metrics: Psnr vs. ssim. In *2010 20th international conference on pattern recognition*, pages 2366–2369. IEEE.

Insights, S. M. (2025). Games. https://www.statista.com/outlook/amo/media/games/worldwide. [Online; accessed 2025-01-10].

Langford, J. and Zhang, T. (2007). The epoch-greedy algorithm for contextual multi-armed bandits. *Advances in neural information processing systems*, 20(1):96–1.

Lee, S., Kim, Y., and Eisemann, E. (2018). Iterative depth warping. *ACM TOG*, 37(5):1–13.

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *proc. WWW*, pages 661–670.

Lu, T., Pál, D., and Pál, M. (2010). Contextual multi-armed bandits. In *Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics*, pages 485–492. JMLR Workshop and Conference Proceedings.

Nehab, D., Sander, P. V., Lawrence, J., Tatarchuk, N., and Isidoro, J. R. (2007). Accelerating real-time shading with reverse reprojection caching. In *Graphics hardware*, volume 41, pages 61–62.

Niklaus, S. and Liu, F. (2020). Softmax splatting for video frame interpolation. In *Proc. CVPR*, pages 5437–5446.

Pfahringer, B., Bensusan, H., and Giraud-Carrier, C. G. (2000). Meta-learning by landmarking various learning algorithms. In *ICML*, pages 743–750. Citeseer.

Rice, J. R. (1976). The Algorithm Selection Problem. volume 15 of *Advances in Computers*, pages 65–118. Elsevier.

Rivolli, A., Garcia, L. P., Lorena, A. C., and de Carvalho, A. C. (2021). A study of the correlation of metafeatures used for metalearning. In *International Work-Conference on Artificial Neural Networks*, pages 471–483. Springer.

Rivolli, A., Garcia, L. P., Soares, C., Vanschoren, J., and de Carvalho, A. C. (2022). Meta-features for meta-learning. *Knowledge-Based Systems*, 240:108101.

Shimizu, J., Sun, H., and Katto, J. (2022). Forward and backward warping for optical flow-based frame interpolation. In *ICAIIC 2022*, pages 082–086. IEEE.

Sutton, R. S., Barto, A. G., et al. (1999). Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134.

Vapnik, V. (2013). *The nature of statistical learning theory*. Springer science & business media.

Verma, S., Kumar, P., and Singh, J. P. (2023). A meta-learning framework for recommending cnn models for plant disease identification tasks. *Computers and Electronics in Agriculture*, 207:107708.

Vermorel, J. and Mohri, M. (2005). Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pages 437–448. Springer.

Wolpert, D. H., Macready, W. G., et al. (1995). No free lunch theorems for search. Technical report, Citeseer.

Wu, S., Kim, S., Zeng, Z., Vembar, D., Jha, S., Kaplanyan, A., and Yan, L.-Q. (2023). Extrass: A framework for joint spatial super sampling and frame extrapolation. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–11.

Wu, S., Vembar, D., Sochenov, A., Panneer, S., Kim, S., Kaplanyan, A., and Yan, L.-Q. (2024). Gffe: G-buffer free frame extrapolation for low-latency real-time rendering. *ACM TOG*, 43(6):1–15.

Yang, S., Zhu, Q., Zhuge, J., Qiu, Q., Li, C., Yan, Y., Xu, H., Yan, L.-Q., and Jin, X. (2024). Mob-fgsr: Frame generation and super resolution for mobile real-time rendering. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–11.

Zhang, G., Zhu, Y., Wang, H., Chen, Y., Wu, G., and Wang, L. (2023). Extracting motion and appearance via inter-frame attention for efficient video frame interpolation. In *Proc. CVPR*, pages 5682–5692.