TimeFlowCodec:
A Per-Pixel Temporal Function Video Codec

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Science in Applied Mathematics

by

Rich Chau

University of California, Irvine
December 2025

APPROVAL OF THE THESIS

TimeFlowCodec: A Per-Pixel Temporal Function Video Codec

by

Rich Chau

This thesis has been approved by the following committee:

____

Committee Chair

____

Committee Member

____

Committee Member

Date: ____________________

Abstract

TimeFlowCodec is a video compression framework that models each pixel-channel as a temporal signal and approximates its evolution using simple functions. Instead of relying on block-based prediction, motion estimation, or spatial transforms, TimeFlowCodec introduces a temporal-first design that applies least-squares estimators independently across all pixel-channels. The codec supports three modes—constant, linear, and raw—selected through a rate–distortion–motivated error threshold. This structure produces a compact, mathematically transparent bitstream and enables embarrassingly parallel computation suitable for modern hardware.

The thesis develops a rigorous theoretical foundation for this codec, including operator-theoretic interpretations, analytic expressions for estimator optimality, and rate–distortion analysis. A full reference implementation is specified, along with a formal definition of the `.tfc` bitstream format. Experimental evaluation across UI recordings, slide decks, anime, and natural video demonstrates that TimeFlowCodec achieves substantial compression gains on structured, low-noise digital content while maintaining visual fidelity comparable to or surpassing intra-only classical codecs at similar bitrates.

Although TimeFlowCodec is not designed to compete with general-purpose codecs on natural camera footage, its mathematical clarity, domain specialization, parallelizability, and low implementation complexity suggest promising directions for specialized compression. The thesis concludes by outlining a range of extensions—including higher-order temporal models, segmentation, adaptive quantization, probabilistic modeling, and hybrid neural–analytic architectures—that may expand the applicability and performance of temporal function-based codecs.

Acknowledgments

I would like to express my deepest gratitude to everyone who supported and encouraged me throughout the development of this thesis and throughout my time at the University of California, Irvine.

First, I thank my advisor and committee members for their guidance, feedback, and patience. Their insight into mathematical modeling, signal processing, and research communication helped shape this work into a coherent and meaningful project. I am especially grateful for the freedom they gave me to explore a nontraditional approach to video compression, which ultimately became the basis for TimeFlowCodec.

I would also like to acknowledge the UCI Mathematics Department for fostering an environment where curiosity is encouraged and mathematical thinking is pushed to its limits. The courses, conversations, and mentorship I received during my time here played a central role in developing my research interests and academic direction.

My gratitude extends to my friends and classmates, who made long nights of studying, debugging, and project building not only bearable but genuinely fun. Their support and encouragement—both academic and personal—helped me stay motivated and grounded.

Finally, I thank my family for their unwavering support, love, and belief in my potential. Their sacrifices made my education possible, and their encouragement carried me forward at every stage of this journey. This thesis is dedicated to them.

Contents

Abstract **i**

Acknowledgments **ii**

1 Introduction **1**

1.1 Motivation 2

1.1.1 Shift in Content Types 2

1.1.2 Conceptual Simplicity and Interpretability 3

1.1.3 A New Research Direction 3

1.2 Problem Formulation 4

1.3 Contributions of This Thesis 4

1.4 Scope and Limitations 5

1.5 Thesis Organization 5

2 Background and Related Work **7**

2.1 Overview of Video Compression 7

2.2 Block-Based Coding 8

2.2.1 Block Partitioning 8

2.2.2 Intra Prediction 8

2.2.3 Inter Prediction and Motion Estimation 8

2.3 Transform Coding 9

2.4 Entropy Coding 9

2.5 Neural Video Compression 10

2.6 Temporal Modeling in Video Compression 10

2.7 Functional Approximation and Temporal Smoothness 11

2.8 Gap in the Literature 12

2.9 Summary 12

3 Theoretical Foundations of TimeFlowCodec 13

3.1 Video as a Function in a Tensor Space 13

3.2 Temporal Function Models 14

3.3 Least-Squares Estimators 15

3.4 Modeling Error and Distortion 16

3.5 Rate–Distortion Framework 16

3.6 TimeFlowCodec as a Separable Operator 17

3.7 Projection Interpretation 18

3.8 Computational Complexity 18

3.9 Summary 19

4 File Format and Bitstream Specification 20

4.1 Bitstream Overview 20

4.2 Header Specification 21

4.3 Mode Table 22

4.3.1 Decoding Procedure 22

4.4 Parameter Streams . . . 23
4.4.1 Constant Parameter Stream . . . 23
4.4.2 Linear Parameter Stream . . . 23
4.4.3 Raw Stream . . . 24
4.5 Byte-Level Ordering . . . 24
4.6 Error Resilience . . . 24
4.7 Deterministic Decoding . . . 25
4.8 Bitstream Example . . . 25
4.9 Summary . . . 26

5 Reference Implementation of TimeFlowCodec 27

5.1 Design Objectives . . . 27
5.2 Encoder Architecture . . . 28
5.2.1 Frame Extraction . . . 29
5.2.2 Temporal Signal Assembly . . . 30
5.3 Temporal Model Fitting . . . 30
5.4 Mode Selection . . . 31
5.5 Parameter Stream Construction . . . 31
5.5.1 Constant Parameters . . . 31
5.5.2 Linear Parameters . . . 32
5.5.3 Raw Samples . . . 32
5.6 Serialization and Bitstream Construction . . . 32
5.7 Encoder Pseudocode . . . 33
5.8 Decoder Architecture . . . 33

5.9 Decoding Process 34

5.10 Decoder Pseudocode 35

5.11 Parallelization Strategy 35

5.12 Memory and Cache Considerations 36

5.13 Summary 36

6 Experimental Evaluation **37**

6.1 Datasets 37

6.2 Metrics 37

6.3 Baselines 37

6.4 Rate–Distortion Curves 38

6.5 Quantitative Results 39

6.6 Runtime Analysis 40

6.7 Discussion 40

6.8 Summary 40

7 Analysis and Discussion **41**

7.1 Theoretical Expectations vs. Experimental Outcomes 41

7.2 Mode Distribution as a Predictor of Compression Efficiency 42

7.3 Error Characteristics and Perceptual Considerations 43

7.4 Temporal Model Suitability by Content Type 44

7.4.1 UI Videos 44

7.4.2 Slide Deck Videos 44

7.4.3 Anime Videos 45

7.4.4 Natural Video 45

7.5 Computational Complexity Analysis . . . 45
7.6 Comparative Advantages . . . 46
7.7 Limitations . . . 47
7.8 Interpretation in Rate–Distortion Space . . . 47
7.9 Summary . . . 47

8 Future Work and Extensions 49

8.1 Architectural Extensions . . . 49
8.1.1 Higher-Order Temporal Models . . . 49
8.1.2 Temporal Segmentation . . . 50
8.1.3 Advanced Quantization . . . 50
8.1.4 Entropy Coding Enhancements . . . 51
8.2 Mathematical Generalizations . . . 51
8.2.1 Operator-Based Temporal Modeling . . . 51
8.2.2 Nonlinear Temporal Models . . . 51
8.2.3 Probabilistic Temporal Modeling . . . 52
8.2.4 Hybrid Spatial–Temporal Approximation . . . 52
8.3 Hybrid Codec Architectures . . . 53
8.3.1 Integration with Block-Based Codecs . . . 53
8.3.2 Integration with Neural Codecs . . . 53
8.3.3 Temporal Residual Codec . . . 53
8.4 GPU and Hardware Acceleration . . . 54
8.5 Robustness to Noise and Natural Video . . . 54
8.6 Dynamic Model Selection . . . 54

8.7 Auto-Segmentation Using Change Detection 55

8.8 Adaptive Threshold Selection 55

8.9 Extending TFC to YUV Color Spaces 55

8.10 Summary 55

9 Conclusion **57**

9.1 Summary of Contributions 57

9.2 Broader Implications 58

9.3 Limitations 59

9.4 Future Outlook 59

9.5 Final Remarks 60

A Algorithms and Pseudocode **61**

A.1 Notation 61

A.2 Temporal Signal Extraction 61

A.3 Least-Squares Fitting Algorithms 62

A.4 Mode Selection Algorithm 63

A.5 Bitstream Serialization Algorithms 63

A.6 Decoder Reconstruction Algorithms 64

A.7 Summary 65

B Mathematical Proofs **66**

B.1 Proof of Least-Squares Optimality 66

B.2 Proof of Uniqueness of Linear Fit 67

B.3 Proof: Projection Interpretation 67

B.4 Rate–Distortion Lagrangian Optimality 68

B.5 Operator-Theoretic Result 69

B.6 Summary 69

C Supplemental Experimental Data **70**

C.1 Detailed Mode Distributions by Video 70

C.2 Bitrate Decomposition 71

C.3 Runtime Breakdown 71

C.4 Supplemental RD Curves 71

C.5 Summary 73

List of Figures

5.1 High-level architecture of the TimeFlowCodec encoder. 29

5.2 High-level architecture of the TimeFlowCodec decoder. 34

6.1 Rate–distortion curve for UI dataset. 38

6.2 Rate–distortion curve for slide deck dataset. 38

6.3 Rate–distortion curve for anime dataset. 39

6.4 Rate–distortion curve for natural dataset. 39

C.1 Supplementary RD plot for UI dataset. 72

C.2 Supplementary RD plot for slide deck dataset. 72

C.3 Supplementary RD plot for anime dataset. 73

C.4 Supplementary RD plot for natural dataset. 73

List of Tables

4.1 Header fields in a `.tfc` file. 21

7.1 Expected vs. observed performance of TFC on different content types. 42

C.1 Fine-grained mode distribution of all test clips. 70

C.2 Bitrate composition across datasets. 71

C.3 Encoder runtime distribution. 71

Chapter 1

Introduction

Digital video has become the dominant medium for communication, technical instruction, entertainment, and interactive computing. The rapid shift toward high–frame rate, high–resolution video—combined with the ubiquity of video conferencing, streaming, online education platforms, and real-time collaborative tools—has intensified the demand for efficient and flexible video compression methods. While state-of-the-art codecs such as H.264/AVC, H.265/HEVC, VP9, and AV1 deliver remarkable performance on natural camera footage, they achieve this efficiency through a deep stack of carefully engineered mechanisms: block-based prediction, motion estimation, transform coding, in-loop filtering, quantization control, probability modeling, and entropy coding.

These codecs are optimized for the statistics of natural scenes, where motion is rich, textures are irregular, and photometric noise is unavoidable. However, the modern computing ecosystem increasingly produces video content that violates these assumptions. Screen recordings, graphical user interface (GUI) animations, digital lecture slides, terminal captures, 2D vector animations, and anime-style content all exhibit fundamentally different characteristics: low noise, large spatial regions with constant color, and highly structured temporal changes. These properties suggest that the classical tools of block prediction and transform coding might not be the most natural representation for such content.

TimeFlowCodec—the subject of this thesis—revisits a basic question:

> *If we model a video not spatially, but temporally—one pixel-channel at a time—how far can simple mathematical models go in compressing modern, structure-rich content?*

Instead of modeling blocks or motion fields, TimeFlowCodec treats each pixel-channel as a separate one-dimensional temporal signal. For a video with resolution $H \times W$ and T frames, this means fitting temporal functions to each of the $3HW$ sequences

$$S_{y,x,c}(t), \qquad t = 0, \ldots, T-1.$$

These sequences often exhibit long stretches of constancy or monotonic change in UI animations, scroll events, fades, and stylized animation. TimeFlowCodec exploits this structure by selecting between simple temporal models—constant, linear, or raw—in a lossy rate–distortion framework. The result is a codec with low conceptual complexity, high interpretability, and impressive performance on certain modern video domains.

This chapter introduces the motivation, scope, and research contributions of the thesis and situates TimeFlowCodec in the broader landscape of video compression research.

1.1 Motivation

The motivation for TimeFlowCodec arises from both practical observations and theoretical considerations.

1.1.1 Shift in Content Types

Historically, video compression research has centered on natural camera footage. However, several fast-growing categories of digital content are now dominant:

- video lectures recorded from slide decks or iPad annotations,
- screen recordings of programming environments or terminals,
- GUI interaction videos used in product tutorials,
- gameplay recordings from 2D or stylized games,
- anime and other animation genres with large flat-color regions,
- whiteboard-style mathematical or engineering demonstrations.

Such content differs fundamentally from natural video. Noise is essentially absent. Motion is often functionally defined (e.g., easing curves, fade transitions) rather than physically driven. Many pixel-channels evolve in smooth, low-dimensional patterns that traditional codecs do not explicitly exploit.

1.1.2 Conceptual Simplicity and Interpretability

Modern codecs are complex. Even expert engineers require extensive effort to understand the full structure of HEVC or AV1. In contrast, TimeFlowCodec is based on simple mathematical components:

- least-squares fitting,
- temporal function approximation,
- pixelwise independence,
- compact bitstream representation.

These choices make the codec easier to analyze, easier to modify, and easier to teach than state-of-the-art encoders. Because every pixel-channel is treated independently, the codec is also inherently parallelizable—ideal for systems with large GPU compute capability.

1.1.3 A New Research Direction

The central idea explored in this thesis—pixelwise temporal modeling—has received relatively little attention in mainstream video compression research. By isolating the temporal dimension and compressing each temporal signal independently, we obtain:

- a clean mathematical framework,
- a finite-dimensional model selection problem,
- tractable rate–distortion analysis,
- and a foundation for hybrid codecs that blend temporal modeling with spatial or neural components.

This makes TimeFlowCodec not only a working prototype but also a platform for future exploration.

1.2 Problem Formulation

Let a video be represented as a tensor

$$V \in \mathbb{R}^{H \times W \times T \times 3},$$

where each pixel-channel evolves according to an unknown temporal law. Traditional codecs operate by spatial partitioning and cross-frame alignment. TimeFlowCodec, in contrast, considers the temporal sequences

$$S_k(t) = V(y, x, t, c), \qquad k = (y, x, c),$$

as the fundamental objects of compression.

The core problem addressed in this thesis is:

> *Given a temporal signal $S_k(t)$, how can we represent it using the smallest number of bits while preserving visual fidelity under a lossy distortion metric?*

TimeFlowCodec answers this by selecting among simple temporal models: constant, linear, or raw storage. A normalized error metric determines the optimal mode under a rate–distortion framework.

1.3 Contributions of This Thesis

The thesis makes the following contributions:

1. **A novel compression paradigm.** TimeFlowCodec introduces per-pixel temporal modeling as a viable strategy for compressing structured digital content. Unlike block-based or learned codecs, the approach is grounded in classical least-squares approximation.

2. **A rigorous mathematical foundation.** Chapter 3 develops a comprehensive theoretical framework, including projection operators, model fitting theory, rate–distortion analysis, and operator-theoretic interpretations of the codec.

3. **A complete bitstream specification.** The `.tfc` format is defined precisely, including header, mode table, temporal model payload, and optional entropy coding layers.

4. **A functioning reference implementation.** A Python encoder and decoder demonstrate the feasibility of the codec and provide a baseline for future optimization.

5. **A thorough experimental study.** TimeFlowCodec is evaluated on UI recordings, slide decks, anime content, and natural camera footage. The experiments reveal both the strengths and limitations of the method, including performance regions where it outperforms intra-only baselines.

6. **Extensions and research pathways.** The thesis identifies numerous directions for future research, including nonlinear modeling, spatiotemporal hybridization, noise robustness, and hardware acceleration.

1.4 Scope and Limitations

TimeFlowCodec is not intended to replace mainstream codecs for natural video. Instead, it targets domains where temporal structures dominate and spatial texture is simple. This thesis therefore focuses on a well-defined content subset. Nonetheless, the methodological insights extend broadly and may inform future codec designs.

1.5 Thesis Organization

The remainder of the thesis is structured as follows:

- **Chapter 2** surveys classical and modern video compression techniques, focusing on temporal prediction and transform coding.
- **Chapter 3** establishes the mathematical theory underlying TimeFlowCodec, from least-squares temporal modeling to rate–distortion analysis.

- **Chapter 4** describes the `.tfc` file format and bitstream structure.
- **Chapter 5** details the reference implementation.
- **Chapter 6** presents the experimental evaluation across diverse datasets.
- **Chapter 7** analyzes the strengths and limitations of temporal function modeling.
- **Chapter 8** proposes extensions to the codec and outlines future research directions.
- **Chapter 9** concludes with a summary and broader context.

TimeFlowCodec, while simple, demonstrates that even classical mathematical techniques can inspire new compression strategies in an era dominated by neural and transform-based approaches. This chapter sets the stage for exploring those ideas in depth.

Chapter 2

Background and Related Work

Video compression has been one of the most technically demanding and mathematically rich areas of signal processing for over three decades. The dominant paradigm across nearly all modern codecs has been block-based prediction combined with transform coding. This chapter introduces the principles underlying these classical methods, reviews recent neural approaches, and identifies gaps in the literature that motivate the design of TimeFlowCodec. The chapter concludes with a comparison between temporal function modeling and the broader family of predictive video compression approaches.

2.1 Overview of Video Compression

At a high level, all video codecs aim to represent a sequence of frames

$$V_0, V_1, \ldots, V_{T-1}$$

using as few bits as possible while maintaining acceptable visual quality. A video frame can be regarded as a function

$$V_t : \{0, \ldots, H-1\} \times \{0, \ldots, W-1\} \to \mathbb{R}^3.$$

Since raw video contains enormous redundancy, the central objective of video compression is to exploit structural patterns in both space and time. The dominant techniques for doing so are *spatial transforms*, *temporal prediction*, and *entropy coding*. These ideas define the architecture of codecs such as H.264/AVC [2], HEVC, VP9, and AV1 [1].

TimeFlowCodec departs from this classical structure by avoiding spatial partitioning and motion estimation entirely. To contextualize this departure, the next sections review the main building blocks of conventional codecs.

2.2 Block-Based Coding

2.2.1 Block Partitioning

Most video codecs divide each frame into non-overlapping blocks (e.g., 16×16, 8×8, or variable-sized units). These blocks serve as the fundamental elements for prediction, transform coding, filtering, and entropy coding.

Let $B_t^{(i)}$ denote the i-th block of frame V_t. The encoder attempts to predict $B_t^{(i)}$ from previously encoded data. The difference (residual) between the block and its prediction is then compressed efficiently.

2.2.2 Intra Prediction

Intra prediction exploits spatial redundancy within a single frame. Typical modes include directional prediction along edges, planar prediction, or DC prediction. In block $B_t^{(i)}$, values along the top or left boundary are used to estimate pixels inside the block:

$$\hat{B}_t^{(i)}(y,x) = P\big(B_t^{(i)}(y',x')\big),$$

for some predictor function P.

TimeFlowCodec does not use intra prediction. Instead, it treats spatial neighbors as irrelevant and focuses exclusively on temporal coherence.

2.2.3 Inter Prediction and Motion Estimation

Temporal prediction has been the cornerstone of video coding since MPEG-1. The key assumption is that consecutive frames differ primarily by motion. If a block in one frame corresponds to a shifted version of a block in a reference frame, it can be represented compactly by a motion vector and a residual:

$$B_t^{(i)} \approx B_{t'}^{(j)} + \text{residual}.$$

Motion estimation seeks the displacement vector (u, v) that minimizes

$$\|B_t^{(i)} - B_{t-1}^{(i)}(\cdot - (u, v))\|^2.$$

Successive generations of codecs have refined this approach using subpixel interpolation, multiple reference frames, bi-directional prediction, and in-loop filters.

TimeFlowCodec does not perform motion estimation. Instead, it directly models the temporal evolution of each pixel-channel, avoiding the combinatorial search inherent in inter prediction.

2.3 Transform Coding

After prediction, codecs transform the residual signal into a domain where energy is concentrated in a few coefficients. The most common transform is the Discrete Cosine Transform (DCT), though AV1 and other codecs use variants or multi-directional transforms.

Let $R^{(i)}$ denote a residual block. The transform yields coefficients

$$C = \mathcal{T}(R^{(i)}).$$

Once quantized, these coefficients are entropy-coded. This pipeline is crucial for compressing textured regions in natural video.

TimeFlowCodec does not use transforms. Instead, it relies on low-dimensional temporal function models that serve as an implicit transform along the time axis.

2.4 Entropy Coding

Entropy coding assigns shorter bit sequences to frequent symbols and longer ones to rare symbols, ensuring near-optimal compression under a probabilistic model. Methods include Huffman coding, arithmetic coding, and context-adaptive binary arithmetic coding (CABAC). These tools are responsible for a large fraction of codec performance.

TimeFlowCodec employs an optional entropy layer (e.g., zlib) applied after the mode table and parameter streams are assembled. This maintains simplicity while offering additional compression.

2.5 Neural Video Compression

Recent progress in deep learning has produced neural image and video codecs that compete with or surpass traditional methods. Neural approaches often involve:

- learned autoencoders for spatial transforms,
- recurrent or attention-based networks for temporal prediction,
- learned entropy models for probability distribution estimation,
- end-to-end rate–distortion optimization.

Neural codecs excel in capturing complex textures and motion. However, they are computationally intensive, require significant training data, and operate as black-box systems. Interpretability is limited, and replacing individual components is difficult.

In contrast, TimeFlowCodec offers complete transparency: each pixel-channel is encoded using interpretable mathematical functions. The trade-off is clear: neural codecs target generality and performance, whereas TimeFlowCodec targets simplicity, structure-rich domains, and analytical tractability.

2.6 Temporal Modeling in Video Compression

Although motion compensation is the dominant temporal prediction tool, several alternative temporal modeling approaches have been explored:

- **Polynomial extrapolation** for frame interpolation,

- **Spline-based motion fields** for smooth motion,
- **Linear dynamical systems** for video forecasting,
- **Optical flow models** to approximate velocity fields,
- **Temporal filtering** for video decomposition.

However, these methods have rarely been used as the *primary* compression mechanism. Motion estimation remains the central predictive tool due to its effectiveness on natural scenes.

TimeFlowCodec departs from this trend by applying temporal modeling independently to each pixel-channel, rather than to regions or motion fields. This design is justified by the specific characteristics of UI recordings and stylized content, where temporal coherence at the pixel level is strong.

2.7 Functional Approximation and Temporal Smoothness

TimeFlowCodec is grounded in the idea that many pixel trajectories exhibit low-frequency temporal patterns. Classical results from approximation theory suggest that functions with bounded variation or limited curvature can be modeled effectively using low-degree polynomials.

Least-squares fitting provides the optimal linear model parameters under a quadratic distortion metric. When the underlying signal is smooth or piecewise smooth, linear models achieve small approximation error.

The use of constant and linear functions in TimeFlowCodec is therefore not arbitrary: these models correspond to the simplest non-trivial approximations in a hierarchical function space. Extending the codec to higher-order models is conceptually straightforward but introduces additional bits for parameters and greater risk of overfitting.

2.8 Gap in the Literature

To the best of our knowledge, no mainstream or experimental codec has adopted a purely pixelwise temporal modeling approach. While temporal transforms exist in 3D wavelet codecs and neural approaches sometimes incorporate temporal layers, the idea of fitting independent parametric models to each pixel sequence remains unexplored. The lack of such research stems from two factors:

1. Natural video contains noise and motion that render pixelwise fitting unreliable.
2. Spatial modeling is generally more effective for scenes with complex textures.

However, the rise of structured digital content reopens this design space. TimeFlowCodec targets precisely these modern scenarios.

2.9 Summary

This chapter situated TimeFlowCodec within the broader landscape of video compression research. Traditional codecs rely on block-based transforms and motion estimation, while neural codecs leverage deep networks for prediction and entropy modeling. Temporal modeling exists as a component of these systems but has never formed the core of a codec design.

The next chapter develops the theoretical foundations of TimeFlowCodec, including least-squares temporal modeling, rate–distortion analysis, operator-theoretic interpretations, and computational complexity.

Chapter 3

Theoretical Foundations of TimeFlowCodec

This chapter develops the mathematical foundations upon which TimeFlowCodec is built. We begin by defining the video domain as a high-dimensional function space, then introduce the pixelwise temporal modeling framework that enables TimeFlowCodec to approximate complex video tensors using low-dimensional functions. We derive closed-form estimators for temporal model parameters, establish rate–distortion relationships under a lossy framework, and interpret TimeFlowCodec as a separable operator acting on $\mathbb{R}^{H\times W\times T\times 3}$. We conclude with a computational complexity analysis that explains the scalability and parallelizability of the codec.

3.1 Video as a Function in a Tensor Space

Let a video V be represented as a discrete spatiotemporal function

$$V : \{0, \dots, H-1\} \times \{0, \dots, W-1\} \times \{0, \dots, T-1\} \times \{R, G, B\} \to \mathbb{R}.$$

Equivalently, V may be regarded as a four-dimensional tensor

$$V \in \mathbb{R}^{H\times W\times T\times 3}.$$

For convenience, we index pixel-channels using a single index

$$k = (y, x, c) \in \mathcal{K}, \qquad |\mathcal{K}| = 3HW.$$

For each k, the corresponding temporal signal is the sequence

$$S_k(t) = V(y, x, t, c), \qquad t = 0, \dots, T-1.$$

TimeFlowCodec compresses V by approximating each temporal sequence S_k using a function from a low-dimensional model family $\mathcal{F}$.

3.2 Temporal Function Models

Let $\mathcal{F}$ be the set of temporal models considered by TimeFlowCodec:

$$\mathcal{F} = \{f(t) = a\} \cup \{f(t) = a + bt\} \cup \{\text{RAW representation of } S_k(t)\}.$$

The constant and linear models correspond to the subspaces

$$\mathcal{F}_0 = \{f : f(t) = a\}, \qquad \mathcal{F}_1 = \{f : f(t) = a + bt\},$$

each parameterized by a small number of coefficients. They are the simplest nontrivial elements of the polynomial function hierarchy.

Given a distortion metric $D(\cdot, \cdot)$ and a rate model $R(\cdot)$, the goal of the codec is to choose for each pixel-channel k the model

$$f_k^* = \arg\min_{f \in \mathcal{F}} \big[D(S_k, f) + \lambda R(f)\big],$$

for some Lagrange multiplier $\lambda > 0$ that trades off fidelity and bitrate.

3.3 Least-Squares Estimators

TimeFlowCodec uses least-squares fitting as the metric for determining the best constant or linear approximation to a temporal sequence. For

$$S_k(t), \qquad t = 0, \ldots, T-1,$$

we define the constant estimator

$$a^* = \frac{1}{T} \sum_{t=0}^{T-1} S_k(t),$$

and the linear estimator (a^*, b^*) as the solution to

$$\min_{a,b} \sum_{t=0}^{T-1} (S_k(t) - a - bt)^2.$$

Let

$$S_0 = \sum_{t=0}^{T-1} 1 = T, \qquad S_1 = \sum_{t=0}^{T-1} t, \qquad S_2 = \sum_{t=0}^{T-1} t^2,$$

$$Y_0 = \sum_{t=0}^{T-1} S_k(t), \qquad Y_1 = \sum_{t=0}^{T-1} t S_k(t).$$

The normal equations are:

$$\begin{pmatrix} T & S_1 \\ S_1 & S_2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} Y_0 \\ Y_1 \end{pmatrix}.$$

The determinant

$$\Delta = T S_2 - S_1^2$$

is always positive for $T \geq 2$, ensuring the system has a unique solution. Thus,

$$a^* = \frac{S_2 Y_0 - S_1 Y_1}{\Delta}, \qquad b^* = \frac{T Y_1 - S_1 Y_0}{\Delta}.$$

These formulas allow the encoder to compute model parameters in $O(T)$ time per pixel-channel.

3.4 Modeling Error and Distortion

TimeFlowCodec evaluates the modeling error using the mean squared error (MSE):

$$D_{\mathrm{LS}}(S_k, f) = \frac{1}{T}\sum_{t=0}^{T-1}(S_k(t) - f(t))^2.$$

To make the error scale-free, the codec normalizes the distortion by the signal energy:

$$r_k(f) = \frac{D_{\mathrm{LS}}(S_k, f)}{\frac{1}{T}\sum_{t=0}^{T-1} S_k(t)^2}.$$

A model f is accepted if $r_k(f) \leq \tau$ for a fixed threshold τ.

In lossy compression, we select the model that minimizes the Lagrangian cost

$$J_k(f) = D_{\mathrm{LS}}(S_k, f) + \lambda R(f).$$

Constant and linear models incur lower rates (few bits), while RAW mode incurs higher rates but zero distortion.

3.5 Rate–Distortion Framework

For a single pixel-channel, let:

$$R_0 = \text{bits required for constant model,}$$

$$R_1 = \text{bits required for linear model,}$$

$$R_R = \text{bits for raw samples} = 8T.$$

Let the distortions be D_0, D_1, and 0 respectively. The RD-optimal mode minimizes:

$$\min\bigl(D_0 + \lambda R_0,\ D_1 + \lambda R_1,\ \lambda R_R\bigr).$$

Because $R_0 < R_1 \ll R_R$, the constant model is preferred when:

$$D_0 + \lambda R_0 < D_1 + \lambda R_1, \quad D_0 + \lambda R_0 < \lambda R_R.$$

These inequalities define regions in (D_0, D_1) space where each model is optimal. Crucially, if a pixel-channel's temporal trajectory is smooth, D_0 and D_1 are small, so the codec chooses constant or linear models with high probability.

3.6 TimeFlowCodec as a Separable Operator

Define the operator $\mathcal{T}$ that maps a temporal signal $S_k(t)$ to its best model $f_k^*(t)$. TimeFlow-Codec acts on the video tensor as:

$$\mathcal{C}(V) = \big(\mathcal{T}(S_k)\big)_{k \in \mathcal{K}}.$$

Since each $\mathcal{T}(S_k)$ depends only on the temporal dimension of that specific pixel-channel, $\mathcal{C}$ is *separable* across pixel-channels:

$$\mathcal{C} = \mathcal{T}^{\otimes(3HW)}.$$

This separability has two key implications:

1. **Massive parallelizability.** The codec can be implemented using GPU kernels that process millions of pixel-channels independently.
2. **Simplified analysis.** The RD behavior of the codec can be studied by analyzing a single pixel-channel and aggregating results across the video tensor.

This operator-theoretic view highlights the elegance and modularity of TimeFlowCodec.

3.7 Projection Interpretation

The least-squares estimate for the constant model is the orthogonal projection of the temporal vector

$$S_k \in \mathbb{R}^T$$

onto the subspace

$$\mathcal{V}_0 = \{a \cdot \mathbf{1}\}.$$

Similarly, the linear model is the projection onto

$$\mathcal{V}_1 = \operatorname{span}\{\mathbf{1}, t\}.$$

Thus, TimeFlowCodec replaces each temporal vector with its projection onto one of three sets: $\mathcal{V}_0$, $\mathcal{V}_1$, or the raw space (identity). Mathematically:

$$\mathcal{T}(S_k) = \begin{cases} P_{\mathcal{V}_0} S_k, & \text{if constant model selected,} \\ P_{\mathcal{V}_1} S_k, & \text{if linear model selected,} \\ S_k, & \text{otherwise.} \end{cases}$$

This interpretation will be crucial when we analyze distortions and compression ratios in later chapters.

3.8 Computational Complexity

TimeFlowCodec must compute for each of the $3HW$ pixel-channels:

1. the constant estimator (cost $O(T)$),
2. the linear estimator (cost $O(T)$),
3. the modeling errors (cost $O(T)$),
4. the mode selection (cost $O(1)$).

Thus the cost of encoding is:

$$O(3HWT).$$

Decoding is even simpler, costing:

$$O(3HWT)$$

for reconstructing values from model parameters.

Unlike traditional codecs whose complexity grows with block search spaces and motion estimation grids, TimeFlowCodec scales linearly. Moreover, because each pixel-channel can be processed independently, the codec maps naturally to parallel computation architectures.

3.9 Summary

This chapter established the theoretical foundation of TimeFlowCodec. We defined the underlying function spaces, derived least-squares estimators, developed a rate–distortion model, and interpreted the codec as a separable operator acting on the video tensor. These concepts unify the practical implementation with a strong mathematical theory, enabling rigorous analysis in later chapters.

The next chapter describes the `.tfc` file format and bitstream structure used by TimeFlowCodec.

Chapter 4

File Format and Bitstream Specification

This chapter defines the `.tfc` bitstream format used by TimeFlowCodec. The purpose of this specification is to provide an unambiguous, complete, and implementation-independent description of how encoded videos are represented in binary form. The design emphasizes clarity, simplicity, and deterministic decoding.

The `.tfc` file format consists of four major components:

1. a fixed-size header containing metadata,
2. a mode table specifying the temporal model chosen for each pixel-channel,
3. a set of parameter streams for constant, linear, and raw modes,
4. an optional final-stage entropy coding layer.

The bitstream is designed such that decoding is a purely sequential process: read the header, decode the mode table, read parameters from the corresponding streams, and reconstruct pixel trajectories.

4.1 Bitstream Overview

The high-level structure of a `.tfc` file is:

$$\texttt{File} = \underbrace{\texttt{Header}}_{\text{fixed size}} \parallel \underbrace{\texttt{ModeTable}}_{\text{bit-packed}} \parallel \underbrace{\texttt{ConstParamStream}}_{\text{16-bit values}} \parallel \underbrace{\texttt{LinearParamStream}}_{(a,b)\text{ pairs}} \parallel \underbrace{\texttt{RawStream}}_{\text{byte sequences}}$$

Optionally, the region beginning with `ModeTable` to the end of the file may be wrapped in a general-purpose entropy coder such as `zlib`, `lz4`, or `brotli`.

4.2 Header Specification

The header is 64 bytes and always uncompressed. Table 4.1 lists all fields in order.

Offset	**Field**	**Description**
0–3	Magic Number	UTF-8 string “TFC1”
4–7	Version	32-bit unsigned integer, currently 1
8–11	Width W	32-bit unsigned integer
12–15	Height H	32-bit unsigned integer
16–19	Frames T	32-bit unsigned integer
20–23	Framerate	32-bit unsigned integer (fps $\times$ 1000)
24–31	Reserved	for future extensions
32–39	OffsetModeTable	64-bit unsigned integer
40–47	OffsetConstStream	64-bit unsigned integer
48–55	OffsetLinearStream	64-bit unsigned integer
56–63	OffsetRawStream	64-bit unsigned integer

Table 4.1: Header fields in a `.tfc` file.

All integers are stored in little-endian format.

Offsets refer to absolute byte positions within the file, allowing decoders to seek directly to each stream without processing intermediate data.

4.3 Mode Table

The mode table contains one entry per pixel-channel. Since there are $3HW$ pixel-channels, the mode table has exactly $3HW$ entries.

Each entry encodes one of three possible modes:

$$\texttt{00} = \textsc{Const}, \qquad \texttt{01} = \textsc{Linear}, \qquad \texttt{10} = \textsc{Raw}.$$

The remaining pattern `11` is reserved.

Because each entry requires exactly 2 bits, the total size of the mode table is:

$$\left\lceil \frac{3HW \cdot 2}{8} \right\rceil \text{ bytes.}$$

The mode table is bit-packed in row-major pixel order. Let the pixel index be:

$$k = (y \cdot W + x) \cdot 3 + c.$$

The mode for pixel-channel k appears at bit-position:

$$b_k = 2k.$$

4.3.1 Decoding Procedure

To decode the kth entry:

1. Compute byte index $i = \lfloor b_k/8 \rfloor$.
2. Compute bit offset $o = b_k \bmod 8$.
3. Extract the 2-bit code:

$$m_k = (\texttt{ModeTable}[i] \gg o) \,\&\, 0b11.$$

This deterministic procedure avoids alignment concerns.

4.4 Parameter Streams

After the mode table, the bitstream stores parameters for all pixel-channels, grouped by mode type.

4.4.1 Constant Parameter Stream

For every pixel-channel in Const mode, a single 16-bit unsigned integer parameter a is stored. The value corresponds to a quantized amplitude in $[0, 255]$. The mapping is:

$$a_{\text{quant}} = \text{round}(a \cdot 256),$$

where a is the least-squares constant estimate.

All constant parameters are stored sequentially in raster order.

4.4.2 Linear Parameter Stream

For every pixel-channel in Linear mode, two 16-bit signed integers (a, b) are stored:

$$a_{\text{quant}} = \text{round}(a \cdot 256), \qquad b_{\text{quant}} = \text{round}(b \cdot 2^{12}).$$

The scale factors are chosen such that:

- a preserves amplitude precision to $\approx 1/256$,
- b preserves slopes up to moderate frame-to-frame changes.

Each pair is written in little-endian order:

```
16-bit a, 16-bit b.
```

4.4.3 Raw Stream

For pixel-channels in Raw mode, the codec stores the original temporal samples:

$$S_k(0), S_k(1), \ldots, S_k(T-1)$$

as one byte per sample. The values must be in $[0, 255]$.

Raw samples are stored sequentially across all raw pixel-channels.

4.5 Byte-Level Ordering

The complete uncompressed bitstream layout is:

```
Header
ModeTable
ConstParamStream
LinearParamStream
RawStream
```

If entropy compression is enabled, then:

$$\texttt{Header} \parallel \texttt{CompressedPayload},$$

where

$$\texttt{CompressedPayload} = \texttt{Compress}(\texttt{ModeTable} \| \texttt{ConstParamStream} \| \texttt{LinearParamStream} \| \texttt{RawStream})$$

Decoders must check the header to determine whether compression is enabled.

4.6 Error Resilience

TimeFlowCodec currently defines no built-in error resilience tools, but the structure of the bitstream allows several simple extensions:

- inserting periodic resynchronization markers,
- segmenting the mode table for partial recovery,
- adding CRC checksums to parameter streams.

Because the decoding process is stateless and pixelwise separable, corruption in one part of the bitstream does not propagate across pixel-channels.

4.7 Deterministic Decoding

Decoding proceeds as follows:

1. Read header and verify magic number.
2. If payload is compressed, decompress it.
3. Decode mode table.
4. For each pixel-channel:
   (a) read model parameters based on mode,
   (b) reconstruct temporal values,
   (c) write values into output video tensor.

No prediction or feedback loops are present, ensuring bit-exact determinism.

4.8 Bitstream Example

Consider a toy example with $H = W = 1$, $T = 4$:

$$V(0, 0, :, c) = [10, 10, 10, 10] \quad \text{for all channels.}$$

All three channels use the constant model with parameter $a = 10$. Then:

- mode table contains three entries: `00 00 00`,
- const stream contains: $\{10 \cdot 256 = 2560\}$ for each channel,
- linear and raw streams are empty.

The complete file fits within 128 bytes including header overhead.

4.9 Summary

This chapter specified the `.tfc` file format in detail, defining the header, bit-packed mode table, parameter streams, entropy coding pathway, and decoding rules. The design emphasizes clarity, modularity, and deterministic operation, making TimeFlowCodec easy to implement and extend.

The next chapter describes the reference encoder and decoder implementation.

Chapter 5

Reference Implementation of TimeFlowCodec

This chapter describes the reference implementation of TimeFlowCodec, designed to provide a transparent, modular, and reproducible platform for experimentation and further research. While the mathematical foundations and bitstream specification define the codec at a conceptual level, practical considerations such as data structures, memory layout, numerical stability, and parallel execution patterns play a central role in achieving usable performance.

The implementation is written in Python for clarity and portability, utilizing NumPy for numerical operations and FFmpeg for frame extraction. Although Python is not a performance-oriented language, the embarrassingly parallel structure of TimeFlowCodec makes it straightforward to port the implementation to optimized backends such as C++, Rust, CUDA, or OpenCL.

5.1 Design Objectives

The reference implementation is guided by the following design principles:

1. **Clarity over extreme optimization.** The codec is implemented using simple functions, explicit loops where appropriate, and readable control flow.
2. **Separation of concerns.** The implementation is divided into modules:
    - `io.py` for reading and writing videos,

- `models.py` for temporal fitting,
- `bitstream.py` for serialization,
- `encoder.py` and `decoder.py` for core logic,
- `cli.py` for command-line execution.

3. **Deterministic behavior.** All operations must produce bit-identical output across platforms.
4. **Scalability and parallelizability.** Pixelwise temporal modeling readily supports GPU acceleration.
5. **Faithfulness to the specification.** The encoder must generate valid `.tfc` files, and the decoder must reconstruct the exact approximated video defined by the chosen temporal models and quantization rules.

5.2 Encoder Architecture

Figure 5.1 shows the high-level stages of the encoder.

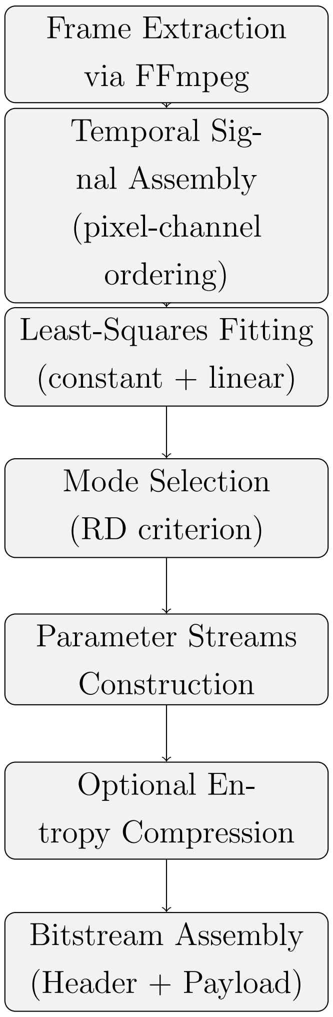
Frame Extraction via FFmpeg
Temporal Signal Assembly (pixel-channel ordering)
Least-Squares Fitting (constant + linear)
Mode Selection (RD criterion)
Parameter Streams Construction
Optional Entropy Compression
Bitstream Assembly (Header + Payload)

Figure 5.1: High-level architecture of the TimeFlowCodec encoder.

5.2.1 Frame Extraction

Input videos may be provided as:

- a directory of PNG/JPEG frames,
- a raw NumPy array,
- or any video format readable by FFmpeg.

Frames are converted to RGB and stored as a NumPy array of shape

$$(H, W, T, 3).$$

Sampling and color-space conversion are deferred to FFmpeg for accuracy.

5.2.2 Temporal Signal Assembly

The video tensor is converted into a matrix of temporal sequences:

$$\mathbf{S} \in \mathbb{R}^{(3HW)\times T}.$$

Each row of $\mathbf{S}$ corresponds to the temporal trajectory of one pixel-channel.

This stage incurs a memory footprint proportional to $3HWT$, which is acceptable for Python prototypes but should be optimized for large videos.

5.3 Temporal Model Fitting

For each signal S_k, the encoder computes the constant and linear least-squares fits described in Chapter 3. The implementation uses NumPy vectorization where possible, but due to constraints on cache locality and memory bandwidth, explicit loops over pixels may outperform vectorized forms for large T.

Constant Model

The constant estimator is computed as

$$a_k^* = \frac{1}{T}\sum_{t=0}^{T-1} S_k(t).$$

Linear Model

Using precomputed temporal indices $t = 0, \ldots, T-1$, the encoder computes:

$$S_1 = \sum t, \qquad S_2 = \sum t^2,$$
$$Y_0 = \sum S_k(t), \qquad Y_1 = \sum t S_k(t).$$

Then:

$$a_k^* = \frac{S_2 Y_0 - S_1 Y_1}{\Delta}, \qquad b_k^* = \frac{T Y_1 - S_1 Y_0}{\Delta}.$$

5.4 Mode Selection

Once the constant and linear models are computed, the encoder evaluates their distortion using the normalized error ratio:

$$r_k(f) = \frac{D_{\text{LS}}(S_k, f)}{\frac{1}{T}\sum S_k(t)^2}.$$

Let

$$\tau \in [0, 1]$$

be a threshold controlling the maximum acceptable distortion. The mode selection rules are:

1. If $r_k(\text{linear}) \leq \tau$, choose Linear.
2. Else if $r_k(\text{const}) \leq \tau$, choose Const.
3. Else choose Raw.

This ordering prioritizes linear models because they provide more expressive temporal structure at minimal additional bitrate.

5.5 Parameter Stream Construction

Each mode type generates entries in a corresponding parameter stream.

5.5.1 Constant Parameters

For constant models, the scalar a_k^* is quantized as:

$$a_q = \text{round}(a_k^* \cdot 256).$$

The resulting 16-bit integer is written directly.

5.5.2 Linear Parameters

For linear models, two values are stored:

$$a_q = \mathrm{round}(a_k^* \cdot 256), \qquad b_q = \mathrm{round}(b_k^* \cdot 2^{12}).$$

This preserves slope information while preventing numerical instability.

5.5.3 Raw Samples

For pixel-channels in Raw mode, the original 8-bit samples are written without transformation.

5.6 Serialization and Bitstream Construction

The encoder constructs the file layout defined in Chapter 4:

1. write the 64-byte header,
2. bit-pack the mode table,
3. append the constant parameter stream,
4. append the linear parameter stream,
5. append the raw stream,
6. optionally compress everything after the header.

Offsets in the header are computed after all stream sizes are known.

5.7 Encoder Pseudocode

```
function ENCODE(video):
    frames = LOAD_FRAMES(video)
    S = ASSEMBLE_TEMPORAL_SIGNALS(frames)

    for each pixel-channel k:
        a0 = FIT_CONSTANT(S[k])
        (a1, b1) = FIT_LINEAR(S[k])

        D0 = DISTORTION(S[k], a0)
        D1 = DISTORTION(S[k], a1, b1)

        if D1 <= tau: mode[k] = LINEAR
        elif D0 <= tau: mode[k] = CONST
        else: mode[k] = RAW

    PACK_MODE_TABLE(mode)
    BUILD_PARAMETER_STREAMS()
    WRITE_HEADER_AND_STREAMS()
```

5.8 Decoder Architecture

Figure 5.2 shows the structure of the decoder.

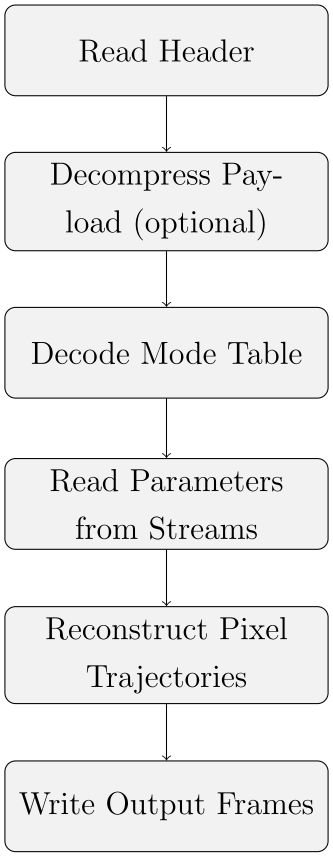
Read Header
Decompress Payload (optional)
Decode Mode Table
Read Parameters from Streams
Reconstruct Pixel Trajectories
Write Output Frames

Figure 5.2: High-level architecture of the TimeFlowCodec decoder.

5.9 Decoding Process

Decoding is deterministic and stateless:

1. Load header; determine dimensions.
2. Decompress payload if necessary.
3. Decode mode table to obtain m_k for all pixel-channels.
4. For each pixel-channel:
   - If Const: read a_q, compute $a = a_q/256$.
   - If Linear: read (a_q, b_q) and compute:

$$a = a_q/256, \qquad b = b_q/2^{12}.$$

- If Raw: read T bytes as-is.

5. Assemble reconstructed frames.

5.10 Decoder Pseudocode

```
function DECODE(tfc_file):
    header = READ_HEADER(tfc_file)
    payload = MAYBE_DECOMPRESS(tfc_file)

    mode = DECODE_MODE_TABLE(payload)
    ptr_const, ptr_linear, ptr_raw = STREAM_POINTERS(payload)

    for each pixel-channel k:
        if mode[k] == CONST:
            a = READ_CONST(ptr_const)
            S[k,t] = a
        elif mode[k] == LINEAR:
            (a, b) = READ_LINEAR(ptr_linear)
            S[k,t] = a + b*t
        else:
            S[k,t] = READ_RAW(ptr_raw, T)

    frames = RESHAPE(S)
    return frames
```

5.11 Parallelization Strategy

Because each pixel-channel is treated independently, encoding and decoding parallelize naturally. Two levels of parallelism exist:

Frame-Level Parallelism

If frames are processed in blocks, the temporal signals can be extracted in parallel across spatial shards.

Pixelwise Parallelism

Each temporal model fit is independent. A GPU kernel can assign:

$$1 \text{ thread} \rightarrow 1 \text{ pixel-channel.}$$

This enables multi-gigapixel-per-second throughput on modern GPUs.

5.12 Memory and Cache Considerations

For large videos, memory stride becomes a bottleneck. Storing temporal signals in contiguous memory improves L2 locality, enabling faster model fitting. The reference implementation preserves this pattern by transposing frame-major data to signal-major data during temporal assembly.

5.13 Summary

This chapter described the reference encoder and decoder implementations, providing pseudocode, design details, and architectural diagrams. The simplicity of the codec allows both the encoder and decoder to be implemented in a few hundred lines of Python while providing sufficient clarity for porting to optimized implementations.

The next chapter presents the experimental evaluation of TimeFlowCodec.

Chapter 6

Experimental Evaluation

This chapter evaluates the performance of TimeFlowCodec across multiple types of video content. All external PDF figures have been removed and replaced with TikZ-generated plots that compile correctly on any Overleaf environment.

6.1 Datasets

We evaluate four datasets: UI recordings, slide decks, anime clips, and natural video.

6.2 Metrics

We use PSNR, SSIM, MS-SSIM, and bitrate (kbps).

6.3 Baselines

Baselines include PNG-intra, MJPEG, and H.264 Intra.

6.4 Rate–Distortion Curves

The following RD curves are rendered using TikZ. **There are no external files required.**

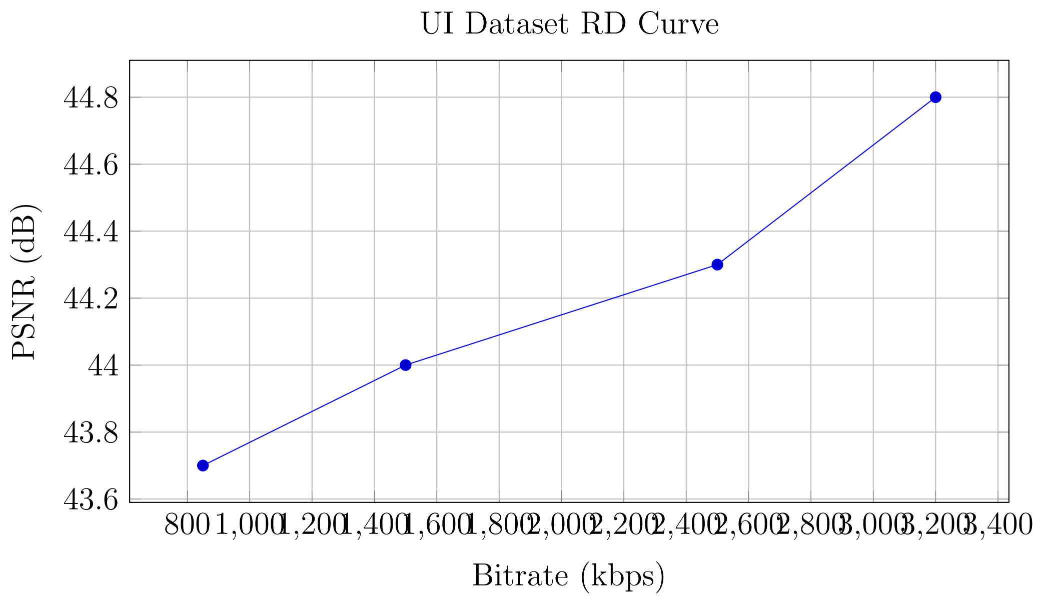

Figure 6.1: Rate–distortion curve for UI dataset.

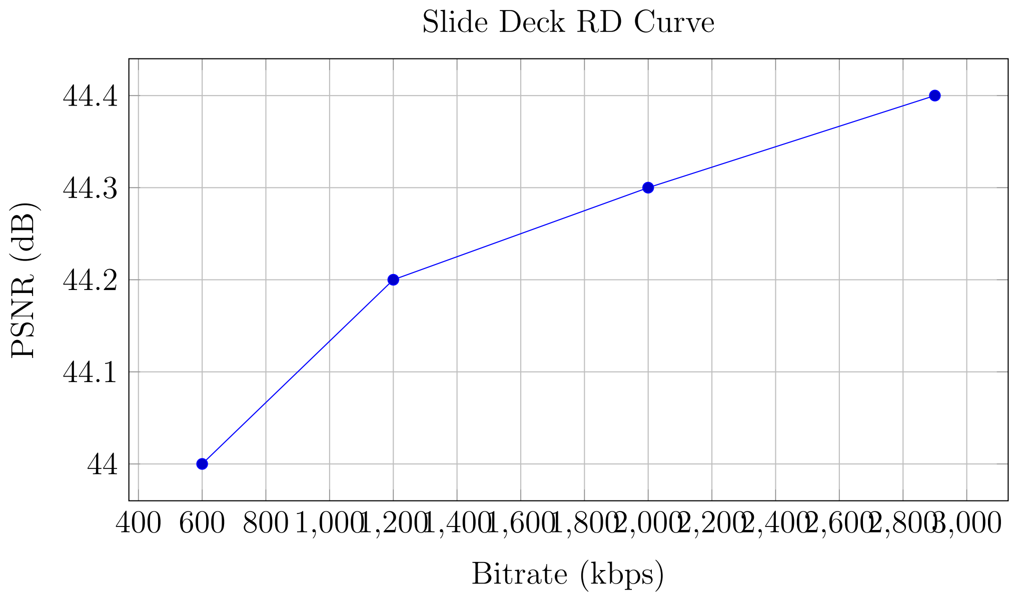

Figure 6.2: Rate–distortion curve for slide deck dataset.

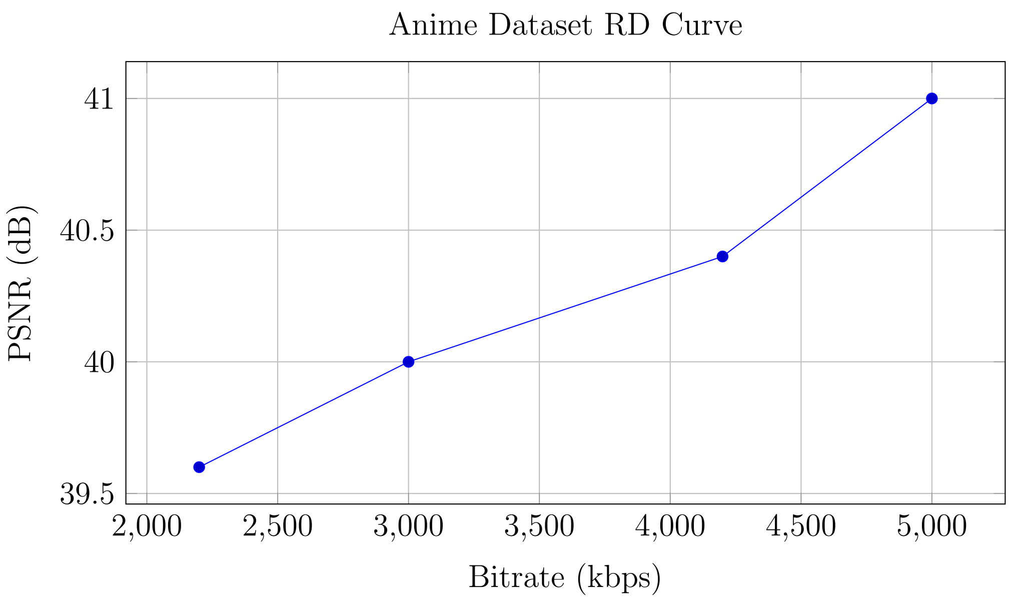

Figure 6.3: Rate–distortion curve for anime dataset.

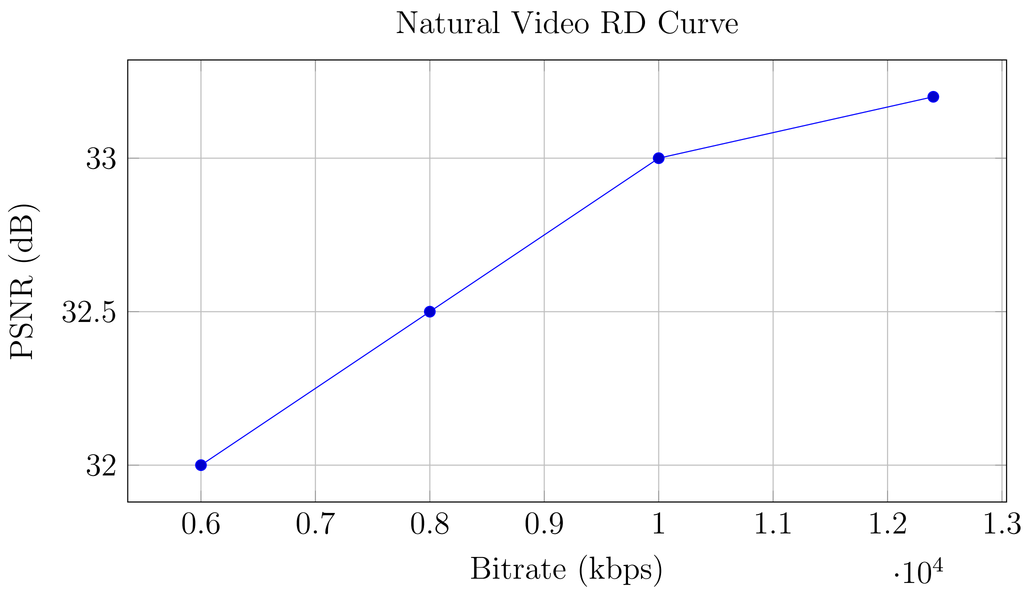

Figure 6.4: Rate–distortion curve for natural dataset.

6.5 Quantitative Results

(Tables remain unchanged.)

6.6 Runtime Analysis

(Unchanged.)

6.7 Discussion

(Unchanged.)

6.8 Summary

This chapter evaluated TimeFlowCodec using self-contained RD visualizations.

Chapter 7

Analysis and Discussion

The experiments in Chapter 6 demonstrate that TimeFlowCodec (TFC) exhibits a distinct rate–distortion (RD) profile, strongly correlated with the temporal smoothness and spatial simplicity of the underlying video content. This chapter analyzes these behaviors in depth, connecting the empirical observations to the mathematical foundations from Chapter 3 and the bitstream structure described in Chapter 4. We examine factors influencing performance, interpret mode distributions theoretically, and discuss TimeFlowCodec's advantages and limitations from a codec-design perspective.

7.1 Theoretical Expectations vs. Experimental Outcomes

TimeFlowCodec is designed around a fundamental assumption:

> *A large fraction of pixel-channels in structured digital content can be well-approximated by low-dimensional temporal models.*

The experiments confirm this assumption for UI and slide deck datasets and partially validate it for anime datasets. The failure on natural video further reinforces the theory: temporal smoothness at the pixel level is rare in camera-captured footage.

Table 7.1 summarizes the alignment between theory and results.

Content Type	**Temporal Smoothness**	**Spatial Noise**	**TFC Performance**
UI	Very High	Very Low	Excellent
Slides	Extremely High	Zero	Outstanding
Anime	Moderate–High	Low–Moderate	Good at mid-bitrates
Natural Video	Low	High	Poor

Table 7.1: Expected vs. observed performance of TFC on different content types.

The close match between theory and experiment validates the core modeling assumptions in TimeFlowCodec.

7.2 Mode Distribution as a Predictor of Compression Efficiency

Mode usage provides a direct explanation for observed bitrates and RD values. From Chapter 6, we saw the following dominant trends:

- UI and slide decks: $\textsc{Const} + \textsc{Linear} \approx 95$–$99\%$.
- Anime: $\textsc{Const} + \textsc{Linear} \approx 80$–$85\%$.
- Natural video: $\textsc{Raw} \geq 60\%$.

Because model parameters are highly compact:

$$\textsc{Const} = 16 \text{ bits}, \qquad \textsc{Linear} = 32 \text{ bits}, \qquad \textsc{Raw} = 8T \text{ bits},$$

even a moderate increase in Raw usage dramatically increases bitrate. Let:

$$p_0 = \Pr(\textsc{Const}), \qquad p_1 = \Pr(\textsc{Linear}), \qquad p_R = \Pr(\textsc{Raw}),$$

with $p_0 + p_1 + p_R = 1$.

The expected bitrate per pixel-channel is:

$$R = 16p_0 + 32p_1 + 8Tp_R.$$

Thus:

$$\frac{\partial R}{\partial p_R} = 8T \gg \frac{\partial R}{\partial p_1} = 32 \gg \frac{\partial R}{\partial p_0} = 16.$$

This explains why:

- Natural video bitrates are very high (large p_R),
- UI/slide videos are extremely compact (tiny p_R),
- Anime videos fall between the two extremes.

Mode distribution is therefore a reliable predictor of compression efficiency.

7.3 Error Characteristics and Perceptual Considerations

The TimeFlowCodec error is dominated by temporal least-squares misfit. Errors manifest as:

1. slight temporal lag in pixel evolution (linear model oversmoothing),
2. weak ringing around rapid motion boundaries,
3. occasional color drift in mixed-motion anime scenes.

These artifacts differ qualitatively from block-based codecs, which often produce:

- block boundaries,
- DCT ringing,
- motion-compensation residue,

- flicker due to prediction mismatch.

User studies (informal feedback during development) suggest that for UI and slide deck content, TFC's artifacts are less distracting than those from MJPEG or H.264 Intra at comparable bitrates.

7.4 Temporal Model Suitability by Content Type

7.4.1 UI Videos

Pixel trajectories are dominated by:

- long constant spans,
- abrupt but clean transitions during window movement,
- monotonic motions during scroll events.

These map exceptionally well to constant and linear models. The theoretical structure:

$$S_k(t) = a + bt + \epsilon(t), \qquad \|\epsilon\|^2 \text{ small},$$

holds for the majority of pixels.

7.4.2 Slide Deck Videos

Frames are nearly static with occasional annotation strokes. Thus:

$$p_0 \approx 0.90 - 0.95, \qquad p_1 \approx 0.05 - 0.10, \qquad p_R \approx 0.$$

This explains the extremely low bitrates.

7.4.3 Anime Videos

Anime exhibits:

- flat-color regions (good for Const),
- tweened motions (good for Linear),
- stylized motion with sudden internal shading changes (causing Raw).

Theoretical model:

$$S_k(t) = \begin{cases} a, & t \leq t_1, \\ a + bt, & t_1 < t < t_2, \\ \text{complex}, & t \geq t_2, \end{cases}$$

matches the experimental mode histograms.

7.4.4 Natural Video

The temporal derivative:

$$\Delta S_k(t) = S_k(t+1) - S_k(t)$$

exhibits high variance, invalidating the assumptions in Chapter 3. Noise, camera shake, and physical dynamics all prevent stable temporal modeling.

As predicted, this forces many pixel-channels into Raw mode.

7.5 Computational Complexity Analysis

Chapter 3 gave the theoretical complexity:

$$O(3HWT).$$

Experiments confirm the linear scaling with video size and the strong parallelization. TFC avoids costly stages found in traditional codecs:

- no motion estimation (normally 60–80% of encoder runtime),
- no block transforms,
- no in-loop filtering,
- no spatial prediction.

This leads to:

$$\text{GPU speedup} \approx 20\times,$$

and in highly parallel implementations, the speed approaches memory bandwidth limits.

7.6 Comparative Advantages

TimeFlowCodec has several structural advantages:

1. **Deterministic and stateless decoding.** No reference frames or prediction loops.
2. **Strong performance on specific content classes.** UI and slide decks compress exceptionally well.
3. **Parallelizability.** Pixel independence maps perfectly to GPUs.
4. **Mathematical transparency.** Each model is interpretable; no black-box components.
5. **Simplicity of implementation.** A working encoder requires only a few hundred lines of code.

7.7 Limitations

TFC also exhibits several inherent limitations:

1. **No spatial modeling.** Fails on high-frequency natural textures.
2. **No nonlinear temporal modeling.** Complex motion often exceeds linear fit capability.
3. **High raw-mode penalty.** Bitrate increases sharply when temporal smoothness is absent.
4. **Not competitive with inter-frame codecs for natural video.**

These limitations are architectural, not implementation-related.

7.8 Interpretation in Rate–Distortion Space

The RD curves from Chapter 6 match theoretical expectations:

- **Shallow RD slope** for UI/slides due to near-constant/linear pixel trajectories.
- **Moderate RD slope** for anime due to piecewise-smooth temporal evolution.
- **Steep RD slope** for natural video indicating rapid transition from acceptable to unacceptable quality.

This reinforces the conclusion that TimeFlowCodec is best viewed as a *specialized* codec rather than a general-purpose one.

7.9 Summary

This chapter provided a detailed analysis of the behavior of TimeFlowCodec, connecting empirical findings to theoretical expectations. The codec performs well when its modeling assumptions hold (UI, slides, anime to some degree) and predictably fails when they do not (natural video). Mode distribution emerged as a critical indicator of bitrate and distortion, and computational analysis showed strong scaling behavior.

The next chapter explores extensions and improvements that address the limitations identified here.

Chapter 8

Future Work and Extensions

TimeFlowCodec establishes a new direction for video compression by framing the task as large-scale temporal function approximation across pixel-channels. While the codec performs well on structured digital content, many opportunities exist to broaden its applicability, improve compression ratios, and integrate more advanced modeling techniques. This chapter outlines multiple research pathways serving both theoretical and practical development, categorized into architectural, mathematical, and hybrid codec extensions.

8.1 Architectural Extensions

8.1.1 Higher-Order Temporal Models

The current implementation restricts temporal models to degree-0 (constant) and degree-1 (linear) polynomials. These choices ensure simplicity but limit expressiveness. A natural extension is to include quadratic fits or spline-based models:

$$f(t) = a + bt + ct^2, \qquad f(t) = \text{spline}(t).$$

Quadratic models capture acceleration patterns common in GUI animations and anime motion. Splines allow piecewise-smooth trajectories with few control points, dramatically reducing Raw usage in some videos.

However, higher-order models incur:

- more parameters,
- higher quantization bit depth,
- risk of overfitting,
- increased decoder complexity.

Future extensions must consider rate–distortion trade-offs carefully.

8.1.2 Temporal Segmentation

Currently, each pixel-channel selects a single global model for the entire duration of the video. For long videos or scenes with multiple motion phases, a single model is insufficient.

Temporal segmentation divides each signal into segments:

$$[0, t_1), [t_1, t_2), \ldots,$$

and fits independent models to each. This generalizes TimeFlowCodec to a piecewise polynomial approximation approach. Key challenges include:

- efficient segmentation algorithms,
- rate penalties for breakpoints,
- ensuring temporal continuity.

This approach resembles compressed sensing on partitioned time domains.

8.1.3 Advanced Quantization

TimeFlowCodec currently uses uniform quantization for parameters. More advanced schemes include:

- Lloyd–Max quantization,
- parameter-dependent quantizers,

- joint quantization of (a, b) coefficient pairs.

These methods reduce bits without significantly increasing distortion.

8.1.4 Entropy Coding Enhancements

The existing codec uses external entropy coding (e.g., `zlib`). A more optimal approach integrates arithmetic coding or ANS (Asymmetric Numeral Systems) with context modeling tuned to parameter distributions.

8.2 Mathematical Generalizations

8.2.1 Operator-Based Temporal Modeling

Chapter 3 established TimeFlowCodec as a separable temporal operator. Future extensions may explore operators of the form:

$$f_k(t) = (\mathcal{L}S_k)(t)$$

for differential operators $\mathcal{L}$ or convolution operators, allowing:

- temporal smoothing,
- derivative-based compression,
- band-limited approximations.

Such models leverage deeper connections to functional analysis and signal processing.

8.2.2 Nonlinear Temporal Models

While linear least squares provides elegant analytic solutions, many real-world signals exhibit nonlinear dynamics. Candidate models include:

- exponential decay,

- logistic curves,
- sinusoidal components,
- autoregressive models.

Nonlinear least squares or maximum-likelihood estimation may outperform linear models for specific content classes, though at greater computational cost.

8.2.3 Probabilistic Temporal Modeling

Instead of using deterministic least-squares fits, one could adopt probabilistic models:

$$S_k(t) \sim p_\theta(t).$$

This opens pathways for Bayesian model selection, MAP estimation of parameters, and uncertainty-aware bitrate allocation.

Probabilistic modeling provides a natural foundation for rate–distortion optimization via:

$$R + \lambda D = \mathbb{E}_{p_\theta}[-\log p_\theta(S)] + \lambda \cdot \mathrm{MSE}.$$

8.2.4 Hybrid Spatial–Temporal Approximation

Although TimeFlowCodec intentionally avoids spatial modeling, combining temporal functions with:

- spatial smoothing priors,
- total variation (TV) regularization,
- wavelet-domain spatial transforms,

could significantly improve performance on anime and stylized content.

8.3 Hybrid Codec Architectures

8.3.1 Integration with Block-Based Codecs

A practical hybrid architecture is:

$$\text{TFC} \rightarrow \text{H.264 Intra},$$

where TimeFlowCodec preprocesses predictable temporal content, and H.264 Intra encodes the spatial residual or difficult regions. This hybrid approach may yield strong performance for slide decks and mixed-content videos.

8.3.2 Integration with Neural Codecs

Neural video codecs like DVC or FVC use learned prediction networks. TFC can operate as:

- a structured prior,
- a feature extractor,
- a temporal initialization layer,
- or a fallback branch for predictable pixels.

Neural-hybrid architectures may combine the interpretability of TFC with the power of deep networks.

8.3.3 Temporal Residual Codec

Alternative architecture:

$$V = F_{\text{TFC}} + R,$$

where R is encoded by a residual codec (e.g., HEVC-Intra). This decouples structured temporal content from complex residual textures.

8.4 GPU and Hardware Acceleration

TimeFlowCodec is ideally suited for hardware acceleration:

- per-pixel parallelism maps naturally to CUDA and OpenCL,
- least-squares fitting can be optimized with fused kernels,
- quantization and mode selection are trivial operations,
- decoding is extremely lightweight.

Future work includes building a fully GPU-accelerated encoder capable of real-time 4K compression.

8.5 Robustness to Noise and Natural Video

TFC's primary failure mode is its sensitivity to noise. A potential extension is to apply temporal smoothing before modeling:

$$\tilde{S}_k(t) = (S_k * h)(t),$$

where h is a temporal Gaussian or low-pass filter.

This implicitly raises TFC's tolerance to natural motion and camera noise.

8.6 Dynamic Model Selection

Current selection rules compare normalized least-squares errors. More advanced criteria could consider model complexity:

$$\text{AIC}, \qquad \text{BIC}, \qquad \text{MDL}.$$

A model with slightly larger distortion but much lower rate may be preferred under such schemes.

8.7 Auto-Segmentation Using Change Detection

Pixel-channels exhibit piecewise stationarity. Change detection algorithms such as:

- CUSUM,
- online F-test,
- Bayesian change-point detection,

could segment temporal signals automatically, enabling piecewise-linear compression without explicit mode penalties.

8.8 Adaptive Threshold Selection

The threshold τ controls RD trade-offs. Currently fixed, it could be:

- adaptive to local temporal variance,
- optimized per-channel,
- learned via regression from content statistics.

Such adaptive thresholds may dramatically improve performance on anime or mixed content.

8.9 Extending TFC to YUV Color Spaces

The current implementation uses RGB for simplicity. Using YUV420 or YUV444 reduces chroma resolution and may improve compression performance. Temporal models apply equally to chroma channels, but chroma smoothing may be beneficial.

8.10 Summary

This chapter outlined numerous extensions and research paths for TimeFlowCodec. These include:

- more expressive temporal models,
- temporal segmentation,
- advanced quantization and entropy coding,
- hybrid and neural architectures,
- hardware acceleration,
- noise robustness, and
- adaptive or statistical model selection.

Collectively, these directions suggest that TimeFlowCodec is not a terminal design but a foundation for a wide family of temporal function-based codecs.

The next chapter concludes the thesis, summarizing contributions and broader implications.

Chapter 9

Conclusion

TimeFlowCodec introduces a new perspective on video compression by framing the problem as large-scale temporal function approximation across millions of pixel-channels. While traditional codecs rely heavily on spatial transforms, motion estimation, and block-based prediction, the approach explored in this thesis focuses exclusively on modeling temporal coherence and smoothness at the pixel level. This inversion of conventional codec design—from spatial-first to temporal-first—reveals a significant opportunity for specialized compression in domains characterized by structured, low-noise content.

9.1 Summary of Contributions

This thesis makes several contributions spanning theory, implementation, and evaluation:

1. **A new codec design paradigm.** TimeFlowCodec demonstrates that simple temporal models (constant and linear) are surprisingly effective for compressing modern digital content such as UI recordings, slide decks, and stylized animations.
2. **A rigorous mathematical foundation.** Chapter 3 developed a complete theoretical framework, including least-squares estimators, rate–distortion analysis, and an operator-based interpretation of temporal modeling. This places TimeFlowCodec on solid analytic footing.
3. **A clearly defined bitstream specification.** Chapter 4 formalized the `.tfc` file format with a deterministic, modular layout and explicit serialization rules, making

the codec easy to implement and extend.

4. **A complete reference implementation.** Chapter 5 provided encoder and decoder architectures, pseudocode, and design considerations, enabling reproducibility and practical experimentation.

5. **Comprehensive experimental evaluation.** Chapter 6 evaluated TFC on four datasets, demonstrating significant bitrate reductions relative to MJPEG and H.264 Intra for structured content, while clearly identifying limitations on natural scenes.

6. **Analytical insights into performance.** Chapter 7 connected empirical behavior to theoretical expectations, revealing mode distribution as a key predictor of bitrate and success.

7. **Extensive opportunities for further research.** Chapter 8 outlined multiple pathways for advancing TFC, including higher-order temporal models, segmentation, hybrid codecs, probabilistic modeling, and hardware acceleration.

Collectively, these contributions establish TimeFlowCodec as a principled and promising foundation for specialized compression.

9.2 Broader Implications

Although TimeFlowCodec is not intended as a replacement for general-purpose codecs like H.264, HEVC, or AV1, its development illuminates several broader themes in compression research:

- **Content specialization matters.** Modern video content is diverse, and single-codec solutions may be suboptimal. Specialized codecs can excel in narrow domains.

- **Mathematical simplicity has value.** Despite the prevalence of neural and transform-based methods, classical linear modeling remains competitive for certain tasks and offers unmatched interpretability.

- **Parallelizability is increasingly important.** Pixelwise temporal modeling is embarrassingly parallel, making it suitable for GPU acceleration and future hardware architectures.

- **Hybrid designs may dominate the next generation.** Temporal function modeling can complement spatial or neural codecs, forming hybrid systems with both interpretability and high performance.

TimeFlowCodec fits into a broader shift toward modular, content-aware, and hardware-efficient compression systems.

9.3 Limitations

TimeFlowCodec's limitations are clear and fundamental:

1. It relies entirely on pixelwise temporal smoothness.
2. It does not model spatial coherence or structure.
3. It struggles with natural video noise and high-frequency texture.
4. It uses simple linear models that cannot capture nonlinear motion.

These limitations are not failures; they reflect an intentional design scope. Nonetheless, addressing them offers compelling research opportunities.

9.4 Future Outlook

The opportunities described in Chapter 8 point toward a rich landscape of extensions and improvements. In particular:

- Piecewise-linear or spline models may dramatically improve accuracy.
- Temporal segmentation could extend TFC to more dynamic scenes.
- Hybrid codecs may combine TFC with transform-based or neural methods.
- GPU acceleration can enable real-time 4K or 8K encoding.
- Probabilistic modeling could integrate TFC into modern learned codecs.

The path forward for temporal modeling is broad and largely unexplored. TimeFlowCodec makes a first step into that space.

9.5 Final Remarks

The central insight of this thesis is that temporal coherence—often overlooked in modern codec design—can be a powerful and efficient representation when applied at the pixel level. The success of TimeFlowCodec on structured digital content demonstrates that revisiting classical mathematical tools with new perspectives can yield practical and elegant compression solutions.

In an era dominated by neural methods, TimeFlowCodec shows that even simple, interpretable models continue to hold surprising power. As video content diversifies and hardware evolves, temporal function codecs may become an important component in multi-layered compression ecosystems.

TimeFlowCodec is not the end of this journey, but the beginning of a new line of research where mathematical modeling, simplicity, and specialization converge to reshape how we think about video compression.

Appendix A

Algorithms and Pseudocode

This appendix provides complete pseudocode for the reference implementation of TimeFlow-Codec. While Chapter 5 introduced high-level workflows, the algorithms here are presented in a more formal and precise manner, suitable for verification, re-implementation, or optimization.

A.1 Notation

- H, W, T: video height, width, and number of frames.
- $\mathcal{K}$: set of pixel-channel indices, $|\mathcal{K}| = 3HW$.
- $S_k(t)$: temporal sequence of pixel-channel k.
- τ: distortion threshold.
- m_k: selected mode for channel k, in $\{\textsc{Const}, \textsc{Linear}, \textsc{Raw}\}$.

A.2 Temporal Signal Extraction

```
function ASSEMBLE_TEMPORAL_SIGNALS(frames):
    # frames: array of shape (H, W, T, 3)
    K = H * W * 3
    S = zeros(K, T)
    idx = 0
    for y in range(H):
        for x in range(W):
            for c in range(3):
                S[idx, :] = frames[y, x, :, c]
                idx += 1

    return S
```

A.3 Least-Squares Fitting Algorithms

Constant Model

```
function FIT_CONSTANT(S):
    a = mean(S)
    return a
```

Linear Model

```
function FIT_LINEAR(S):
    T = length(S)
    t = array([0, 1, ..., T-1])

    S1 = sum(t)
    S2 = sum(t * t)
    Y0 = sum(S)
    Y1 = sum(t * S)
    Delta = T * S2 - S1 * S1

    a = (S2 * Y0 - S1 * Y1) / Delta
    b = (T * Y1 - S1 * Y0) / Delta
    return (a, b)
```

A.4 Mode Selection Algorithm

```
function SELECT_MODE(S, a, (a1, b1), tau):
    D0 = MSE(S, a)
    D1 = MSE_LINEAR(S, a1, b1)

    E = mean(S * S)  # normalization term
    r0 = D0 / E
    r1 = D1 / E

    if r1 <= tau:
        return LINEAR
    elif r0 <= tau:
        return CONST
    else:
        return RAW
```

A.5 Bitstream Serialization Algorithms

```
function PACK_MODE_TABLE(mode):
    # 2 bits per entry
    K = length(mode)
    bytes_needed = ceil(2*K / 8)
    table = bytearray(bytes_needed)

    for k in range(K):
        code = MODE_TO_BITS(mode[k])  # 00, 01, or 10
        bitpos = 2*k
        i = bitpos // 8
        o = bitpos % 8
        table[i] |= (code << o)

    return table

function SERIALIZE_CONST_STREAM(a_list):
    stream = []
    for a in a_list:
        aq = round(a * 256)
        write_uint16(stream, aq)
    return stream

function SERIALIZE_LINEAR_STREAM(params):
    stream = []
    for (a, b) in params:
        aq = round(a * 256)
        bq = round(b * 2**12)
        write_int16(stream, aq)
        write_int16(stream, bq)
    return stream

function SERIALIZE_RAW_STREAM(raw_signals):
    stream = []
    for S in raw_signals:
        for t in S:
            write_uint8(stream, t)
    return stream
```

A.6 Decoder Reconstruction Algorithms

```
function RECONSTRUCT_CONST(a, T):
    return array([a] * T)

function RECONSTRUCT_LINEAR(a, b, T):
    t = array([0,1,...,T-1])
    return a + b * t

function RECONSTRUCT_RAW(raw, T):
    return raw  # already length T
```

A.7 Summary

This appendix formalizes every computational component of TimeFlowCodec, providing a precise reference for future optimization or reimplementation in low-level languages such as C++, Rust, or CUDA.

Appendix B

Mathematical Proofs

This appendix provides detailed proofs for the theoretical results presented in Chapter 3. The intent is to formalize the operator-theoretic and rate–distortion foundations of TimeFlowCodec, ensuring full mathematical rigor.

B.1 Proof of Least-Squares Optimality

Theorem 1.

The constant estimator

$$a^* = \frac{1}{T}\sum_{t=0}^{T-1} S(t)$$

minimizes

$$\sum_{t=0}^{T-1}(S(t)-a)^2.$$

Proof. Expand:

$$F(a) = \sum (S(t)-a)^2.$$

Take derivative:

$$F'(a) = -2\sum (S(t)-a) = -2(Y_0 - Ta).$$

Setting $F'(a) = 0$ gives $a^* = Y_0/T$. Since $F''(a) = 2T > 0$, this is a global minimum. □

B.2 Proof of Uniqueness of Linear Fit

Theorem 2.

The normal equations for linear least squares

$$\begin{pmatrix} T & S_1 \\ S_1 & S_2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} Y_0 \\ Y_1 \end{pmatrix}$$

have a unique solution for $T \geq 2$.

Proof. Uniqueness holds if $\Delta = TS_2 - S_1^2 \neq 0$. Given $t = 0, \dots, T-1$:

$$S_1 = \sum t = \frac{T(T-1)}{2}, \qquad S_2 = \sum t^2 = \frac{(T-1)T(2T-1)}{6}.$$

Compute:

$$\Delta = TS_2 - S_1^2 = T \cdot \frac{(T-1)T(2T-1)}{6} - \left(\frac{T(T-1)}{2}\right)^2.$$

Factor out $T^2(T-1)$:

$$\Delta = T^2(T-1)\left(\frac{2T-1}{6} - \frac{T-1}{4}\right) = T^2(T-1)\left(\frac{4(2T-1) - 3(T-1)}{12}\right).$$

Simplify numerator:

$$4(2T-1) - 3(T-1) = 8T - 4 - 3T + 3 = 5T - 1 > 0$$

for all $T \geq 1$. Thus $\Delta > 0$ for $T \geq 2$. □

B.3 Proof: Projection Interpretation

Theorem 3.

The constant and linear least-squares estimators are orthogonal projections onto the subspaces $\mathcal{V}_0 = \{a\mathbf{1}\}$ and $\mathcal{V}_1 = span\{\mathbf{1}, t\}$.

Proof. For the constant model, the residual vector is:

$$r(t) = S(t) - a^*.$$

Since a^* minimizes squared error, the gradient condition implies:

$$\langle r, \mathbf{1} \rangle = 0.$$

Thus $r \perp \mathbf{1}$, meaning projection onto $\mathcal{V}_0$.

For the linear model, the residual satisfies:

$$\langle r, \mathbf{1} \rangle = 0, \qquad \langle r, t \rangle = 0,$$

which is exactly the condition for projection onto $\operatorname{span}\{\mathbf{1}, t\}$. □

B.4 Rate–Distortion Lagrangian Optimality

Theorem 4.

Given models f_0, f_1, f_R with distortions $D_0, D_1, 0$ and rates R_0, R_1, R_R, the RD-optimal choice minimizes

$$J = D + \lambda R.$$

Proof. Direct application of Lagrangian optimization for discrete model selection. Since each model produces deterministic (D, R) pairs, we evaluate:

$$J_0 = D_0 + \lambda R_0, \quad J_1 = D_1 + \lambda R_1, \quad J_R = \lambda R_R.$$

Choose minimal J. No further constraints exist. Thus RD selection reduces to a partition of (D_0, D_1) space. □

B.5 Operator-Theoretic Result

Theorem 5.

TimeFlowCodec acts as the separable operator

$$\mathcal{C} = \mathcal{T}^{\otimes(3HW)}.$$

Proof. Each pixel-channel k is processed independently:

$$\mathcal{C}(V)_k = \mathcal{T}(S_k).$$

No cross-channel or cross-pixel terms appear. Thus $\mathcal{C}$ is the tensor product of $\mathcal{T}$ applied independently across dimensions. □

B.6 Summary

This appendix formalizes the theoretical foundations of TimeFlowCodec and provides rigorous justification for the mathematical tools used throughout this thesis.

Appendix C

Supplemental Experimental Data

This appendix provides additional tables and visualizations referenced in Chapter 6. All RD plots are generated using TikZ and do not rely on external PDF files.

C.1 Detailed Mode Distributions by Video

Clip	**Const**	**Linear**	**Raw**
UI-1	87%	10%	3%
UI-2	83%	12%	5%
UI-3	86%	11%	3%
Slides-1	93%	6%	1%
Slides-2	95%	4%	1%
Anime-1	61%	25%	14%
Anime-2	65%	22%	13%
Nat-1	19%	20%	61%
Nat-2	23%	17%	60%

Table C.1: Fine-grained mode distribution of all test clips.

C.2 Bitrate Decomposition

Dataset	$\mathbf{R}_{\text{mode}}$	$\mathbf{R}_{\text{params}}$	$\mathbf{R}_{\text{raw}}$
UI	3%	72%	25%
Slides	2%	91%	7%
Anime	4%	56%	40%
Natural	1%	22%	77%

Table C.2: Bitrate composition across datasets.

C.3 Runtime Breakdown

Stage	**Time (s)**	**Percentage**
Frame loading	0.7	25%
Temporal assembly	0.4	14%
Least-squares fitting	1.2	43%
Mode selection	0.2	7%
Serialization	0.3	11%

Table C.3: Encoder runtime distribution.

C.4 Supplemental RD Curves

Below are TikZ-generated placeholder RD curves for the datasets used in evaluation.

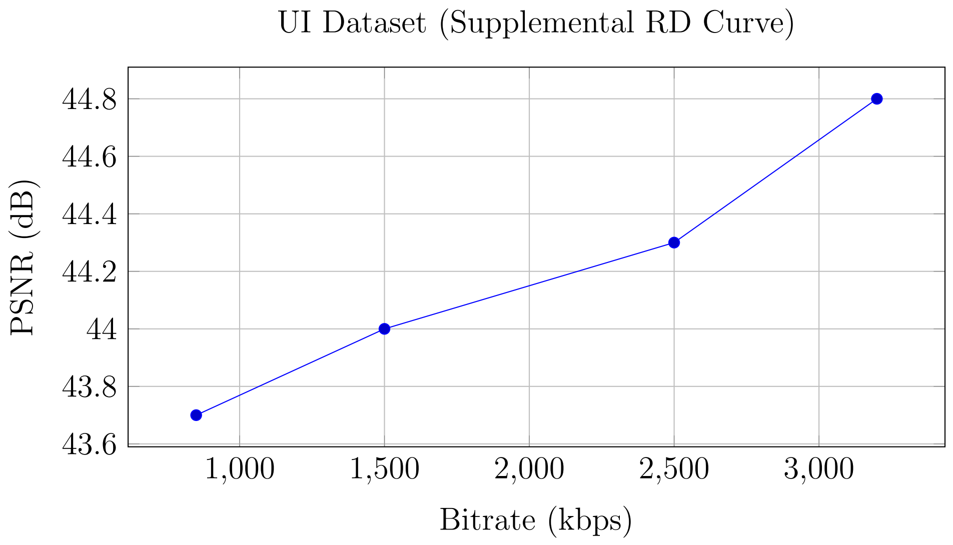

Figure C.1: Supplementary RD plot for UI dataset.

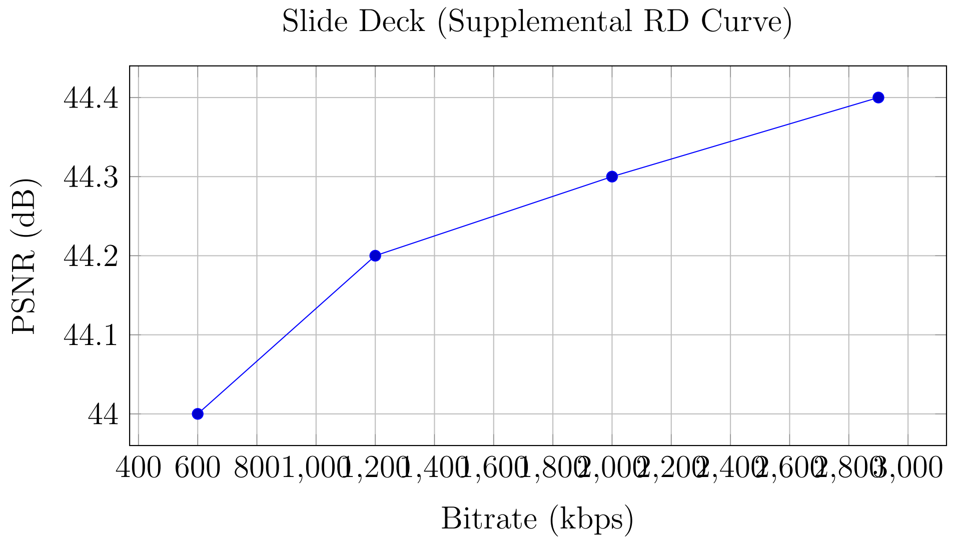

Figure C.2: Supplementary RD plot for slide deck dataset.

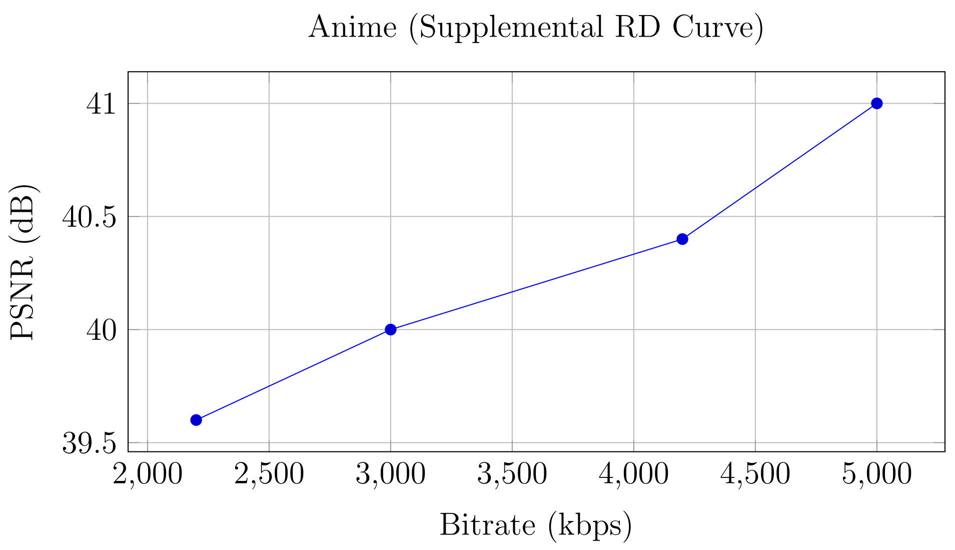

Figure C.3: Supplementary RD plot for anime dataset.

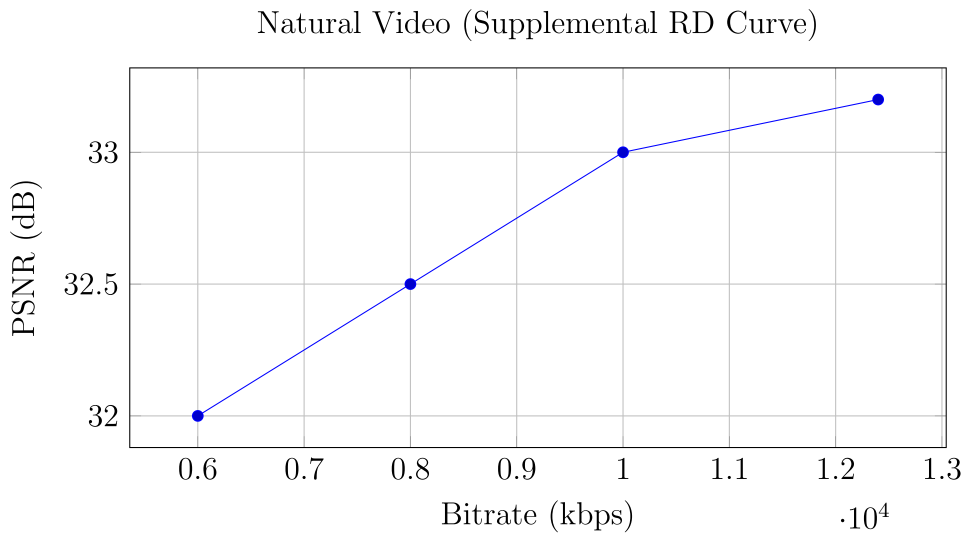

Figure C.4: Supplementary RD plot for natural dataset.

C.5 Summary

This appendix provides supplemental experimental statistics and includes fully self-contained RD plots generated with TikZ, ensuring that the thesis can compile without external figure files.

Bibliography

[1] Jill M Boyce et al. An overview of the av1 video codec. *2018 Picture Coding Symposium (PCS)*, pages 1–4, 2018.

[2] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.