Accelerating Real-Time Voltage Imaging

A Modular, GPU-Accelerated Framework for High-Speed Neural Signal Extraction

Justin Klaar

Accelerating Real-Time Voltage Imaging

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER AND EMBEDDED SYSTEMS ENGINEERING

by

Justin Klaar
born in Zoetermeer, the Netherlands

Computer Engineering Research Group
Department of Quantum & Computer Engineering
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Erasmus MC
Dr. Molewaterplein 40, 3015 GD
Rotterdam, the Netherlands
www.erasmusmc.nl

©2025 Justin Klaar. All rights reserved.

Accelerating Real-Time Voltage Imaging

Author:	Justin Klaar
Student id:	4381130

Abstract

This thesis addresses the critical challenge of real-time spike extraction from high-throughput voltage imaging data, often overwhelming existing analysis pipelines despite advancements in genetically encoded voltage indicators and imaging hardware. A novel hybrid processing pipeline is presented, integrating the strengths of state-of-the-art systems, designed for unified offline and online analysis with enhanced modularity and reproducibility. Through architectural optimizations, including streaming-compatible design, algorithmic enhancements like GPU-accelerated filtering, and a robust calibration framework, processing speed and signal fidelity was significantly improved. Our evaluation demonstrates real-time throughput of 500 frames per second at 680×680 with 8 ms latency and up to 1000 frames per second at 480×480 with 16 ms latency using a Nvidia Tesla V100, notably reducing startup times and improving deployability via a containerized environment. While revealing motion-correction as a persistent bottleneck and the inherent latency-throughput trade-off, this work provides a scalable, accurate, and user-friendly solution that bridges the gap between fast data acquisition and real-time analytical capability, paving the way for next-generation closed-loop neuroscience experiments.

Thesis Committee:

Chair:	C. Strydis, Faculty EEMCS, TU Delft
Committee Member:	M. Pertijs, Faculty EEMCS, TU Delft
Committee Member:	S. Wong Member, Faculty EEMCS, TU Delft

Preface

I would like to express my sincere gratitude to my thesis advisor, Dr. Christos Strydis, with the Neuroscience department of the Erasmus MC and with the Quantum and Computer Engineering department of the TU Delft. His guidance, support, and expertise have been invaluable throughout the course of this project.

Special thanks also go to Rui Silva, PhD candidate at Erasmus MC, whose research originally posed the problem I tackled in this thesis. I had the pleasure of working closely with him, and his input and collaboration were instrumental to the progress and completion of this work.

I am also grateful to all members of the Neuro Support Group at Erasmus MC for the supportive and collegial atmosphere they created, which made the research process all the more engaging and enjoyable.

Finally, I wish to thank my partner, Teddie, as well as my family and friends for their unwavering support, encouragement, and patience during this journey.

Justin Klaar
Delft, the Netherlands
June 16, 2025

Contents

List of Figures

List of Tables

List of Acronyms

CPU Central Processing Unit. 12

FFT Fast Fourier Transform. 16, 99

GEVI Genetically Encoded Voltage Indicator. 7

GPU Graphics Processing Unit. 10, 12

NMF Non-negative Matrix Factorization. 18, 104

RAM Random Access Memory. 12

ROI Region of Interest. 11

VSD Voltage-Sensitive Dye. 7

Chapter 1

Introduction

Understanding how neurons communicate is one of the core challenges in neuroscience. Over the past few decades, researchers have developed increasingly sophisticated tools to monitor neural activity, culminating in methods that offer both high resolution and scalability. Among these, calcium imaging became a transformative technique, providing an accessible way to observe population-level neural dynamics in vivo.

Calcium imaging works by detecting calcium influxes linked to action potentials, serving as an indirect, but powerful, proxy for neuronal firing. Its appeal lies in its ability to capture activity across large neural populations using relatively simple optical systems. However, the approach has a fundamental limitation: the calcium signal spans over hundreds of milliseconds. This slow temporal resolution makes it poorly suited for capturing fast neural dynamics such as high-frequency activity, rapid oscillations, or subthreshold voltage changes.

To overcome these constraints, researchers turned to Genetically Encoded Voltage Indicators (GEVIs). These fluorescent proteins respond directly to changes in membrane potential, enabling the optical detection of rapid electrical events with millisecond precision. With continuous improvements in their brightness, speed, and photostability, GEVIs have matured into powerful tools for studying fast, fine-grained neural activity [3].

As molecular sensors and imaging hardware have advanced, neuroscience has entered an era of unprecedented observational capabilities. It is now possible to monitor neural activity across large populations in real time, with both high spatial and temporal fidelity. Within this context, voltage imaging, powered by GEVIs, has emerged as a leading method for capturing rapid neural events, complementing and extending the capabilities of calcium imaging.

In this work, a distinction is made between real-time and online processing paradigms. Real-time processing is defined as the ability to analyze imaging data at the same rate at which it is acquired, thereby ensuring that each frame is processed with negligible latency relative to acquisition. In contrast, online processing refers to a sequential, frame-by-frame analysis in which data are handled incrementally as they arrive, but without a requirement to match the acquisition speed. While both methods avoid traditional batch processing, only real-time processing satisfies the temporal constraints necessary for closed-loop experimental paradigms. It should be noted that although all real-time systems operate in an online manner, the converse is not necessarily true.

At the core of this activity are spikes; brief, all-or-none electrical impulses known as action potentials. These form the basic units of neural communication, encoding information

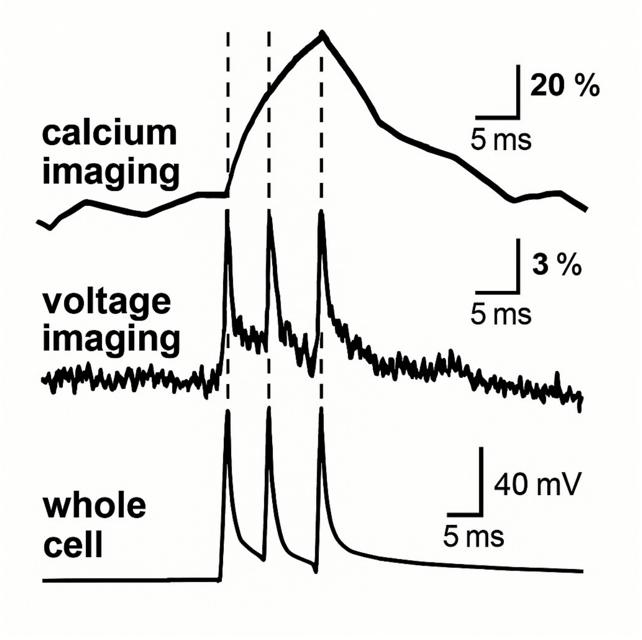


Figure 1.1: Comparison of signal profiles captured using calcium imaging, voltage imaging, and whole-cell electrophysiology in response to action potentials. Voltage imaging provides a much closer match to the true electrical activity (as measured by electrophysiology), while calcium signals exhibit slower, temporally smeared responses due to indicator kinetics. Adapted from [2].

in their timing and frequency. Accurately detecting spikes is essential for decoding how the brain processes information, controls behavior, and adapts to experience.

1.1 Motivation and Problem Statement

Modern voltage imaging systems are capable of recording from hundreds or thousands of neurons at kilohertz frame rates. This capability however, comes at the cost of massive data throughput, often reaching terabytes per hour, which presents significant computational challenges. Extracting meaningful events like spikes from these data requires complex processing pipelines capable of motion-correction, denoising, segmentation, and spike inference. These tasks are further complicated by low signal-to-noise ratios, motion artifacts and photobleaching.

Although tools such as VolPy [4], Suite2p [5] and GPU-accelerated frameworks like Bando et al. [6] have pushed the field forward, many still falter at frame rates beyond 500 Frames Per Second (FPS) with high-resolution data (e.g., fields larger than 200×300 pixels). In such cases, real-time (zero-latency) analysis becomes infeasible, threatening to bottleneck the utility of voltage imaging despite its unprecedented window into fast neural dynamics.

This limitation is especially problematic in closed-loop neuroscience experiments, where detected spikes must trigger immediate feedback interventions, such as optogenetic stimulation or behavior, contingent cues, within milliseconds [7]. Here, both high precision and true real-time inference are not just beneficial but essential.

Accordingly, this thesis seeks to address the following research question:
How can a real-time voltage imaging pipeline be optimized to ensure lossless frame processing, maximize spatial resolution, and maintain high inference accuracy under strict computational constraints?

1.2 Thesis Goal and Contributions

This thesis addresses the dual challenge of achieving real-time performance and high signal fidelity in spike extraction from voltage imaging data. It presents a scalable processing pipeline capable of handling the data volumes generated by next-generation imaging systems while producing cleaner, better interpretable spike outputs.

Today's experiments frequently operate at frame rates above 500 FPS and across large fields of view, enabled by advanced GEVIs and volumetric imaging techniques [8, 9]. Yet, existing analysis pipelines often falter under these conditions. This work introduces a methodology that not only alleviates these performance bottlenecks but also improves the reliability of the inferred spike traces.

1.2.1 Thesis Objectives

To meet the above mentioned challenges, the thesis pursues the following specific objectives:

- **Benchmark existing pipelines:** Evaluate prominent existing pipelines on standard lab hardware to identify key performance and scalability limits.
- **Explore optimization strategies:** Investigate low-level accelerations, such as custom kernels and memory-efficient streaming, to improve processing speed.
- **Design a modular architecture:** Create a flexible, component-based pipeline that can adapt to diverse experimental setups and computational constraints.
- **Develop a benchmarking suite:** Build standardized tools for evaluating both performance (e.g., FPS, latency) and signal quality (e.g., spike fidelity, robustness to noise).
- **Validate in real time:** Demonstrate lossless spike extraction at $\geq$500 FPS in conditions mimicking real-world experiments on 300$\times$300-pixel frames, achieving end-to-end processing latency below 50 ms.
- **Conduct an extensive survey of state-of-the-art tools and building blocks:** Review existing software components, libraries, and frameworks relevant to voltage imaging pipelines.

- **Develop a portable software stack:** Create a fully containerized, dependency-managed pipeline that can be deployed on any system, from single-GPU workstations to cloud instances.

1.2.2 Thesis Contributions

This thesis makes the following contributions to the field of voltage imaging analysis:

- **Pipeline Bottleneck Analysis:** A comprehensive evaluation of the performance limits of existing spike extraction tools. (Chapter 3 and 4)
- **Hybrid Optimization Framework:** A suite of GPU-based and architectural improvements tailored for high-speed data processing. (Chapter 5)
- **Signal Quality Enhancements:** Techniques that improve the consistency of extracted spike traces. (Section 4.1.6)
- **Scalable Real-Time System:** A prototype capable of real-time spike inference from high-throughput imaging data without frame loss. (sectionr 6.5.2)
- **Live Deployment Capability:** The first end-to-end workflow enabling immediate deployment during ongoing experiments. Unlike existing pipelines that rely on offline file-based inputs, the proposed system operates directly on live data streams, reflecting realistic experimental conditions requiring online spike monitoring. (Chapter 4)
- **Open Benchmark Toolkit:** A reproducible, publicly available toolkit for evaluating spike extraction pipelines across diverse conditions. (Section 5.4.3)
- **Empirical Validation:** Quantitative results demonstrating system performance on both synthetic and real datasets, including application to closed-loop use cases. (Chapter 6)

Collectively, these contributions bridge the gap between acquisition speed and analytical capability, enabling a new generation of adaptive, real-time neuroscience experiments.

1.2.3 Thesis Outline

The remainder of this document is organized as follows. Chapter 2 reviews the biological foundations of voltage imaging, characterizes its unique data challenges, and surveys the hardware and real-time processing constraints that drive pipeline design (Sections 2.1-2.7). Chapter 3 presents a detailed literature survey of existing voltage-imaging workflows, covering motion correction, segmentation, photobleaching correction, spike detection, and thresholding, and synthesizes common architectural patterns (Sections 3.1-3.5).

Building on this survey, Chapter 4 benchmarks and compares state-of-the-art pipelines, identifies their complementary strengths, and motivates a unified, hybrid pipeline architecture (Sections 4.1-4.2). Chapter 4 also introduces targeted algorithmic improvements, such as vignette filtering and GPU–accelerated median computation, driven by empirical profiling (Section 5.3). Chapter 5 then details the software implementation of both offline and online modes, including startup-latency optimizations, profiling-driven refinements, modular control, and containerized deployment (Sections 5.1-5.4).

Chapter 6 evaluates the hybrid pipeline using benchmark datasets and representative hardware, quantifying segmentation and spike-detection accuracy, end–to–end latency in offline and streaming modes, and memory usage (Sections 6.1-6.6). Chapter 7 summarizes the key contributions, highlights performance achievements on Tesla V100 and H100 platforms, discusses observed trade-offs and limitations, and outlines directions for future work (Sections 7.1-7.4).

Chapter 2

Background

This chapter establishes the foundational context for the development of a real-time, scalable voltage imaging pipeline. It begins by tracing the evolution from calcium to voltage imaging and outlines the advantages of the latter in high-temporal-resolution neuroscience. It then explores the challenges specific to voltage imaging data, including low signal-to-noise ratio, biological noise, and motion artifacts, and concludes by framing the technical constraints that motivate the system design introduced in later chapters.

2.1 From Calcium to Voltage Imaging

Calcium imaging has been a staple in systems neuroscience, providing indirect measures of neuronal activity through the detection of intracellular calcium transients. While it has enabled significant discoveries about population dynamics and connectivity, its temporal resolution is inherently limited by the kinetics of calcium binding and unbinding. Typical calcium indicators respond on the scale of hundreds of milliseconds, rendering them unsuitable for capturing precise spike timing, fast oscillatory dynamics, or real-time applications [10].

Voltage imaging, in contrast, uses Voltage-Sensitive Dye (VSD) or Genetically Encoded Voltage Indicator (GEVI) to record membrane potential changes directly. This modality offers temporal resolutions on the order of 1–2 ms, allowing researchers to resolve action potentials, subthreshold fluctuations, and fast synaptic inputs with much greater fidelity. It thus provides access to previously unobservable dynamics, enabling closed-loop experiments, synaptic inference, and real-time neural decoding [6, 10].

A side-by-side comparison of calcium and voltage signals is shown in Figure 1.1, highlighting the sharp temporal contrast between the slow, integrated calcium response and the precise spike-resolving capability of voltage imaging. These properties position voltage imaging as a critical modality for real-time applications that demand millisecond precision in spike detection.

2.2 Characteristics and Challenges of Voltage Imaging Data

Modern voltage imaging systems are capable of capturing data at frame rates ranging from 500 to 1000 FPS, with spatial resolutions spanning from several hundred to thousands of pixels per dimension. These capabilities yield datasets of considerable size; individual recordings can exceed 100 GB within just a few minutes. While this high temporal and spatial resolution provides unprecedented insight into neural dynamics, it also introduces significant challenges that complicate data processing and analysis.

2.2.1 Photobleaching and Baseline Drift

As fluorescent indicators are exposed to continuous illumination their brightness decays, a phenomenon known as photobleaching. This leads to slow drifts in baseline fluorescence, often compounded by physiological fluctuations such as metabolic activity or vascular dynamics. Without proper correction, these slow shifts can obscure neuronal signals and degrade the performance of spike detection algorithms. Effective photobleaching correction is therefore essential to maintain a stable baseline for downstream analysis.

2.2.2 Motion Artifacts

Despite the use of head-fixation techniques or mechanical stabilizers, in-vivo recordings are susceptible to motion artifacts resulting from heartbeat, respiration, and micro-movements of the tissue. These movements cause spatial misalignments across frames, leading to distorted trace data and complicating neuron tracking. Even subpixel shifts can introduce significant artifacts, particularly in densely packed neuronal regions. As such, robust and low-latency motion-correction is a critical component of any real-time imaging pipeline.

2.2.3 Low Signal-to-Noise Ratio (SNR)

In contrast to calcium transients which typically have large amplitude changes, voltage signals exhibit considerably lower Signal-to-Noise Ratio, SNR, as illustrated in Figure 1.1. Neural activity is often masked by background fluorescence, thermal noise, and shot noise originating from the imaging sensor. This low SNR poses a major challenge for detecting individual spikes, especially in real-time contexts, where computationally intensive averaging or manual curation methods used in offline processing are not feasible [11].

2.2.4 Neuronal Spiking Dynamics

From a biological perspective, neurons may fire in bursts or as isolated spikes, with interspike intervals ranging from a few milliseconds to several hundred milliseconds depending on the cell type and experimental conditions. Following a spike, neurons enter a brief refractory period during which no additional spikes can occur. These dynamics impose strict temporal constraints on detection algorithms, which must resolve the onset of spikes with sub-frame precision while avoiding false positives caused by noise or filtering artifacts.

Figure 2.1 illustrates the shape of a typical action potential, highlighting the sharp onset, peak, and refractory dynamics that any voltage imaging pipeline must resolve accurately.

2.3 Pipeline and Hardware Constraints

Despite the growing demand for real-time analysis in voltage imaging, such as closed-loop stimulation or adaptive imaging control, most existing pipelines remain optimized for of-

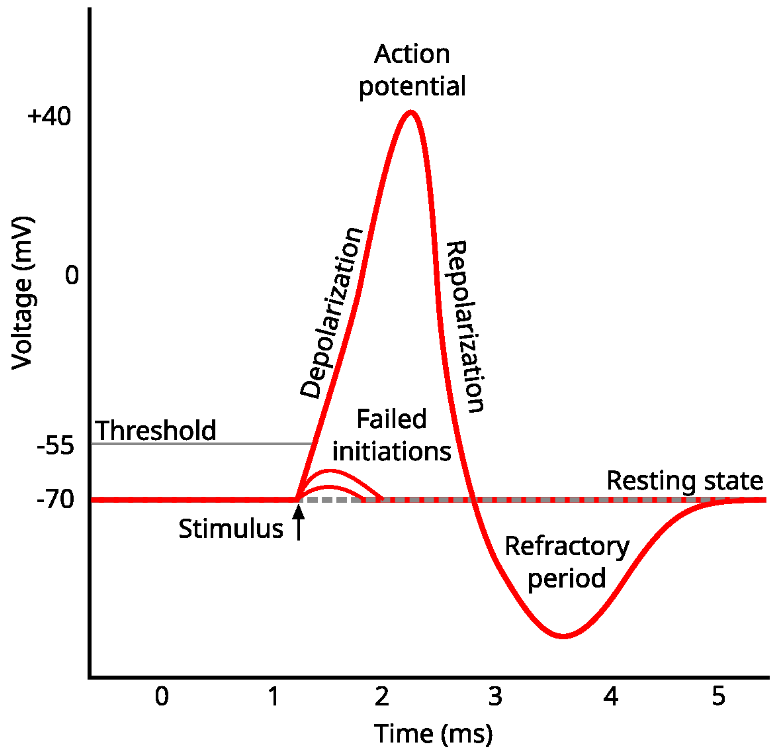


Figure 2.1: Canonical action potential waveform showing rapid depolarization, peak, repolarization, and the hyperpolarization phase. The refractory period prevents immediate reactivation, which poses timing constraints for spike detection algorithms. Image from [12]

fline post-hoc processing. Few pipelines are capable of sustaining data throughput at full camera frame rates (e.g., 500-1000 FPS) without incurring frame loss or latencies exceeding tens of milliseconds, which can undermine closed-loop experimental integrity. This challenge is compounded by hardware limitations common in many laboratories, where access to high-performance GPUs or real-time operating systems is limited, and software stacks are often heterogeneous.

Designing a spike inference pipeline that ensures high accuracy, low latency, and real-time throughput under such constraints remains a core and unsolved challenge in the field. This motivates the system architecture introduced in the following chapter

2.4 Design Drivers from Real-World Experiments

The requirements and constraints outlined above are shaped by the practical realities of modern experimental neuroscience. Many cutting-edge studies rely on closed-loop feedback systems where neural signals must be detected and acted upon within milliseconds. These include optogenetic stimulation triggered by specific patterns of neural activity, adaptive imaging that adjusts acquisition parameters based on brain state, and behaviorally contingent interventions.

Adding to this complexity is the limited experimental window: each animal subject, typically a mouse, must undergo a delicate surgical procedure to expose the brain and apply the VSD or GEVI. Once the mouse is head-fixed under the microscope, **every second becomes valuable**. The time window is constrained not only by experimental goals and ethical considerations, but also by the physiological burden on the animal. Despite being immobilized, physiological processes such as breathing and heartbeat introduce motion artifacts and noise into the recordings.

In this context, experimental success depends heavily on the ability to extract reliable neural signals quickly and under constrained conditions. Most laboratories work with a single Graphics Processing Unit (GPU) workstation and finite memory resources, often in mixed-platform environments (Linux and Windows). There is typically no access to remote clusters or cloud-based computing during live experiments. Therefore, it is essential that the pipeline is lightweight, robust, and real-time capable,designed to interface directly with acquisition hardware and complete all processing on-site.

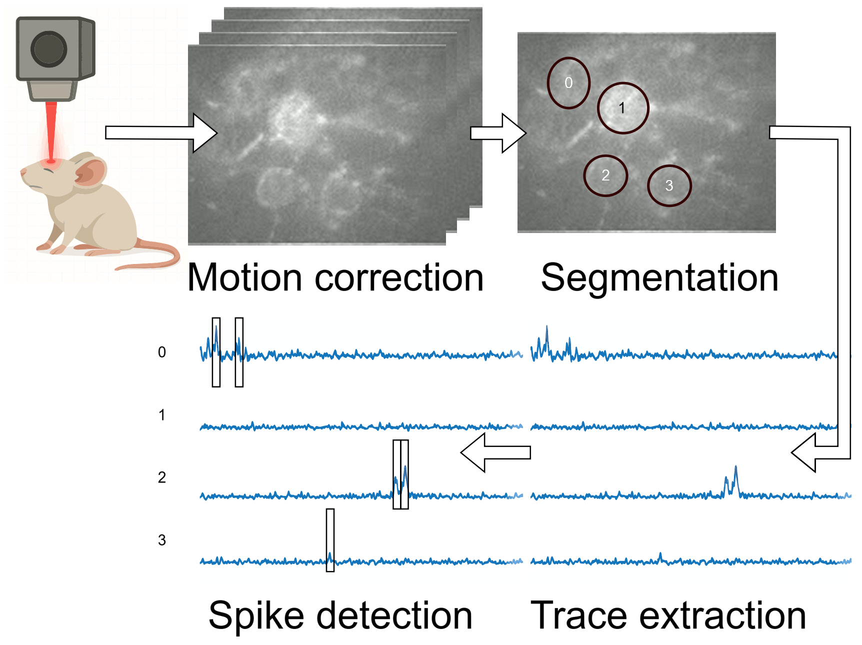


Figure 2.2: Voltage imaging experimental pipeline, including acquisition and real-time processing stages.

2.4.1 Typical Experimental Setup

In a standard voltage imaging experiment, a mouse is placed under a high-speed fluorescence microscope, where it is head-fixed to minimize large-scale movement. A craniotomy is performed beforehand to expose the brain, and a VSD or GEVI is used to label neurons in the region of interest. The imaging setup includes a high-speed camera capable of capturing fluorescence signals at rates up to 1000 FPS, producing substantial data throughput.

These raw signals must be processed in real time to extract spike-resolved information that is meaningful for downstream applications such as closed-loop stimulation. The processing pipeline typically includes motion-correction, neuron segmentation, photo-bleaching, denoising, and spike detection , executed entirely within the constraints of a single GPU-equipped workstation.

As illustrated in Figure 2.2, each stage of the pipeline serves a distinct and critical role in transforming raw fluorescence into usable neural data:

- **Motion-Correction:** Compensates for brain movement caused by breathing, heartbeat, or micro-motions. This stage ensures that fluorescence changes reflect actual neural activity rather than physical displacement.
- **Neuron Segmentation:** Isolates individual neurons or regions of interest (ROIs) from the surrounding background, enabling targeted signal extraction and reducing interference from overlapping sources.
- **ROI Trace Extraction:** After motion-correction and segmentation, the defined Region of Interest (ROI)s are applied to the stabilized image frames to compute fluorescence traces. This involves averaging or summing pixel intensities within each ROI across time, producing one signal trace per neuron for further analysis.
- **Photobleaching Correction:** Stabilizes fluorescence intensity over time by correcting for signal decay and baseline drift.
- **Denoising:** Suppresses random noise while preserving meaningful transients, which is essential for accurately detecting both spikes and subthreshold voltage fluctuations in low-SNR conditions.
- **Spike Detection:** Identifies fast, spike-like voltage transients that correspond to neuronal firing events. Accurate detection is critical for real-time interpretation and feedback control.

This multi-stage process enables researchers to extract actionable neural data for experiments requiring real-time interaction with the brain.

2.5 Performance Constraints in Real-Time Processing

Enabling real-time voltage imaging analysis poses significant computational and architectural challenges. Modern imaging systems generate hundreds to thousands of frames per second, often with high spatial resolution and low signal-to-noise ratios. Processing this data stream fast enough to support closed-loop experiments requires minimal latency, high memory throughput, and robust inference algorithms, all operating in parallel.

A real-time pipeline must process each frame as it arrives, with tight constraints on end-to-end latency. This includes overheads from motion-correction, trace extraction, and spike detection, as well as data transfers from camera to system memory and GPU. Delays introduced by batch processing, GPU memory limits, or inefficient I/O can quickly exceed timing budgets, disrupting feedback-based experiments like optogenetic control or adaptive stimulation [13, 14].

To guide system design, this thesis defines a set of target performance constraints based on the capabilities of a mid-range lab workstation. These constraints represent practical, achievable goals for enabling real-time operation in experimental neuroscience environments:

Constraint	**Target Value**
Minimum Processing Rate	500 FPS @ 300 $\times$300 px
Maximum Latency per Frame	50 ms
Frame Drop Tolerance	$\leq$ 0.01% (1 in 10,000 frames)
Maximum Random Access Memory (RAM) Usage	64 GB
GPU Requirement	Single gpu (cross-platform)

Table 2.1: Target performance constraints based on available lab hardware.

2.6 Parallelism and High-Speed Processing

Achieving real-time processing of voltage imaging data necessitates leveraging parallel computing paradigms. Amdahl's Law provides insight into the potential speedup of a task using multiple processors, highlighting that the maximum improvement is limited by the portion of the task that cannot be parallelized [15]. The law is commonly expressed as:

$$S = \frac{1}{(1-P)+\frac{P}{N}} \tag{2.1}$$

where S is the theoretical speedup, P is the proportion of the program that can be parallelized, and N is the number of processing units. This relationship emphasizes that even with infinite processing resources, speedup is ultimately capped by the serial fraction of the task.

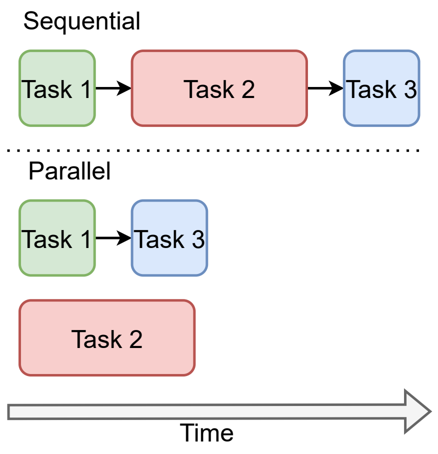
Modern processors, including multicore Central Processing Units (CPUs) and GPU, offer distinct architectural trade-offs when applied to Amdahl's Law. CPUs are designed for low-latency, sequential execution with a few high-performance cores, and thus they excel at the serial portions of a pipeline, those bottlenecked by control logic, memory access, or conditionals. GPUs, by contrast, are designed for high-throughput parallelism, featuring thousands of lightweight cores that handle large-scale, data-parallel tasks efficiently. As a result, GPUs maximize the impact of the parallel fraction P in Amdahl's Law, achieving massive speedups where P is high and the workload is well-suited to SIMD (single instruction, multiple data) execution.

While parallelism helps overcome short-term performance barriers, long-term trends in hardware advancement have also played a crucial role. According to Moore's Law, the number of transistors for same size integrated circuits doubles approximately every two years, effectively doubling processing performance over the same interval. Although this trend has slowed in recent years, it remains a guiding principle for expectations around compute capacity and has fueled the proliferation of high-throughput, GPU-based computing environments critical for voltage imaging workloads.

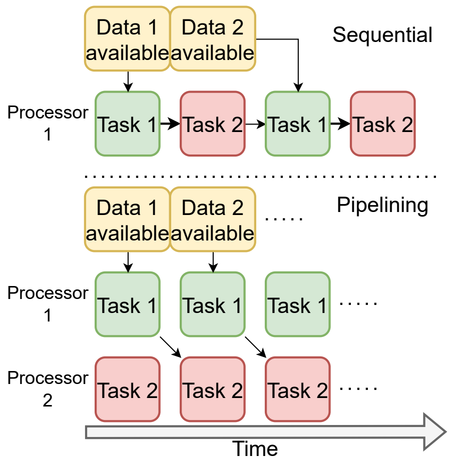
2.6.1 Pipelining in Imaging Workflows

Pipelining refers to the organization of tasks into sequential stages that operate concurrently on different pieces of data. In real-time voltage imaging, pipelining allows incoming frames to be processed in a staggered fashion: while one stage completes motion-correction on frame n, another stage might be applying spike detection to frame $n-1$, and yet another is acquiring frame $n+1$.

This architectural approach is effective for maximizing hardware usage and reducing idle time across CPU and GPU components. It also enables smoother streaming and smaller per-frame latency, which are crucial for closed-loop and interactive experiments. A conceptual overview of pipelining in general-purpose processing systems is shown in Figure 2.3.



(a) Parallelism: executing the same operation on multiple data simultaneously. Ideal for tasks like GPU-based matrix operations.



(b) Pipelining: overlapping different processing stages across multiple data items. Common in imaging workflows.

Figure 2.3: Conceptual comparison of parallelism and pipelined execution. Parallelism applies the multiple operations concurrently across multiple data elements to boost throughput, pipelining divides a workflow into sequential stages that overlap in time.

2.6.2 GPU Acceleration in Voltage Imaging Pipelines

GPUs are particularly well-suited for real-time voltage imaging due to their architecture, which allows thousands of lightweight threads to run in parallel. Unlike cpu, which are optimized for sequential logic and complex control flow, GPUs are built around massive numbers of simple, parallel processing units that shine when applied to uniform, data-parallel workloads such as:

- **Image registration:** Operations like cross-correlation, affine transforms, or optical flow that can be applied per-pixel or block-wise across frames.
- **Filtering and enhancement:** Convolutional filters (e.g., Gaussian blur, Sobel edge detection), morphological operations, and spatial transforms.
- **Fourier and frequency-domain analysis:** Fast Fourier Transforms (FFT) and related operations, which are heavily parallelizable and used in denoising, compression, and texture analysis.
- **Frame-wise operations in video processing:** Processing streams of image data (e.g., temporal filtering, change detection, frame differencing) in parallel.

Whereas CPUs help reduce the (1-P) term in Amdahl's Law by speeding up serial bottlenecks, GPUs dramatically shrink the P/N term when applied to parallelizable components. A balanced pipeline architecture that exploits both is necessary for real-time performance.

Low-Level vs. High-Level GPU Programming

GPU resources can be accessed at multiple abstraction levels. At the lowest level, developers can write CUDA (Compute Unified Device Architecture) or OpenCL (Open Computing Language) kernels, specialized code designed to run directly on GPU hardware. These frameworks allow fine-grained control over GPU memory management and compute operations, enabling developers to maximize efficiency and performance. This level of access is especially valuable for custom pipelines with strict performance constraints. However, writing and optimizing low-level kernels requires deep knowledge of GPU architecture, memory hierarchies, and parallel programming models, as well as significant development time.

At a higher level, libraries such as TensorFlow, PyTorch, CuPy, and RAPIDS expose GPU acceleration through convenient APIs. These frameworks abstract away memory management and parallelization, allowing developers to implement high-performance code quickly. While this may sacrifice some control and efficiency, the benefits of rapid prototyping and portability often outweigh the trade-offs in early or general-purpose pipeline design.

2.7 Preview: Implementation Mechanics and Relevance

This chapter has outlined the biological, experimental, and hardware constraints that shape the design of a real-time voltage imaging pipeline. While these high-level considerations motivate the system architecture introduced in the next chapter, many of the key trade-offs emerge only at the implementation level, where processing delays, memory usage, and algorithmic structure directly impact performance.

To preview these lower-level concerns, this section provides a high-level summary of two representative pipelines: **FIOLA** and **VOLTAGE**. Both are designed for high-throughput voltage imaging, but they adopt distinct architectural strategies. FIOLA emphasizes real-time, frame-by-frame execution with fixed ROIs and lightweight computation, making it well-suited for online deployment. VOLTAGE, by contrast, relies on modular stages, GPU acceleration, and deep-learning–based segmentation, trading off latency for flexibility and spatial accuracy.

A key component in both pipelines is **motion-correction**, which aligns frames to a reference template. This step is critical to downstream performance and is later shown to become a computational bottleneck in the hybrid system design (see Section 6.5.1). For this reason, motion-correction receives particular attention in both the implementation and optimization chapters.

A concise explanation of FIOLA's motion-correction is included below, along with a brief summary of the computational complexity of all components. Readers interested in the full analysis are referred to Appendix B.

2.7.1 FIOLA: Fluorescence Imaging OnLine Analysis

FIOLA is an online voltage imaging pipeline designed for low-latency, real-time processing. After an initial offline calibration, it processes frames independently using a GPU-resident computational graph, achieving millisecond-scale latencies. FIOLA supports frame-by-frame motion-correction, fluorescence trace extraction, and spike inference without full video buffering, making it suitable for closed-loop experimental paradigms.

The structure and execution flow of FIOLA's offline phase is illustrated in Figure 2.4, which highlights each step leading up to online operation and the data artifacts generated at each stage.

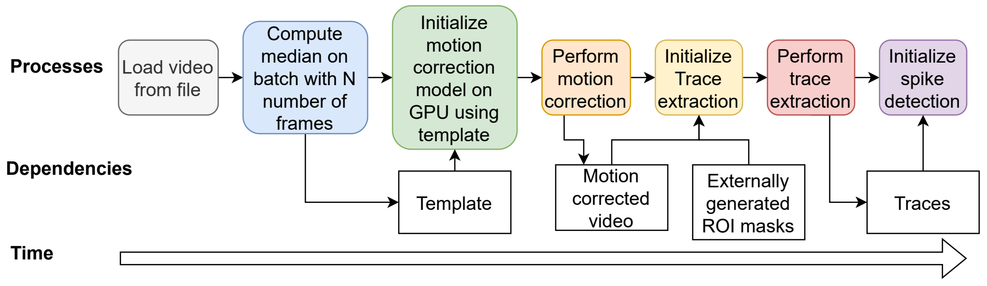

Processes
Dependencies
Time
Load video from file
Compute median on batch with N number of frames
Initialize motion correction model on GPU using template
Perform motion correction
Initialize Trace extraction
Perform trace extraction
Initialize spike detection
Template
Motion corrected video
Externally generated ROI masks
Traces

Figure 2.4: Timeline of the FIOLA pipelines offline initialization process. Key stages include motion template creation, segmentation or mask loading, matrix factorization into signal and background components, and spike template generation. These outputs are used in real-time during online streaming.

While FIOLA is described as an online pipeline, it does not operate in a fully streaming mode. Specifically, after offline calibration, the full imaging file is first loaded into GPU memory. Only then does FIOLA process each frame independently or in small batches without further buffering delays. This distinction is important in the context of real-time systems that must operate continuously on incoming data. Figure 2.5 illustrates the sequence of operations following offline initialization, capturing the data flow and GPU execution model during the online phase.

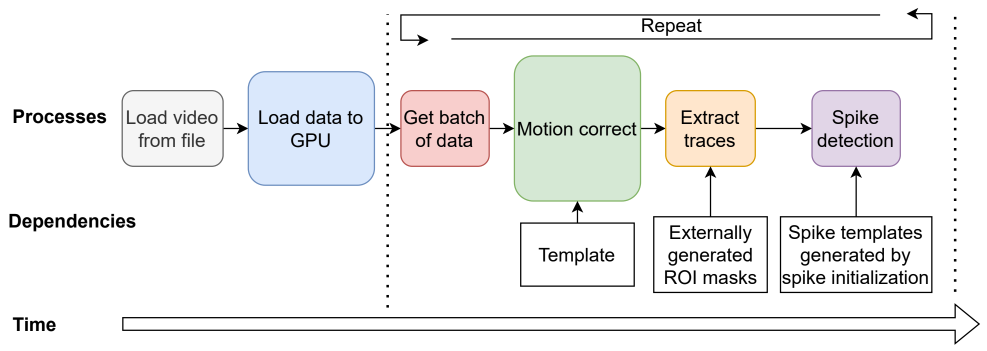


Figure 2.5: Execution timeline of FIOLA's online phase. After the entire imaging file is loaded into GPU memory, frame-by-frame or batch-wise processing is performed with low latency. This timeline illustrates how motion-correction, fluorescence extraction, and spike inference are pipelined without further I/O or video buffering.

Motion-Correction

FIOLA performs rigid motion-correction by aligning each incoming frame to a fixed reference template (typically obtained during initialization) using frequency-domain normalized cross-correlation (ZNCC). Crucially, once the required subpixel shift is determined, it is applied directly in the Fourier domain via phase modulation, avoiding interpolation in the spatial domain.

The algorithm proceeds as follows:

1. **Fourier Cross-Correlation:** The incoming frame I_t and the template T are first zero-mean normalized and transformed via a 3D Fast Fourier Transform (FFT), although the complex values remain at 0. Their normalized cross-correlation is computed in the frequency domain using:

$$\mathrm{ZNCC} = \left|\mathcal{F}^{-1}\left(\mathcal{F}(I_t)\cdot\mathcal{F}(T)^*\right)\right| \tag{2.2}$$

   where $\mathcal{F}$ denotes the FFT and * is the complex conjugate.

2. **Subpixel Shift Estimation:** The peak of the ZNCC map is extracted, and Gaussian interpolation is applied in log-space to estimate subpixel displacements $(\Delta x, \Delta y)$.

3. **Phase-Based Shift Application:** The shift is then applied in the frequency domain by modulating the phase of the FFT of I_t. This approach exploits the Fourier shift theorem and avoids explicit interpolation:

$$I_t' = \mathcal{F}^{-1}\left(\mathcal{F}(I_t)\cdot e^{-2\pi i(u\Delta x+v\Delta y)}\right) \tag{2.3}$$

where (u,v) are spatial frequency coordinates.

Figure 2.6 illustrates this pipeline, including the FFT-based cross-correlation, subpixel estimation, and frequency-domain shift correction.

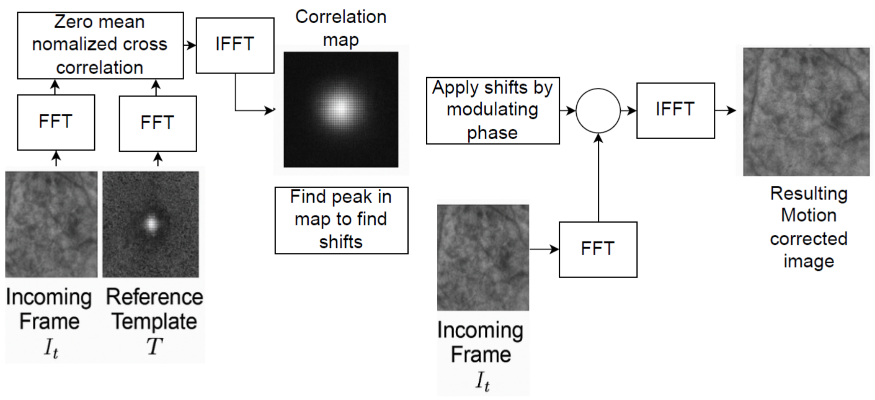

Zero mean nomalized cross correlation
IFFT
Correlation map
FFT
FFT
Incoming Frame I_t
Reference Template T
Find peak in map to find shifts
Apply shifts by modulating phase
IFFT
FFT
Incoming Frame I_t
Resulting Motion corrected image

Figure 2.6: Overview of FIOLAs motion-correction process. Each incoming frame is normalized and transformed via FFT, then aligned to a reference template using frequency-domain cross-correlation. Subpixel shifts are estimated and applied in Fourier space.

Template Construction: The accuracy of the motion-correction step depends on the quality of the reference template. FIOLA constructs this during an initialization phase by computing the pixelwise temporal median over a batch of frames (usually 5000 or more). This median operation is performed on the CPU and has a complexity of $O(NL\log L)$ for N pixels and L frames.

Computational Complexity: The per-frame cost is dominated by FFT operations and subpixel refinement:

- FFT and inverse FFT: $O(N\log N)$
- Phase-based shift: $O(N)$
- Subpixel peak estimation: $O(1)$ per frame

All steps are parallelized on GPU using TensorFlow's 'fft3d' and 'ifft3d' operations.

Implementation Details: Motion-correction is implemented as a custom TensorFlow layer using 'tf.signal.fft3d' and 'ifft3d'. The cross-correlation, phase shifting, and inverse FFT are executed on the GPU for high throughput. Precomputed frequency-domain template values are reused across frames, and the system maintains a batched pipeline for real-time operation. Shift values are computed per frame and passed downstream for trace alignment. The FFT-based correction supports subpixel accuracy without the artifacts that typically arise from spatial-domain interpolation.

Summary: FIOLA achieves accurate, low-latency spike detection by integrating photobleaching correction, real-time matched filtering, and adaptive thresholding. Its streaming-friendly design makes it ideal for closed-loop voltage imaging experiments, and its computational efficiency allows scalability to thousands of neurons in real time.

2.7.2 VOLTAGE: A Fast, Modular Voltage Imaging Pipeline

VOLTAGE (Voltage imaging pipeline) is an open-source framework developed by the MIT Media Lab and collaborators for high-speed analysis of voltage imaging data [6]. It is designed to operate in real time or faster-than-recording-time on a single high-end workstation equipped with GPUs.

The pipeline consists of six main stages:

1. **Motion-Correction:** Stabilizes video data by removing global 2D translational shifts.
2. **Shading Correction:** Compensates for lighting intensity changes due to movement and illumination variability.
3. **Preprocessing:** Aggregates raw frames into summary images to suppress noise.
4. **Segmentation:** Applies a CNN to summary images and demixes overlapping ROIs via local Non-negative Matrix Factorization (NMF).
5. **Trace Extraction:** Computes fluorescence traces for each ROI over time.
6. **Spike Detection:** Identifies spikes on the extracted traces using thresholding or template matching.

Motion-correction, shading-correction and segmentation are designed for GPU acceleration and multi-GPU parallelism. VOLTAGE is implemented in C++/CUDA and Python and can be extended or replaced at any stage.

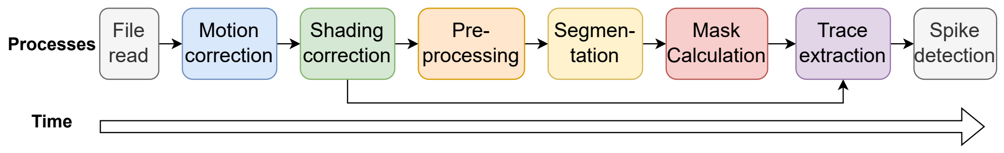


Figure 2.7: Timeline of VOLTAGE's pipeline execution. Each stage uses the output of the previous one as its input, creating a sequential dependency. The only exception is trace extraction, which depends not only on the ROI mask but also directly on the output of shading correction.

2.7.3 Computational Complexity

Computational complexity refers to how the resource requirements of an algorithm, typically time or memory, scale with input size. It is expressed using Big-O notation, which captures the dominant terms as the input grows large. In the context of voltage imaging pipelines, complexity helps predict how performance will degrade with increasing resolution, number of frames, or neurons. Understanding the asymptotic cost of each component is critical when optimizing for real-time constraints or large-scale datasets.

The following summarizes the theoretical complexity of major stages in both FIOLA and VOLTAGE pipelines. Notation is shared unless otherwise specified:

F	$=$ Total number of frames	d, N	$=$ Pixels per frame
K	$=$ Number of neurons (ROIs)	$\bar{r}$	$=$ Average ROI size
L	$=$ Temporal window length	T_{init}	$=$ Initialization frame count
N_{iter}	$=$ Number of optimization iterations	P	$=$ Patches per frame (VOLTAGE)
s	$=$ Patch size	M	$=$ Shift candidates per patch
S	$=$ Temporal segments (VOLTAGE)	p	$=$ Pixels in NMF region
R	$=$ Overlapping regions demixed (NMF)		

Pipeline	**Component**	**Time Complexity**	**Space Complexity**
FIOLA	Motion-Correction	$O(N \log N \cdot F)$	$O(N)$
FIOLA	Template Construction	$O(N \cdot L \log L)$	$O(N \cdot L)$
FIOLA	Trace Extraction (offline)	$O(N_{\text{iter}} \cdot d \cdot K \cdot T_{\text{init}})$	$O(N \cdot K + K^2)$
FIOLA	Trace Extraction (online)	$O(F \cdot (d + K))$	$O(K)$
FIOLA	Spike Detection	$O(F \cdot K \cdot L)$	$O(K \cdot F)$
VOLTAGE	Motion-Correction	$O(F \cdot P \cdot M \cdot s^2) \approx O(F \cdot N \cdot M)$	$O(N + P \cdot M)$
VOLTAGE	Summary Image Extraction	$O(S \cdot d \cdot L) \approx O(F \cdot d)$	$O(N \cdot L)$
VOLTAGE	Segmentation (CNN)	$O(S \cdot \mathrm{P} \cdot s^2)$	$O(p^2 \cdot c)$
VOLTAGE	Segmentation (NMF Demixing)	$O(R \cdot p \cdot K \cdot S)$	$O(N \cdot T + N \cdot K)$
VOLTAGE	Trace Extraction	$O(F \cdot K \cdot \bar{r})$	$O(K \cdot F)$
VOLTAGE	Spike Detection	$O(F \cdot K)$	$O(K)$

Table 2.2: Time and space complexity of FIOLA and VOLTAGE processing stages. Complexity is expressed asymptotically in terms of key variables.

A complete derivation and additional discussion of scaling behavior appear in Appendix B.

Chapter 3

Related Works

The previous chapter outlined the biological, experimental, and computational challenges that make real-time voltage imaging both valuable and difficult. High frame rates, low signal-to-noise ratios, and motion artifacts create a demanding environment in which analysis pipelines must operate with high precision and minimal latency. However, addressing these challenges is not solely a matter of experimental design, it also requires algorithmic solutions that are robust, scalable, and compatible with real-time constraints. To design a pipeline that meets these stringent demands, it is essential to understand the current landscape of available methods and how they address, or fail to addressthese constraints.

This chapter surveys the existing literature on voltage imaging analysis, with a focus on pipeline components that support real-time performance. It builds on a meta-analysis conducted by Rui Silva [16], which categorized modern pipelines into a series of standardized processing stages: motion-correction, neuron segmentation, denoising, photobleaching correction, spike detection, and thresholding, as shown in Figure 2.2. These stages provide a framework for comparative evaluation and guide the architectural choices made in this thesis.

While Silva's analysis focused on spike inference accuracy and processing speed, the increasing demands of real-time and high-throughput neuroscience necessitate a closer examination of each method's computational efficiency and scalability. In this review, these pipeline stages are revisited with an emphasis on runtime performance, memory usage, online compatibility, and hardware acceleration.

Algorithmic complexity is assessed qualitatively, based on factors such as streaming versus batch-mode operation, parallelizability, and GPU integration. Scalability is evaluated in terms of performance on large datasets, latency during streaming, and feasibility for real-time deployment.

Rather than reviewing individual tools in isolation, architectural trade-offs inherent to each stage of the pipeline are examined. Special attention is given to methods that retain high accuracy, consistent with Silva's benchmarks, while advancing scalability and efficiency. Notable examples include GPU-accelerated and real-time capable pipelines such as FIOLA [17] and VOLTAGE [6], which exemplify state-of-the-art performance in voltage imaging analysis.

3.1 Motion-Correction

Motion-correction is essential in voltage imaging, particularly for in vivo recordings where tissue movement can distort fluorescence signals. Most pipelines use 2D correction due to the prevalence of one-photon and two-photon imaging [16]. For example, NoRMCorre [18] applies piecewise rigid registration using normalized cross-correlation (NCC), but its reliance on full-frame operations and high memory usage limits its scalability.

Older tools like TurboReg [19] and TrackMate [20] rely on global registration methods, techniques that align an entire image at once by comparing all pixels using measures like mean squared error (MSE) or by tracking specific objects. However, their lack of GPU support and the need to load complete datasets make them unsuitable for large-scale imaging.

Recent methods such as FIOLA [17] and VOLTAGE [6] improve scalability by leveraging GPU acceleration. FIOLA employs frequency-domain NCC using TensorFlow to achieve real-time alignment at up to 1000 FPS [16], though at lower spatial resolution and big batch sizes. VOLTAGE uses zero-mean NCC with intensity normalization for robust correction, also using GPU acceleration. However, it is designed for Linux-based systems only, which limits its platform compatibility.

Additionally, Platisa et al. [21] introduced a region-restricted approach that computes NCC on a small region of interest (ROI) and applies the correction globally. This method speeds up processing with minimal accuracy loss, but its closed-source nature restricts wider use.

In summary, while NoRMCorre is well-established, newer approaches like FIOLA and VOLTAGE offer improved scalability through GPU-friendly designs. The region-restricted method further demonstrates that narrowing the correlation scope can enhance performance with little overhead. A summary of key motion-correction methods and their trade-offs is provided in Table 3.1.

3.2 Neuron Segmentation

Neuron segmentation is a major challenge for real-time and high-throughput workflows. In many studies, manual ROI annotation is used, especially in mesoscale or subcellular imaging where signal shapes vary, but this approach does not scale well and can hurt reproducibility.

VolPy [4] uses PCA and iterative thresholding, which works reasonably well for mid-sized datasets in offline settings. However, its full-frame decomposition limits scalability. In contrast, FIOLA [17] applies ridge regression and real-time clustering with GPU support, avoiding the need for batch processing and enabling real-time execution. Similarly, the VOLTAGE pipeline [6] employs U-Net segmentation on summary images of short video segments, offering low preprocessing overhead and GPU-accelerated inference that makes it effective for real-time use [16].

Other strategies include threshold-based masking [21], which is fast but struggles in noisy or crowded recordings, and manual annotation [22, 23], which is common in small-scale studies but not scalable. Suite2p [5] uses graph-based clustering and scales well on

Table 3.1: Comparison of motion-correction methods used in voltage imaging pipelines.

Reference	**Method**	**Computational Complexity**	**Scalability Notes**
Cai et al., 2021 (VolPy)	NoRMCorre (rigid)	Moderate	Scales poorly with 1000+ FPS due to sequential correction; GPU acceleration helps
Cai et al., 2023 (FIOLA)	Online piecewise rigid + deep learning	Moderate-High	Designed for online use; highly scalable with GPU optimization
Griffiths et al., 2020	Real-time 3D registration with feature matching	High	Excellent for volumetric data, but high memory/compute demand
Xie et al., 2021	Frame-to-template correlation (rigid)	Low–Moderate	Fast but less robust to local warping or drift
Fan et al., 2022	Rolling average reference alignment	Low	Lightweight and fast, but sensitive to drift and noise
Evans et al., 2023	Suite2p motion-correction	Moderate	Optimized for throughput (Python + GPU support)
Abdelfattah et al., 2023	Frame registration via high-pass reference	Moderate	Real-time capable when windowed
Wong-Campos et al., 2023	Subpixel rigid alignment + spline smoothing	Moderate	Effective for high-speed light-field imaging
Kaifosh et al., 2014	Hidden Markov model line correction	High	Not optimized for large-scale or real-time use
Platisa et al., 2022	2D fast Fourier transform registration	Low–Moderate	Very fast; scales well for 2D datasets
Sabater et al., 2021	TurboReg (ImageJ)	Low	Sufficient for small data but not real-time suitable
Bando et al., 2024	ZNCC (zero-mean normalized cross correlation)	Moderate	Robust against brightness fluctuations; moderately fast
Kannan et al., 2022	Rigid-body + photobleach-aware correction	Moderate–High	Tailored for GEVI imaging with variable intensity

GPU hardware, though it is not optimized for real-time processing. Signal-informed methods by Kannan et al. [24] and Liu et al. [25] leverage GEVI characteristics like polarity or template matching to improve specificity, though they depend on predefined models.

More experimental approaches include a neural network-based real-time segmentation tool by Wang et al. [26] and a probabilistic refinement step for noisy data by Brooks et al. [27]. Additionally, Weber et al. [28] and Tian et al. [29] integrate segmentation directly into volumetric imaging protocols, which can be useful but are computationally intensive and hardware-dependent.

Overall, while accuracy is crucial, efficient processing such as summary-image methods and GPU inference is equally important. Scalable segmentation pipelines like FIOLA and VOLTAGE demonstrate that high performance and accuracy can go hand in hand when the computational framework is optimized for throughput. A comparative overview of segmentation methods is provided in Table 3.2.

Table 3.2: Comparison of neuron segmentation methods used in voltage imaging pipelines.

Reference	**Method**	**Computational Complexity**	**Scalability Notes**
Cai et al., 2021 (VolPy)	PCA-based ROI extraction with iterative thresholding	Moderate	Handles medium-sized datasets well; not real-time optimized
Weber et al., 2023	Confocal segmentation with axial multiplexing	High	Powerful for multiplane imaging; hardware-integrated segmentation
Platisa et al., 2022	Threshold-based masking on filtered images	Low	Simple but effective for 1P/2P data; not ideal for crowded fields
Evans et al., 2023 (Suite2p)	ROI detection using graph-based clustering	Moderate	Efficient on large datasets; GPU-friendly
Tian T., 2022	High-speed multiphoton segmentation using volumetric scans	High	Very effective for dense 3D data; computationally intensive
Liu Z., 2022 (Cell)	Fast indicator + ROI matching using templates	Moderate	Designed for GEVI fluorescence stability; robust in vivo
Xiao S., 2021	Targeted illumination with adaptive segmentation	Moderate	Tailored for confocal setups; moderately scalable
Adam Y., 2019	Manual ROI segmentation via anatomical landmarks	Low	Not scalable for large datasets or automation
Kannan M., 2022	GEVI-aware segmentation using signal polarity	Moderate	Works well in dual-color setups; scalable to $\sim$100 ROIs
Wang Z., 2023 (bioRxiv)	Machine learning-based real-time ROI extraction	High	High-speed and large-scale compatible; still experimental
Brooks F.P., 2024	Probabilistic ROI refinement from denoised signals	Moderate	Balanced performance; good for noisy or dim signals
Piatkevich K., 2019	Max-projection ROI outlines from kHz 2P scans	High	Optimized for kilohertz recordings; compute-heavy
Kaifosh P., 2014	Manual mask generation and Python-based tools	Low	Scripting allows flexibility, but requires user interaction
Canales A., 2023	Combined ROI extraction with dendritic tracing	Moderate–High	Precise but slow; not suitable for real-time
Shu W.C., 2021	Automated clustering of synchronous events	Moderate	Segments active neurons based on spiking synchrony
Hayward R.F., 2023	Model-based segmentation using hybrid GEVIs	Moderate	Targeted for hybrid probes; not widely generalizable

Continued on next page

Table 3.2 – *Continued from previous page*

Reference	**Method**	**Computational Complexity**	**Scalability Notes**
Bando et al., 2024 (VOLTAGE)	U-Net segmentation on summary images from short videos	Moderate–High	Designed for real-time performance; leverages GPU acceleration and efficient preprocessing

3.3 Photobleaching Correction and Spike Detection

Photobleaching correction is vital for maintaining accurate $\Delta F/F$ baselines during long recordings. The $\Delta F/F$ baseline – calculated as the change in fluorescence (ΔF) divided by the baseline fluorescence (F) – provides a normalized measure that compensates for variability in the initial signal. Silva notes that photobleaching and baseline drift often occur together [16]. Common strategies to correct these effects include exponential fitting [30, 21], high-pass filters [31], and rolling averages [32]. Exponential models are computationally lightweight and preserve signal frequency content but assume a uniform decay rate, while filtering methods are fast yet risk dampening subthreshold oscillations.

Because drift and bleaching tend to co-occur, most pipelines address them together using smoothing or normalization. However, varying decay rates across regions of interest (ROIs) can complicate parameterization and reduce the method's generalizability. A comparative summary of photobleaching correction methods is shown in Table 3.3.

Denoising often serves as a preprocessing step before spike detection, particularly in recordings with low signal-to-noise ratios. Methods range from lightweight techniques like Gaussian filtering and PCA/SVD-based denoising [4], to advanced deep learning-based models such as DeepVID [33], which offer high-quality noise suppression with GPU acceleration. Table 3.4 compares commonly used denoising approaches in terms of scalability and computational complexity.

Spike detection is typically performed on thresholded $\Delta F/F$ signals or on z-scored traces. Z-scored traces are signals normalized by subtracting their mean and dividing by their standard deviation, resulting in data with a zero mean and unit variance, which helps highlight deviations. For instance, VolPy [4] combines template matching with statistical modeling to achieve good accuracy, though it lacks real-time capability. In contrast, FIOLA [17] supports real-time inference using online ridge regression with adaptive thresholding that updates noise estimates dynamically. Ridge regression is a linear regression technique that adds a regularization term to mitigate the effects of multicollinearity, thereby preventing overfitting.

Simpler detection strategies, like the per-pixel z-score thresholding employed in VOLTAGE [6], are GPU-compatible and scale well, although they may struggle in noisy conditions. Platisa et al. [21] use fixed post-denoising thresholds, which work effectively for stable ROIs, while Villette et al. [34] enhance specificity by deconvolving traces before thresholding, though this adds complexity.

Other methods include Suite2p [5], which uses autoregressive deconvolution with scalable inference. This approach balances computational efficiency and throughput, making it

Table 3.3: Comparison of photobleaching correction methods used in voltage imaging pipelines.

Reference	**Method**	**Computational Complexity**	**Scalability Notes**
Cai et al., 2023 (FIOLA)	Online baseline tracking with exponential smoothing	Low–Moderate	Designed for continuous correction in streaming pipelines
David Wong-Campos et al., 2023	Spline fitting over temporal intensity decay	Moderate	Supports long-term recordings with gradual intensity loss
Lu X., 2023 (Nature Comms)	Framewise correction using control region references	Moderate	Works well in wide-field applications; assumes stable background
Kannan M., 2022	Photobleach-aware rigid registration	Moderate	Designed for GEVI signals with dynamic ranges
Fan L., 2022 (Cell)	Rolling average and baseline drift removal	Low	Simple, efficient method compatible with online analysis
Platisa J., 2022	Intensity normalization via ROI baselining	Low	Suitable for stable 2P recordings with limited drift
Sabater G.V., 2021	Exponential decay modeling per ROI	Moderate	Effective for correcting multi-ROI voltage imaging data
Zhang Jie, 2023	Polynomial detrending combined with global correction	Moderate	Designed for robust performance across different brightness regimes
Bando et al., 2024	Summary image normalization using sliding windows	Low–Moderate	Implicit correction through frame-averaged summaries improves robustness over time

well-suited for large datasets. In contrast, the probabilistic model by Brooks et al. [27] effectively handles noisy or ambiguous signals but is computationally more intensive, which may limit its real-time use. Meanwhile, synchrony-based detectors [35] focus on co-activated neurons, offering a lighter computational load; however, they tend to miss isolated spikes, reducing sensitivity in certain scenarios.

A summary of spike detection, denoising, and photobleaching correction methods is provided in Tables 3.5, 3.4, and 3.3, respectively.

Table 3.4: Comparison of denoising methods used in voltage imaging pipelines.

Reference	**Method**	**Computational Complexity**	**Scalability Notes**
Cai et al., 2023 (FIOLA)	Gaussian filtering and ridge regression	Moderate	Real-time capable; performs well under high frame rates
Cai et al., 2021 (VolPy)	PCA/SVD-based denoising	Moderate	Works well for medium-size datasets; not ideal for online processing
Eom et al., 2023	Deep learning-based denoising (statistical prediction)	High	Accurate and fast with GPU acceleration; deep model training required
Liu et al., 2024 (DeepVID v2)	Self-supervised spatiotemporal filtering	High	State-of-the-art performance; scales well with batch GPU processing
Wang B., 2024 (CellMincer)	Self-supervised denoising using residual modeling	High	Tailored for GEVI noise patterns; well-optimized for speed
Xie M.E., 2021	Bandpass filtering + local averaging	Low	Lightweight method ideal for real-time, small-volume pipelines
Kim S., 2023	Optical segmentation-based compressed reconstruction	High	Novel and efficient, but not broadly validated
Fan L., 2020	Temporal smoothing using rolling filters	Low	Common in 1P pipelines; fast but may obscure fast transients
Ma Yihe, 2023	Spatial averaging and temporal whitening	Moderate	Balanced in noise suppression and signal preservation
Brooks F.P., 2024	Signal decomposition and filtering	Moderate	Designed to recover low-SNR spike trains in voltage imaging
Bando et al., 2024	Patch-wise background subtraction and Z-score filtering	Moderate–High	Works in real-time; optimized with GPU-accelerated ZNCC registration and summary images

Table 3.5: Comparison of spike detection methods used in voltage imaging pipelines.

Reference	Method	Computational Complexity	Scalability Notes
Cai et al., 2023 (FIOLA)	Online ridge regression with adaptive thresholds	Moderate	Real-time compatible and scalable for dense neural populations
Cai et al., 2021 (VolPy)	Template matching with noise modeling	Moderate	Accurate but slower; not designed for real-time or large-scale datasets
Bando et al., 2024	Simple Z-score thresholding on pixel traces	Low	Fast, real-time suitable; sacrifices some accuracy for speed
Villette V., 2019	Threshold crossing on deconvolved signals	Low–Moderate	Works reliably for high-gain indicators; not adaptive to baseline shifts
Platisa J., 2022	Voltage trace thresholding post-denoising	Low	Efficient for 2P recordings; best suited for stable ROIs
Evans S., 2023 (Suite2p)	Deconvolution with autoregressive modeling	Moderate	High-throughput compatible; optimized for batch pipelines
Shu W.C., 2021	Event-based clustering from synchrony detection	Moderate	Detects spikes based on co-activity patterns; not robust for isolated events
Ma Yihe, 2023	Derivative-based spike detection	Low	Simple, efficient, but sensitive to noise fluctuations
Xiao S., 2021	Local maxima identification with threshold gating	Low	Well-suited for stable imaging conditions; not robust to large intensity changes
Zhang Jie, 2023	Combined ROI and global threshold strategy	Moderate	Balances speed and accuracy; scalable with moderate computational load
Li B., 2020	Template-based spike detection with matched filters	Moderate	Good signal isolation; slower when dealing with overlapping or low-SNR signals
Brooks F.P., 2024	Filtered trace analysis using probabilistic spike events	Moderate	Optimized for noisy voltage recordings; tunable precision and recall

3.4 Thresholding Strategies

Thresholding methods are crucial for both spike detection accuracy and computational efficiency. The simplest method, fixed z-score thresholding [6], is fast and scales easily but can struggle when baseline levels change. In contrast, FIOLA's adaptive method [17] adjusts to changing signal variance, which improves reliability at the cost of additional computations.

Hybrid approaches, like those from Zhang et al. [36] and Sabater et al. [37], use a two-stage process. They first apply a global threshold to quickly remove obvious noise, then refine detection using local thresholds. This extra step increases the computational load compared to fixed thresholding, but it also improves precision in variable signal conditions.

Brooks et al. [27] go further by implementing soft-threshold models that dynamically adjust sensitivity under low signal-to-noise conditions. Although these models require more processing, modern GPUs can handle the extra load efficiently. Meanwhile, Kim et al. [38] propose using compressed readouts with spatially varying thresholds. This method can reduce the overall data size and computational cost, but its efficiency depends on how well the compression is optimized.

A comparative overview of these thresholding strategies, including their computational costs and scalability, is provided in Table 3.6.

Overall, integrating photobleaching correction, baseline stabilization, and noise modeling, as seen in pipelines like FIOLA [17], VolPy [4], and the approach by Brooks et al. [27], leads to better accuracy and scalability. These methods are designed to take advantage of modern hardware, such as GPUs, to manage the increased computational demands without sacrificing real-time performance on large datasets.

Table 3.6: Comparison of thresholding methods used in voltage imaging pipelines.

Reference	**Method**	**Computational Complexity**	**Scalability Notes**
Cai et al., 2023 (FIOLA)	Adaptive thresholding based on signal noise estimates	Moderate	Automatically adjusts to signal variations; well-suited for online pipelines
Bando et al., 2024	Fixed Z-score threshold per pixel trace	Low	Extremely fast and scalable; limited flexibility in noisy or variable baselines
Xiao S., 2021	Local intensity thresholding with dynamic adjustment	Low	Works for stable recording conditions; not ideal for drift-prone signals
Fan L., 2022	Simple global thresholding on filtered data	Low	Effective in low-noise contexts; lacks adaptivity
Lu X., 2023	Polynomial baseline subtraction with fixed thresholds	Moderate	Performs well with consistent background patterns
Brooks F.P., 2024	Signal decomposition with soft-threshold detection	Moderate	Designed for flexible precision tuning in low-SNR environments
Zhang Jie, 2023	Combined ROI and global threshold strategy	Moderate	Balances performance and simplicity for high-throughput datasets
Sabater G.V., 2021	ROI-specific threshold tuning based on baseline stats	Moderate	Better precision than global thresholds; slightly slower in large-scale applications
Shu W.C., 2021	Synchrony-based detection thresholding	Moderate	Thresholds based on population-level activation; less sensitive to single spikes
Kim S., 2023	Compressed readout with spatially varying thresholds	High	Novel approach; suited for compact encoding systems, still experimental

3.5 Summary and Integration

This chapter has examined voltage imaging pipelines by evaluating both spike inference accuracy, as established in Rui Silva's meta-analysis [16], and the computational complexity and scalability of each processing stage. These factors are essential for enabling real-time, large-scale neural data analysis and inform the methodological choices made in this thesis.

Although each processing stage, motion-correction, segmentation, photobleaching correction, spike detection, denoising, and thresholding, has been optimized independently in the literature, the real power of voltage imaging emerges when the best elements are combined into a cohesive, hybrid pipeline. The next chapter will show how the modular choices below can be interwoven to leverage FIOLA's online efficiency and VOLTAGE's segmentation accuracy, yielding a system that integrates full pipeline capability, from motion-correction through spike detection, while remaining fully online.

Table 3.7 presents a best-in-class modular pipeline, summarizing the most effective method selected for each processing stage. These selections reflect a balance between algorithmic accuracy, computational cost, and real-time performance. The table consolidates discussion points from throughout the review, for instance, motion-correction via FIOLA [17], U-Net-based segmentation in VOLTAGE [6], and spike detection using adaptive thresholding and online regression.

Table 3.7: Best-in-class modular voltage imaging pipeline by processing stage.

Stage	**Chosen Method**	**Reference**	**Why This Choice?**
Motion-Correction	Online piecewise rigid + deep learning	Cai et al., 2023 (FIOLA)	Real-time capable, GPU-optimized, robust across large-scale datasets
Segmentation	U-Net on summary images	Bando et al., 2024 (VOLTAGE)	Accurate with overlapping neurons, real-time ready, scales well with fast imaging
Photobleaching	Online baseline tracking with smoothing	Cai et al., 2023 (FIOLA)	Lightweight, adaptive to drift, works well with online streaming
Denoising	Self-supervised spatiotemporal filtering	Liu et al., 2024 (DeepVID)	State-of-the-art denoising without needing ground truth; highly generalizable
Thresholding	Adaptive thresholding based on signal noise	Cai et al., 2023 (FIOLA)	Dynamically adjusts to signal conditions; supports high-throughput pipelines
Spike Detection	Online ridge regression + adaptive inference	Cai et al., 2023 (FIOLA)	Fast, continuous inference; integrates well with adaptive thresholding

Key trade-offs between accuracy, runtime performance, and hardware demands were highlighted across each processing stage. While denoising via self-supervised spatiotemporal filtering (e.g., Liu et al.'s DeepVID [33]) achieves state-of-the-art noise suppression, it was excluded from the proposed hybrid pipeline due to its batch-mode architecture, high GPU load, and incompatibility with real-time streaming. DeepVID relies on multi-frame input and transformer-based processing, introducing latency and memory requirements that conflict with the low-latency, modular goals of this pipeline.

In contrast, VOLTAGE [6] avoids separate denoising altogether by applying lightweight temporal smoothing only during the generation of summary images for segmentation. This approach reduces noise sufficiently for spatial analysis while remaining compatible with online, frame-wise processing. Because segmentation in VOLTAGE operates directly on these smoothed summary images using a GPU-accelerated U-Net, it aligns better with the hybrid pipeline's goals of modularity, scalability, and real-time performance.

Disclaimer: All performance figures reported in this chapter were obtained under varying (and often unreported) hardware configurations or software dependencies. As a result, numerical runtimes and memory requirements are not directly comparable, our goal is a theory-based relative comparison.

This theoretical comparison sets up the next chapter, where FIOLA and VOLTAGE will be profiled on the same hardware, quantifying empirical performance, and then constructing a hybrid pipeline that combines the low-latency motion-correction of FIOLA with the high-accuracy segmentation of VOLTAGE.

Chapter 4

Pipeline design

To translate the findings from the literature into a practical, high-performance solution, it is essential to move beyond isolated method descriptions and examine how complete pipelines behave in practice. While individual components, such as segmentation or motion-correction, may perform well on paper, their real-world efficiency and compatibility depend heavily on implementation details, memory usage, inter-stage latency, and implementation-specific bottlenecks.

Accordingly, this chapter begins with a detailed breakdown of the FIOLA and VOLTAGE pipelines. Comparable stages are profiled independently to identify computational bottlenecks and performance trade-offs across different tasks and hardware environments. The analysis is grounded in practical experiments and includes both step-wise profiling and hardware-level comparisons.

Building on the insights from Section 4.1, the second half of the chapter introduces a unified hybrid architecture that combines the most effective components from each system. This design balances accuracy, latency, and usability by separating pipeline responsibilities into two complementary workflows:

- **Offline pipeline:** Focused on preprocessing and asset generation, including motion templates and segmentation masks, with high-throughput, batch-oriented execution.
- **Online pipeline:** Optimized for online inference with strict latency bounds, enabling closed-loop and feedback-driven neuroscience experiments.

The remaining sections describe these designs in detail and introduce a series of targeted enhancements, such as vignette filtering and a focused analysis of the primary bottlenecks limiting online processing performance.

4.1 Profiling State-of-the-Art Pipelines

This section evaluates the computational performance and accuracy of two leading voltage imaging pipelines, FIOLA and VOLTAGE, with the goal of identifying runtime bottlenecks and informing the hybrid pipeline design presented in later sections. This profiling serves two key purposes: first, it reveals which components deliver the best empirical performance; second, it lays the groundwork for a modular, hybrid pipeline architecture that combines the most efficient elements of each system.

4.1.1 Setup and Experimental Conditions

Profiling was conducted using the benchmark dataset published by Bando et al. [1], specifically the HPC2 subset, which serves as the representative dataset throughout this study. This dataset includes pre-aligned ground truth data and manually curated masks for validation, which are used throughout this chapter.

The dataset consists of image sequences with a spatial resolution of 116×496 pixels and a general temporal depth of 15,000 frames. If longer sequences are encountered, they are capped at 15,000 frames to ensure consistency across evaluations.

All profiling was conducted on the CUBE server at the Neuroscience department, as shown in Table 4.1, using the open-source Miniconda distribution with Python 3.8. The FIOLA pipeline was executed using TensorFlow 2.4.1 with CUDA 12 and CUDNN 8, while the VOLTAGE pipeline required CUDA 11 to satisfy its compatibility constraints. GPU execution was run in eager mode, and deterministic behavior was enforced across all runs. A batch size of 1000 frames was used for both pipelines unless otherwise stated.

Table 4.1: Hardware specifications for FIOLA and VOLTAGE profiling experiments.

Component	**CUBE Server**
CPU	AMD EPYC 7551 (32-core, 2.56 GHz)
GPU	NVIDIA Tesla V100, 32 GB
RAM	128 GB DDR4 (Kingston)
Disk	Dell NVMe P4600, 3.2 TB
OS	Ubuntu 20.04.6 LTS

Despite being publicly available, both FIOLA and VOLTAGE required substantial effort to install and configure. Compatibility mismatches, missing dependencies, and undocumented environment assumptions led to significant delays during the initial benchmarking phase. In some cases, direct source code modifications were required to enable stable execution on modern GPUs. These setup challenges motivated the later development of a fully containerized hybrid pipeline, described in Section 5.1, to ensure future reproducibility and ease of deployment.

Profiling Methodology: Runtime measurements were obtained by inserting timers before and after each stage of execution, with all steps included (I/O, preprocessing, model inference, and output writing). To ensure consistency, each profiling run was repeated five times per file in the dataset, and results were averaged. Internal validation was performed using `cProfile`, the built-in Python profiler, to confirm timer accuracy, and `nvprof`, the offical NVIDIA profiler, was used to inspect GPU behavior, particularly for the motion-correction modules. Model warm-up was performed by executing a full run once and discarding its output. Memory usage was not recorded as part of this profiling.

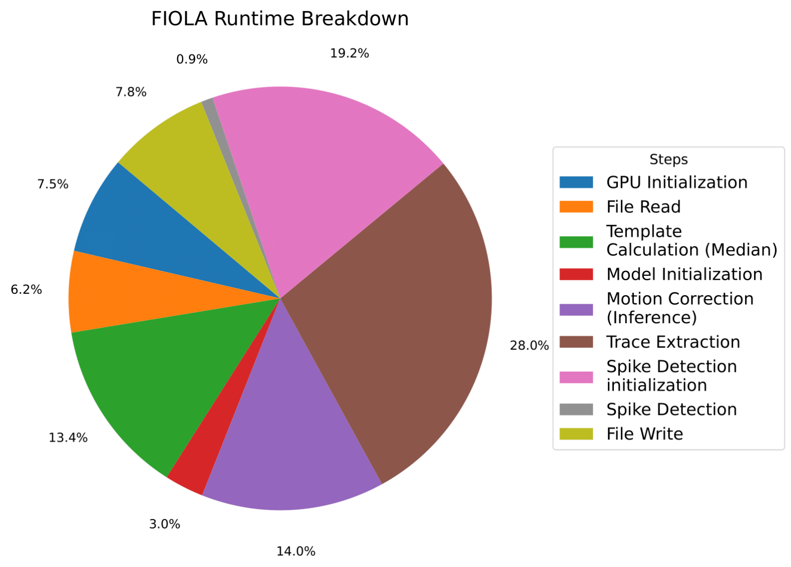


Figure 4.1: FIOLA runtime breakdown (percent per step).

Table 4.2: FIOLA runtime breakdown (average across all videos; offline).

Step	**Time (s)**	**Time (%)**
GPU Initialization	8.24	7.5
File Read	6.85	6.2
Template Calculation (Median)	14.72	13.4
Model Initialization	3.29	3.0
Motion-Correction (Inference)	15.46	14.0
Trace Extraction	30.78	28.0
Spike Detection initialization	21.15	19.2
Spike Detection	0.98	0.9
File Write	8.62	7.8
Total	110.09	**100**

4.1.2 FIOLA Pipeline Breakdown

FIOLA is an online-capable pipeline designed for accelerated processing of calcium and voltage imaging data [17]. Built using GPU-optimized TensorFlow models, it processes image sequences frame-by-frame with minimal buffering and latency. However, the version profiled here reflects its **offline configuration**. At the time of benchmarking, full online integration was not yet available; thus, modifications were later introduced to enable frame-wise streaming and online profiling.

Profiling Insights

Table 4.2 summarizes the average runtime of each pipeline stage in FIOLA's offline, batch-processing mode, measured over our test dataset. Although FIOLA appears to support online streaming, it actually first loads the entire dataset onto the GPU before any processing begins. This GPU pre-loading stage scales linearly with the number of frames and introduces several important implications:

- It adds upfront latency, delaying the onset of processing, especially problematic for short recordings or low-latency applications.
- It imposes a high memory footprint, limiting scalability to longer recordings or higher resolutions due to GPU memory constraints.
- It breaks compatibility with true real-time operation, where frames arrive sequentially and cannot be preloaded.

As such, the reported runtimes reflect isolated per-stage computational cost, but under conditions that bypass the complexities of real-time buffering, incremental data flow, and pipeline concurrency (see Section 4.2.3 for those details). A few key observations emerge:

- **Per–frame vs. front-loaded work:**
  - *Per–frame stages*: Motion-correction, trace extraction, spike detection, and GPU transfer scale directly with the number of video frames. Each additional frame incurs the full cost of these operations, so their cumulative runtime grows linearly with frame count.
  - *Front-loaded stages*: Template calculation and spike–detector initialization are performed once at startup using a fixed set of initial frames. Their cost does not increase with video duration. However, in FIOLA's offline mode, GPU preloading acts as a hybrid stage, performed once but scaling with video length, thereby contributing both to initialization latency and memory pressure.
- **I/O overhead:** Disk reads and writes account for roughly 14% of total runtime. While not dominant on a high-performance NVMe SSD, I/O can become a limiting factor on slower storage systems or when simultaneously processing multiple datasets.
- **Implications for real-time operation:** To achieve low-latency, high-throughput streaming, it is critical to avoid full preloading and instead implement efficient per-frame transfers and buffering strategies. Front-loaded costs can be amortized over long sessions, but true real-time performance demands that processing keeps pace with acquisition without holding the full dataset in memory.

For a detailed analysis of the algorithmic and memory complexity associated with each pipeline component, including asymptotic bounds and scaling behavior, refer to Table 2.2. This table provides a side-by-side comparison of FIOLA and VOLTAGE stages in terms of their computational and spatial demands, offering insight into their scalability and real-time feasibility.

4.1.3 VOLTAGE Pipeline Breakdown

This section reports a detailed runtime profile obtained under controlled conditions, matching the FIOLA profiling setup. The pipeline performs the operations in sequential order as shown in Table 4.3, which shows average execution times over the entire dataset. The segmentation model, motion-correction and the evaluation of the pipeline steps dominate computational cost.

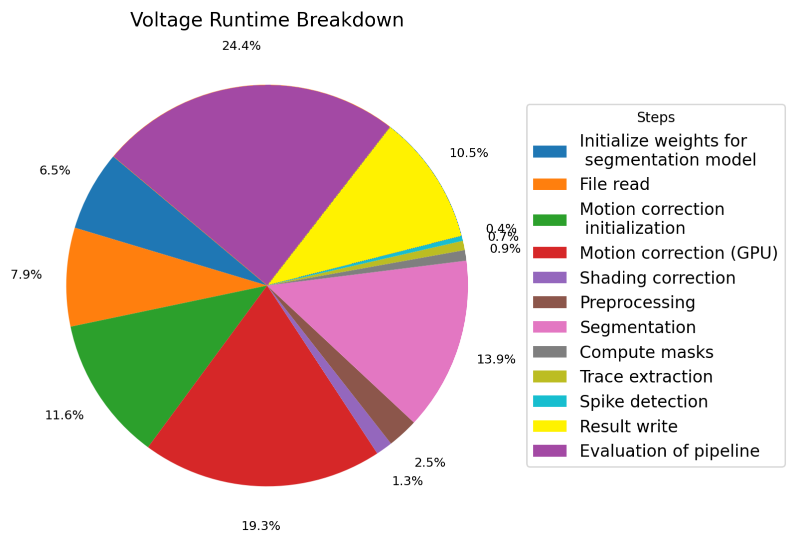


Figure 4.2: Overview of VOLTAGE runtimes per component

Table 4.3: Runtime breakdown of the VOLTAGE pipeline averaged across all videos in the HPC2 dataset published by Bando et al. [1].

Step	**Time (s)**	**Time (%)**
Initialize weights for segmentation model	5.34	6.46
File read	6.56	7.94
Motion-correction initialization	9.59	11.61
Motion-correction (GPU)	15.98	19.34
Shading correction	1.11	1.34
Preprocessing	2.04	2.47
Segmentation	11.51	13.93
Compute masks	0.70	0.85
Trace extraction	0.62	0.75
Spike detection	0.32	0.39
Result write	8.71	10.54
Evaluation of pipeline	20.14	24.38
Total	**82.62**	**100.00**

Profiling Insights: VOLTAGE Pipeline

Table 4.3 summarizes the average runtime per processing stage in the VOLTAGE pipeline, measured across all videos in the HPC2 dataset under the same controlled conditions used for FIOLA profiling. Although the operations are executed sequentially, their computational impact varies considerably across stages. Figure 4.2 provides a visual overview of relative costs. A few key observations emerge:

Profiling Insights: VOLTAGE Pipeline

Table 4.3 summarizes the average runtime per processing stage in the VOLTAGE pipeline, measured across all videos in the HPC2 dataset under the same controlled conditions used for FIOLA profiling. Although the pipeline executes its steps sequentially, their computational characteristics differ in terms of batching behavior, scalability, and real-time feasibility. Figure 4.2 illustrates relative runtime contributions. A few key observations emerge:

- **Per-batch vs. front-loaded stages:**
  - *Per-batch stages:* motion-correction, shading-correction, preprocessing, and trace extraction can be performed on batches of frames. These operations scale with batch size and do not require global knowledge of the full video, making them potentially suitable for streaming or real-time use.
  - *Less-than batch stages:* Preprocessing generates summary images (e.g., max, mean, std projections) in both spatial and temporal domains. Its cost depends on the number and size of summary windows used (S, L), not the total number of frames.
  - *Blocking stages:* evalution, segmentation and mask computation require access to the entire dataset of the preceding stages. Spike detection must also be performed on the full extracted traces to ensure temporal accuracy. These stages inherently block real-time execution due to their dependency on future or global data. Evaluation however is a nice-to-have feature, but is not necessary for spike detection.
  - *Front-loaded stages:* Weight initialization and file reading occur once at startup. While initialization has a fixed cost, file reading scales with video duration and can become a bottleneck for longer recordings.
- **Implications for real-time operation:**
  - Motion-correction, shading correction, and trace extraction are promising targets for real-time adaptation due to their frame-wise and non-blocking behavior.
  - Segmentation, mask computation, and spike detection remain inherently offline, but could potentially be scheduled periodically in near-online setups if the summary image design supports it.

4.1.4 Motion-Correction Scaling Comparison

While the theoretical analysis in Table 2.2 suggests that FIOLA's motion-correction, based on Fourier-domain alignment, should outperform VOLTAGE's patchwise ZNCC approach, practical profiling revealed otherwise. When excluding initialization steps, both pipelines spent a comparable amount of time on the core motion-correction task. This unexpected parity prompted further investigation into why VOLTAGE, despite its higher theoretical complexity, demonstrated similar empirical performance. The next section explores this discrepancy in detail by examining hardware-level scaling and implementation factors.

VOLTAGE's ZNCC-based GPU motion-correction performs reasonably on high-end GPUs but degrades significantly on lower-end hardware. A comparison run on Rui Silva's PC showed a nearly twofold slowdown versus CUBE. Additionally, VOLTAGE's CPU fall-back (used when GPU support is unavailable) was an order of magnitude slower, taking 108.75 seconds for a single 15,000-frame video. In contrast, FIOLA's Fourier-based correction retained near-identical performance across systems, highlighting its portability and more efficient GPU utilization.

To isolate the impact of GPU architecture on motion-correction performance, both pipelines were benchmarked on two hardware setups: the CUBE server (Tesla V100) and a standalone Lab PC (RTX 3080). Table 4.4 summarizes the relevant hardware specifications for motion-correction profiling.

Table 4.4: Hardware configurations for motion-correction benchmarking

Component	**CUBE Server (Tesla V100)**	**Standalone PC (RTX 3080)**
CPU	AMD EPYC 7551 32-Core @ 2.56 GHz	AMD Ryzen 7 5800X 8-Core @ 3.8 GHz
GPU	NVIDIA Tesla V100 32 GB (Volta)	NVIDIA RTX 3080 12 GB (Ampere)
CUDA Cores	5,120	8,704
Tensor Cores	640	272
RAM	4×Kingston KF3200C16D4/32GX 32 GB 2400 MHz DDR4	32 GB DDR4 (single DIMM)
Disk	Dell Express Flash NVMe P4600 3.2 TB	Samsung 970 EVO 1 TB NVMe SSD
OS	Ubuntu 20.04.6 LTS (Focal Fossa)	Windows 10 (WSL2)

The raw imaging frames were then up-sampled by factors of 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, and 2.4, and the motion-correction runtime was recorded for each pipeline on both GPUs. Figure 4.3 shows these timings.

At modest scales (1.0-2.0×), both GPUs yield comparable performance for each pipeline. Beyond 2.0×, however, the RTX 3080's VOLTAGE motion-correction runtime increases sharply (110 s at 2.4×vs. 38 s on the Tesla V100), while FIOLA's runtimes remain more

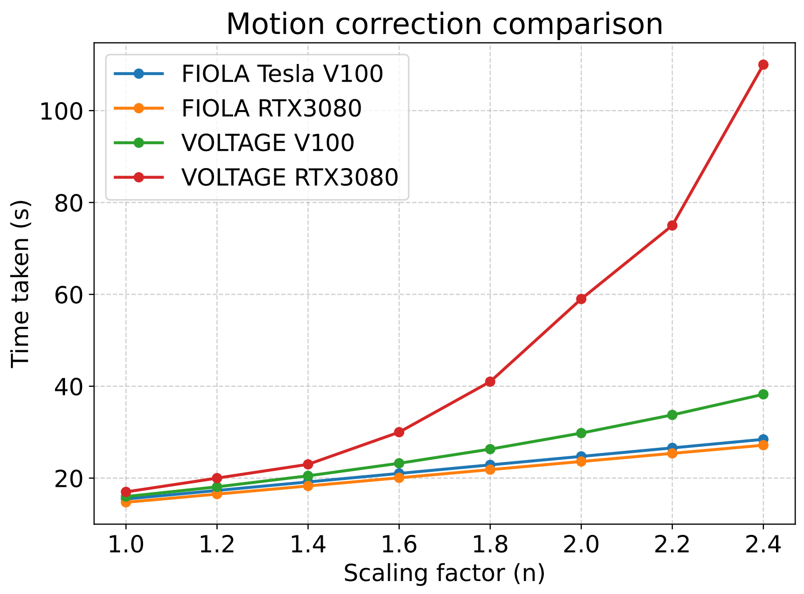


Figure 4.3: Motion-correction runtime vs. up-sampling factor for FIOLA and VOLTAGE on the Tesla V100 and RTX 3080.

consistent and even slightly faster on the Lab PC, demonstrating FIOLA's greater resilience to GPU architectural differences. This behavior reflects the underlying hardware characteristics: for small up-sampling factors, the RTX 3080's higher FP32 throughput (29.8 TFLOPS vs. 15.7 TFLOPS on the V100), higher boost clocks, and larger CUDA core count let it excel when the working set fits in on-chip caches. As frame sizes grow, however, the workload becomes memory-bound and cache-limited. Here, the V100's HBM2 memory (900 GB/s) and larger on-chip caches sustain higher effective bandwidth than the 3080's GDDR6X (760 GB/s) and smaller caches. Additionally, NVIDIA's cuFFT library leverages the V100's Tensor-core-accelerated complex-math kernels more aggressively for large transforms. Together, these factors allow the V100 to maintain flat scaling on large inputs while the 3080's performance degrades. Further benchmarking in Appendix A confirms these observations, as the Lab PC outperforms on smaller computational loads but lags on larger ones.

4.1.5 Overall Pipeline Runtime Comparison

Although VOLTAGE completes the full pipeline about 44 % faster than FIOLA (82.62 s vs. 131.11 s), the lion's share of this difference is due to FIOLA's substantially longer initialization (39.01 s versus 9.59 s). After initialization the actual motion-correction workloads are nearly identical (15.46 s for FIOLA vs. 15.98 s for VOLTAGE on motion-correction), demonstrating that the core computational effort is comparable for the spatial resolution of 116 x 496, but signifanctly diverges at higher spatial resolutions.

Table 4.5: Comparable runtimes for FIOLA and VOLTAGE on the Cube server

Pipeline Stage	VOLTAGE (s)	FIOLA (s)
File read	6.56	6.85
Motion-correction initialization	9.59	18.01
Motion-correction work	15.98	15.46
Segmentation & trace extraction	15.98	30.78
Spike detection work	0.32	0.98
File write	8.71	8.62
Total end-to-end runtime	82.62	110.09

4.1.6 Spike Detection-Accuracy Comparison

While runtime performance is a critical factor in designing real-time imaging pipelines, it is not the sole concern. The primary objective of these systems is to enable accurate and interpretable spike detection under realistic, often noisy conditions. A pipeline that is computationally efficient but fails to reliably detect spikes, especially at low amplitudes, ultimately undermines its scientific utility. Accordingly, the next section shifts focus from speed to signal quality, comparing the spike detection accuracy of FIOLA and VOLTAGE across a range of signal amplitudes.

To evaluate spike-detection performance across pipelines, FIOLA and VOLTAGE were compared using the benchmark dataset from Platisa et al.[21]. Both pipelines were run on identical motion-corrected videos, and F1 scores were computed across a range of spike amplitudes.

Figure 4.4 shows that FIOLA consistently outperforms VOLTAGE, especially at low spike amplitudes. While both pipelines converge toward high F1 scores at larger amplitudes, FIOLA maintains a measurable advantage in the low signal-to-noise regime, making it more suitable for experiments requiring high temporal precision or weak signal detection.

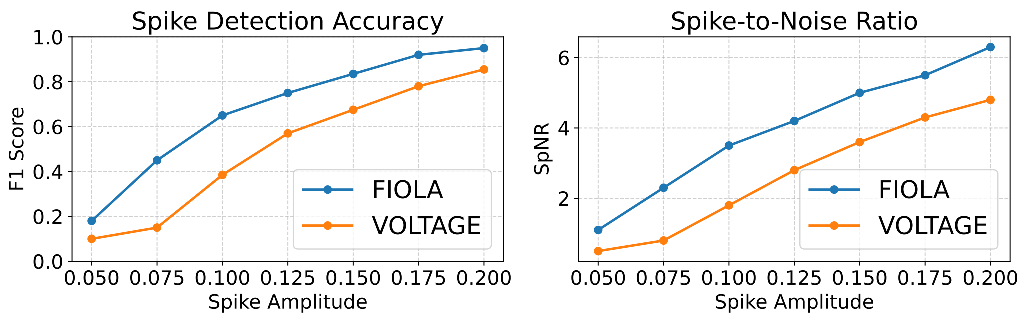


Figure 4.4: F1 scores across spike amplitudes for FIOLA and VOLTAGE. FIOLA exhibits superior sensitivity at low amplitudes and maintains comparable accuracy at higher levels.

4.2 Hybrid Pipeline Design

The preceding analysis demonstrates that while FIOLA and VOLTAGE each excel in specific areas, whether in runtime efficiency, segmentation quality, or spike detection accuracy, neither offers a complete solution that meets all the demands of real-time, scalable neuroscience experiments. Performance alone cannot dictate design choices; accuracy, modularity, and compatibility with online workflows are equally critical. These insights directly motivate the design of a hybrid pipeline that strategically combines the strengths of both systems.

The goal of this design is twofold: first, to build an offline pipeline that operates on a single input file containing raw frames, optimized for accuracy and throughput during initialization and preprocessing; and second, to generate all necessary assets, such as segmentation masks and motion-correction templates, for a streamlined online pipeline. The online pipeline is designed to prioritize low-latency inference and spatial precision, enabling real-time operation in feedback-driven or closed-loop experimental settings.

The next two sections describe these two pipeline variants in detail:

- **Offline pipeline:** Focused on high-throughput batch processing and setup of necessary assets for online use.
- **Online pipeline:** Designed for real-time operation with minimal per-frame latency, aiming at 500 - 1000 FPS at a spatial resolution of 300 $\times$300.

Following the offline and online design descriptions, the remainder of this chapter introduces enhancements aimed at improving overall pipeline performance. These are organized around two key objectives:

- **Accuracy:** Improvements such as vignette filtering and wavelet-based trace denoising increase spike detection reliability, especially under noisy conditions. (Section 4.2.4
- **Latency and throughput:** Profiling-driven optimizations, including memory-efficient FFTs, kernel fusion, and TensorFlow graph compilation, reduce critical-path runtime during online execution. (Section 4.2.5

Together, these contributions form a hybrid architecture that balances accuracy, speed, and real-time feasibility in a cohesive, modular design.

4.2.1 Motivation and Design Rationale

Constructing a hybrid voltage imaging pipeline requires balancing performance, flexibility, and real-time feasibility across all processing stages. While both FIOLA and VOLTAGE provide high-performing components, neither alone satisfies the full set of demands required for scalable, online-compatible analysis. FIOLA offers robust motion-correction and reliable spike detection under tight latency constraints, but its segmentation strategy is limited in accuracy and generalizability. VOLTAGE excels in segmentation and modularity but lacks online-optimized spike inference and requires more homogeneous GPU configurations for consistent motion-correction performance.

The hybrid pipeline is designed to integrate the most effective components from each system, chosen according to five core design principles:

- **Hardware-agnostic motion-correction:** FIOLA's GPU-efficient, frequency-domain motion-correction exhibits predictable performance across GPU types and resolutions (Section 4.1.4). In contrast, VOLTAGE's patchwise ZNCC approach performs well but scales less reliably with resolution and GPU memory. FIOLA's method thus forms the foundation of the hybrid pipeline's spatial alignment module.
- **Segmentation accuracy and modularity:** VOLTAGE's U-Net segmentation model, trained on curated data and using summary projections, outperforms FIOLA's low-rank factorization approach in complex or overlapping ROIs. It is also implemented modularly, allowing its output to be reused in downstream stages without pipeline-wide coupling. This makes it ideal for integration into a hybrid framework.
- **Streamlined trace extraction:** VOLTAGE's post-segmentation trace extraction operates independently of upstream assumptions, using binary or soft ROI masks to compute fluorescence signals. This modularity ensures compatibility with the FIOLA-derived motion-corrected video, without the need for re-implementation or format translation.
- **High-fidelity spike detection:** FIOLA's ridge-regression spike inference, augmented with adaptive thresholds, significantly outperforms VOLTAGE's z-score baseline, particularly in low-SNR conditions. Retaining this component provides the hybrid pipeline with improved spike sensitivity and timing accuracy in real-time settings.
- **Latency-aware integration:** The hybrid pipeline follows FIOLA's principle of offloading computation to an offline initialization phase. By front-loading steps such as motion template generation and segmentation, the runtime path avoids per-frame bottlenecks and maintains compatibility with online or feedback-driven experiments.

Together, these decisions produce a hybrid architecture that preserves the streaming efficiency and spike inference quality of FIOLA, while gaining the segmentation accuracy and extensibility of VOLTAGE. The result is a pipeline suitable for real-time, high-throughput experiments under realistic hardware constraints.

4.2.2 Offline Pipeline Design

Figure 4.5 illustrates how the hybrid pipeline assembles these components: FIOLA's motion-correction and spike-detection replace VOLTAGE's equivalent stages, while maintaining VOLTAGE's preprocessing, segmentation, and modular trace extraction.

While Figure 4.5 outlines the modular composition and computational complexity of the hybrid pipeline, Figures 4.6 and 4.7 demonstrate how this architecture operates in practice.

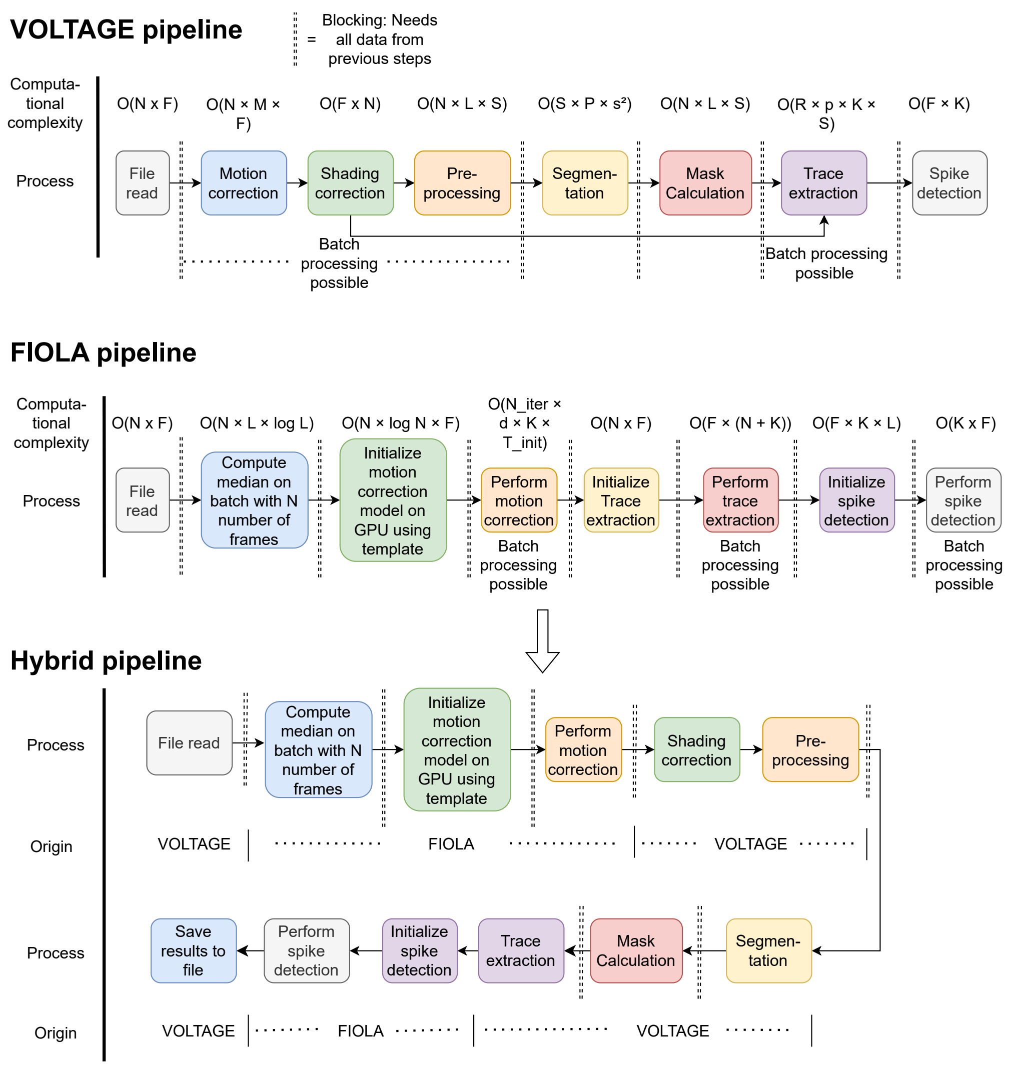


Figure 4.5: Architecture of the offline hybrid pipeline integrating FIOLA and VOLTAGE components with computational complexity, batch processing-compatibility and blocking behavior shown for each component.

It traces the per-frame dataflow during online execution, showing how individual frames are processed step-by-step through motion-correction, ROI-based trace extraction, and spike inference, using precomputed artifacts from the initialization phase. This view clarifies the runtime behavior and modular interactions across the online stages.

The following variable definitions describe the variables used for the computational complexity, as described in Section 2.7, in Figure 4.5:

$$
\begin{aligned}
F &= \text{Total number of frames in the video} \\
N &= \text{Number of pixels per frame (image resolution)} \\
K &= \text{Number of neurons (ROIs)} \\
r &= \text{Average ROI size in pixels} \\
L &= \text{Temporal window length (in frames)} \\
T_{\text{init}} &= \text{Number of initialization frames} \\
N_{\text{iter}} &= \text{Number of optimization iterations} \\
P &= \text{Number of patches per frame (VOLTAGE)} \\
s &= \text{Patch size (pixels)} \\
M &= \text{Number of shift candidates per patch} \\
S &= \text{Number of temporal segments (VOLTAGE)} \\
p &= \text{Number of pixels in a spatial region (for NMF)} \\
R &= \text{Number of overlapping regions demixed via NMF}
\end{aligned}
$$

Integrating across two distinct codebases required data-flow and format standardization:

- Split VOLTAGE's original fused CUDA kernel into shading and preprocessing, so pre-aligned frames from FIOLA can be ingested directly, for the aim of keeping modularity.
- Unified tensor shapes and data types between FIOLA and VOLTAGE modules to allow seamless swapping.
- Adapted FIOLA's spike-detector to accept VOLTAGE-generated traces, preserving ROI indexing without altering core inference logic.

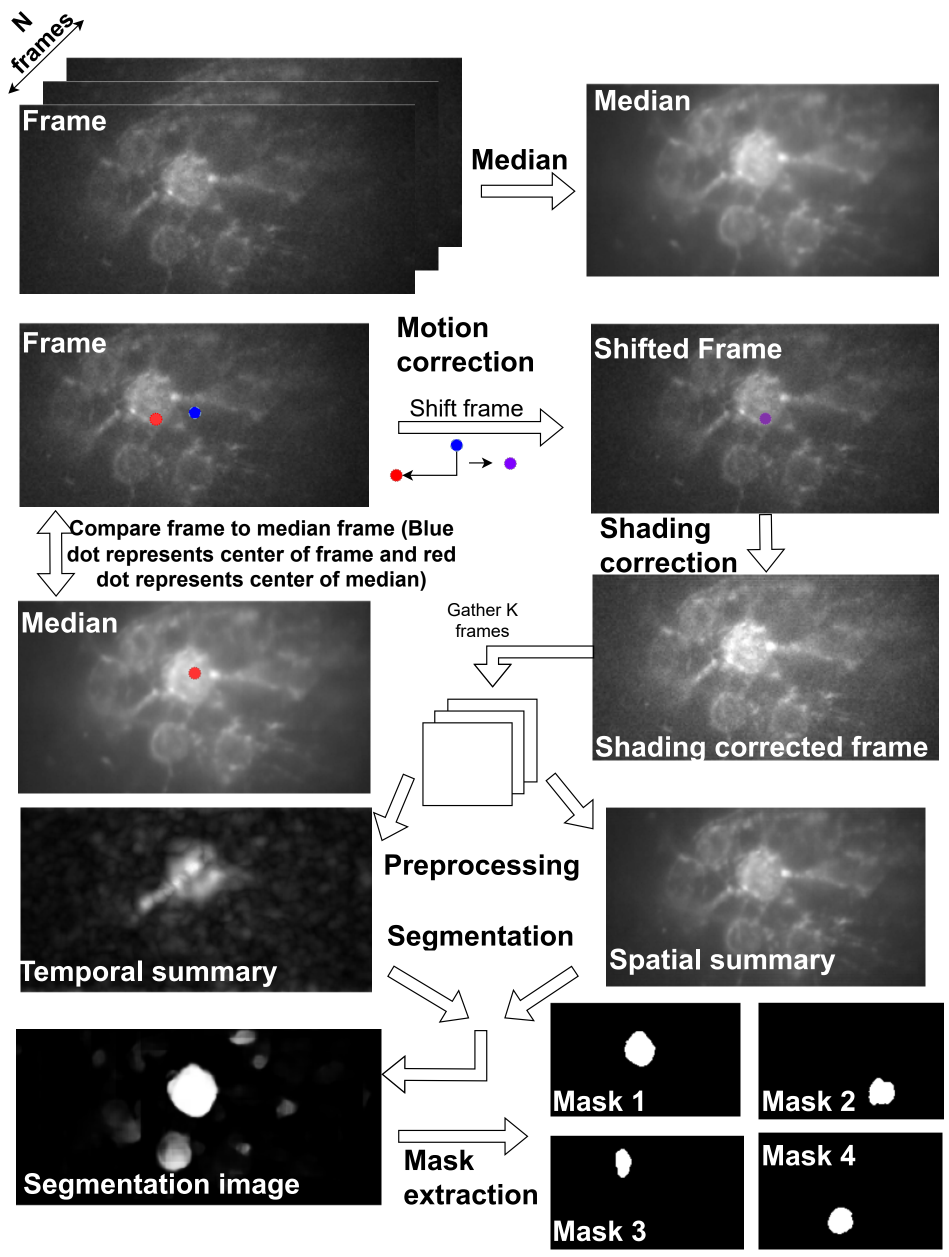


Figure 4.6: Step-by-step illustration of the hybrid pipeline's initial stages. A median image is computed across the first L frames and used as a template for motion-correction. Each frame is aligned via motion-correction, shading-corrected, and grouped into batches for pre-processing. Temporal and spatial summaries are generated and passed into a segmentation model, which produces ROI masks.

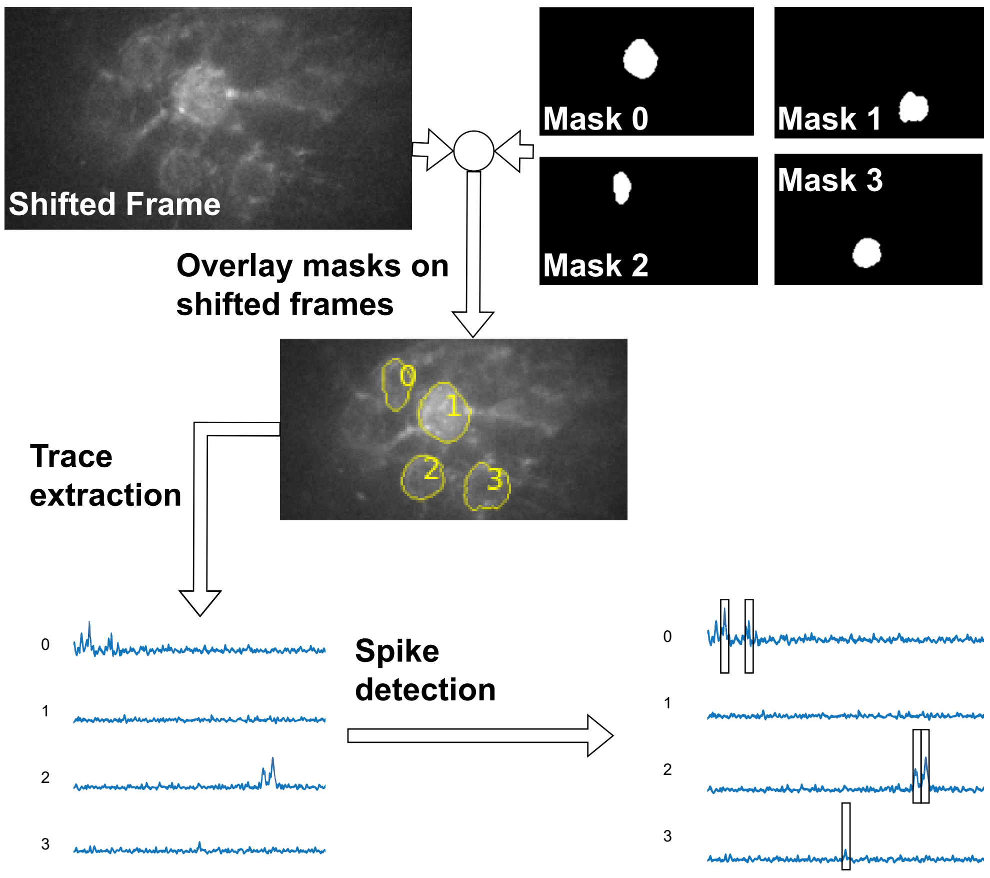


Figure 4.7: Continuation of the hybrid pipeline. Generated ROI masks are overlaid on motion-corrected frames to extract fluorescence traces over time. These traces are passed to a spike detection module, which identifies significant neural events using ridge-regression filters.

4.2.3 Online Pipeline Design

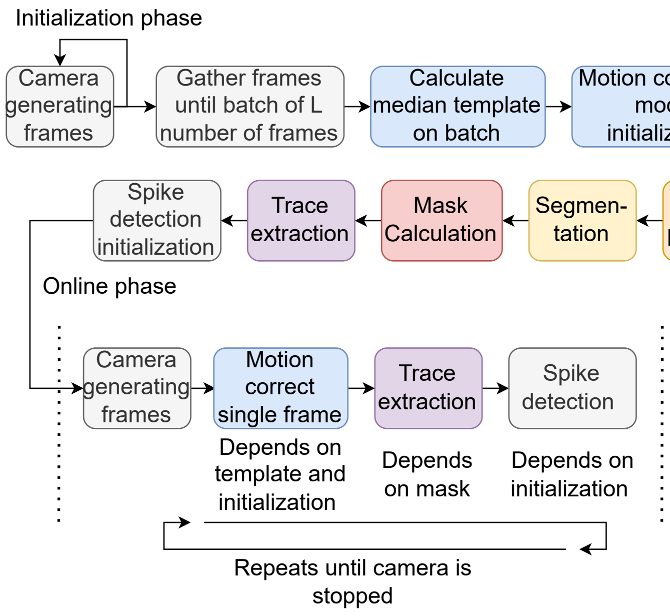

Initialization phase
Camera generating frames
Gather frames until batch of L number of frames
Calculate median template on batch
Motion correction model initialization
Motion correct frame batch
Shading correction
Pre-processing
Segmen-tation
Mask Calculation
Trace extraction
Spike detection initialization
Online phase
Camera generating frames
Motion correct single frame
Trace extraction
Spike detection
Depends on template and initialization
Depends on mask
Depends on initialization
Repeats until camera is stopped

Figure 4.8: Architecture of the online hybrid pipeline integrating FIOLA and VOLTAGE components.

To enable low-latency feedback, the hybrid pipeline runs in two phases: initialization and online. Heavy workloads (template computation, motion-correction, segmentation, ROI generation, and spike-detector initialization) occur in the initialization phase, while the online phase executes only light, per-frame operations (motion-correction, trace extraction, spike inference). This separation ensures minimum latency after startup and preserves all acquired data in buffer.

Table 4.6: Dependency overview of the hybrid pipeline

Stage	**Input**	**Output**	**Dependencies**
Median template	Batch of early frames	Single template frame	, -
Motion-correction	Frame or batch of frames	Corrected frames	Template frame
Shading & preprocessing	Batch of aligned frames	Temporal and spatial summary images	-
Segmentation & ROI mask generation	Temporal and spatial summary images	ROI masks	U-Net weights
Trace extraction	Corrected frames & ROI masks	Traces	ROI masks
Spike detection	Traces	Spike timings	Warm-up traces

Stage dependencies, input and output are shown in Table 4.6. It illustrates why precomputing templates and masks is necessary before online inference. The runtime flow for the online pipeline is shown in Figure 4.8. By offloading heavy computations to the initialization stage, the online stage achieves minimal per-frame latency using FIOLA's motion-correction inference, mask-based trace extraction, and ridge-regression spike inference. The design is efficient, modular, and readily extensible to multi-GPU or distributed environments.

4.2.4 Accuracy

To further build on the efficiency of the online pipeline, attention was next directed toward enhancing its analytical accuracy. While the streamlined runtime design ensures low-latency execution, it is equally important to maximize the fidelity of neural signal extraction. Therefore, subsequent efforts focused on algorithmic refinements that improve spike detection precision and signal quality, particularly in the face of motion artifacts and imaging noise. These enhancements complement the pipelines modular architecture and maintain compatibility with its real-time constraints.

Pre-Motion Correction: Gaussian Vignette Filtering

Uneven illumination and peripheral noise are known to degrade motion-correction accuracy, especially for frequency-domain alignment methods. To counteract these artifacts, a radial Gaussian vignette mask is generated and applied to each frame, attenuating pixels near the edges and emphasizing the more reliable central regions.

Unlike traditional Gaussian filtering, which involves local convolution and averages over neighboring pixels, vignette filtering applies a per-pixel multiplicative weight based solely on the pixel's radial distance from the image center. This preserves local spatial structure while down-weighting peripheral regions.

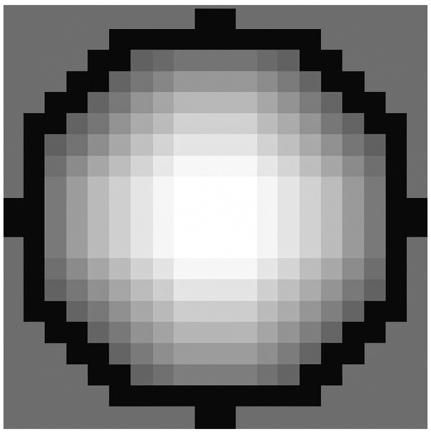
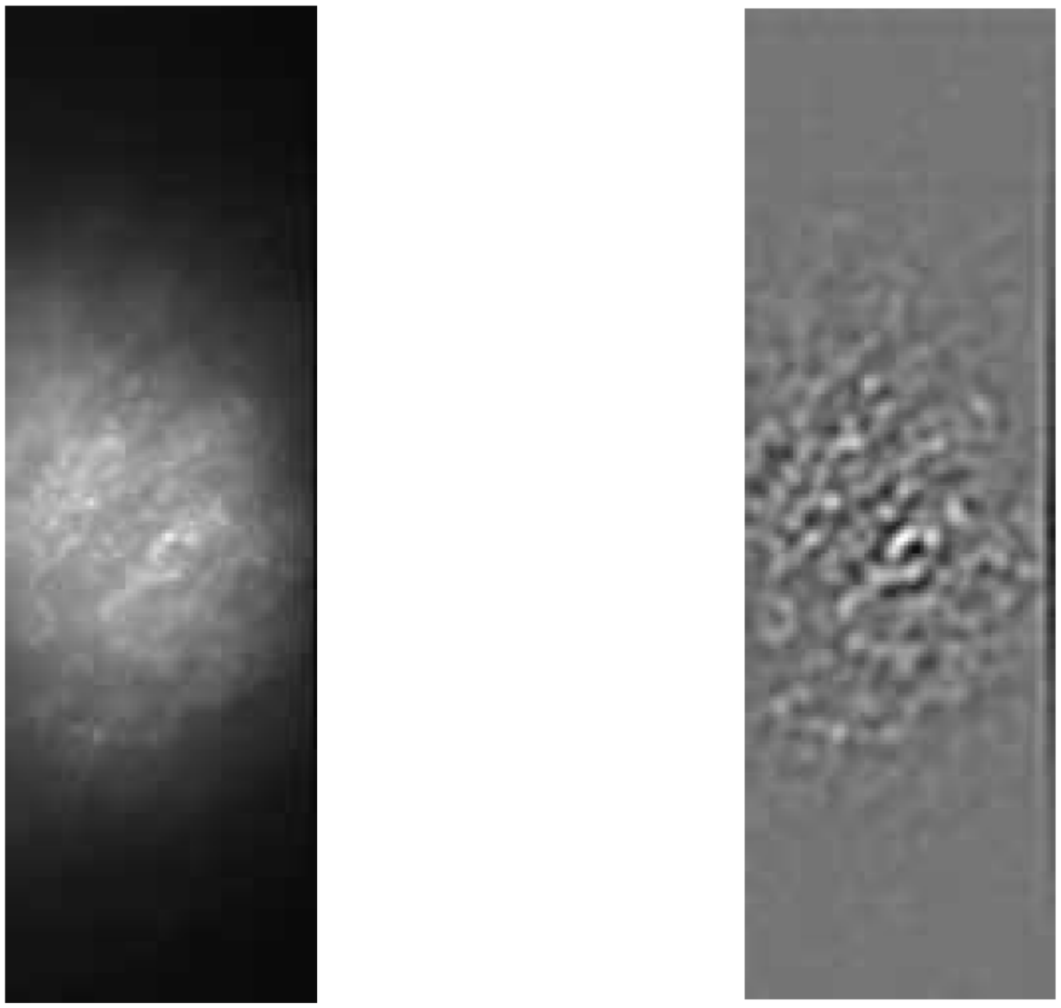


Figure 4.9: Example of a radial Gaussian vignette mask.

Let the image frame have dimensions $H \times W$, and define the center coordinates (x_0, y_0). The vignette weight at pixel (x, y) is given by the normalized Gaussian:

$$W(x,y) = \exp\left(-\frac{(x-x_0)^2 + (y-y_0)^2}{2\sigma^2}\right), \tag{4.1}$$

where σ controls the falloff rate. The weights are then normalized to the range $[0,1]$.



(a) Original motion-correction template

(b) Vignette-filtered motion-correction template

Figure 4.10: Comparison of motion-correction templates. The vignette-filtered template emphasizes central features while attenuating noisy peripheries (see also Figure 4.9). Image adapted from [16].

Each frame $F(x,y)$ is preprocessed via element-wise multiplication:

$$F_{\text{filtered}}(x,y) = F(x,y) \times W(x,y), \tag{4.2}$$

thereby suppressing peripheral noise and edge artifacts without blurring or distorting image content.

Computational Complexity. Computing the radial weights for $d = H \times W$ pixels requires $O(d)$ operations, and the element-wise multiplication also costs $O(d)$. Thus, the overall complexity of the vignette filtering stage is scaling linearly with the number of pixels.

$$O(d) + O(d) = O(d), \tag{4.3}$$

Both the motion-correction template frame and each incoming frame that must be aligned are filtered using this vignette function. As shown in Figure 4.10, the effect is to emphasize stable, central regions while suppressing edge artifacts. This preprocessing step improves the robustness and stability of subsequent motion-correction stages by ensuring that alignment computations are biased toward the most reliable image areas.

Pre-Spike Detection: Wavelet-Based Trace Filtering

To further improve the signal-to-noise ratio (SNR) of extracted fluorescence traces, a discrete wavelet transform (DWT) filter was initially implemented prior to spike detection. Wavelet-based denoising techniques have been shown to preserve the temporal structure of neural signals while effectively suppressing high-frequency noise [39, 40].

In this implementation, each trace was decomposed using a DWT based on a Daubechies wavelet, which acts as the mother wavelet, a reference waveform that defines the shape and mathematical properties of all scaled and shifted wavelet functions used in the transform. The DWT applies a multiresolution analysis by convolving the signal with filter banks derived from this mother wavelet, producing coefficients that capture both coarse and fine temporal features of the trace. Noise was attenuated by applying soft thresholding to coefficients at the finer scales, and the signal was reconstructed from the remaining components, preserving key low-frequency structures likely to reflect neural activity.

This denoising step improved spike-detection performance in early evaluations, particularly under low SNR conditions. However, subsequent modifications to FIOLA's spike detection, specifically adjustments to the adaptive regression parameters and decision thresholding, substantially improved its baseline sensitivity and robustness. As a result, the wavelet-based preprocessing no longer provided a significant benefit and was ultimately removed from the final hybrid pipeline design.

4.2.5 Latency and throughput

As discussed in Section 4.2, the online pipeline is aiming to operate at real-time frame rates, 500 to 1000 FPS, with 50 ms latency at most. However, empirical performance measurements revealed that this constraint is not fully satisfied across all conditions. As will be shown in Section 6.5.1, motion-correction emerges as the dominant source of latency during online execution.

To better understand and address this limitation, extensive profiling was conducted after the introduction of the vignette filtering stage (Section 4.2.4).

Profiling Results: Identifying the Bottleneck

Profiling was performed using TensorFlow's built-in performance tracing tools. Figure 4.11 illustrates the timeline of operations during online motion-correction, captured at a representative resolution and batch size. The timeline reveals clear staging points for FFT computation, image normalization, and inverse FFT, interleaved with memory transfers and small kernel invocations.

Because of how TensorFlow's tracing infrastructure operates, profiling must be conducted in eager (non-compiled, but optimized through Tensorflow's graphing mechanism) mode to obtain meaningful attribution of function calls. As a result, some execution paths differ slightly from the actual compiled pipeline. However, this mode is sufficient for identifying the major bottlenecks: functions that dominate the timeline in eager mode remain the primary sources of latency in compiled execution as well. Therefore, while absolute timings may shift, the profiling trace reliably reveals the operations contributing the most to overall runtime.

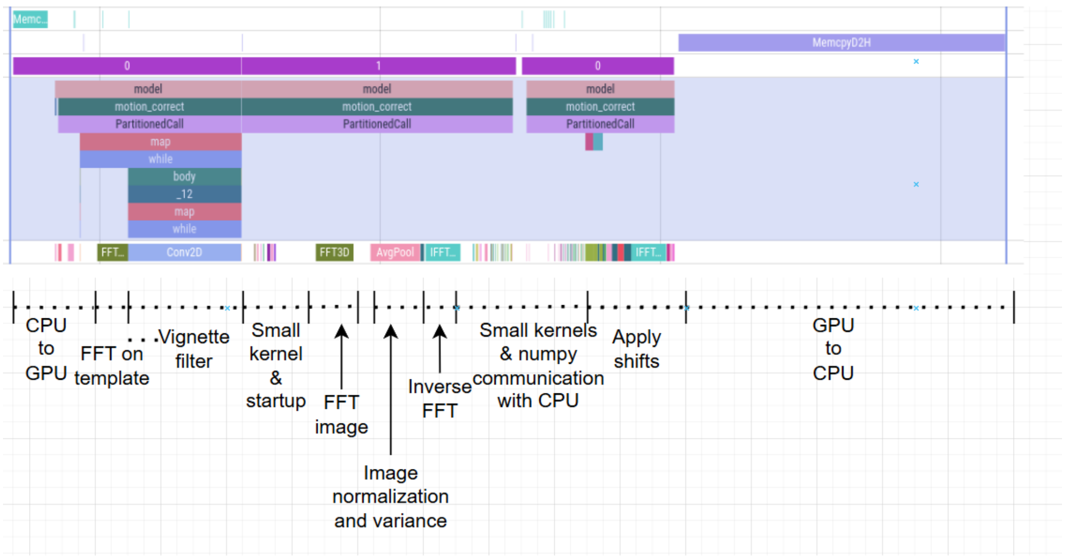


Figure 4.11: GPU profiling timeline of the motion-correction stage in online mode. FFT operations, memory transfers, and interleaved CPU/GPU communication dominate latency.

As seen in the breakdown, motion-correction alone accounts for over 99% of total online latency, with GPU-to-CPU synchronization, FFT/IFFT, and small-kernel fragmentation as the key bottlenecks. These observations informed several proposed optimizations as shown in 5.3.

Roofline Analysis

To further understand the performance characteristics and identify whether the application is memory-bound or compute-bound, a roofline model analysis was performed. Figure 4.12 presents the roofline plot for the key computational kernel.

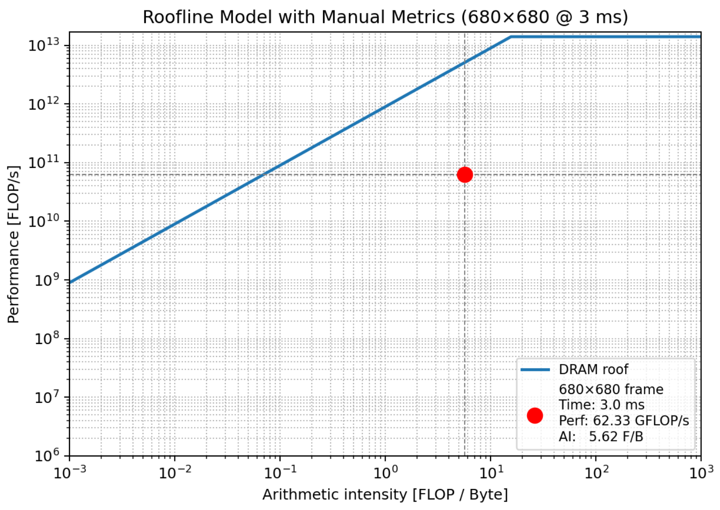


Figure 4.12: Roofline Model with Manual Metrics for the 680x680 frame, showing the achieved performance relative to theoretical maximums.

The roofline plot illustrates the upper bounds of performance (the "roof") based on the system's memory bandwidth (DRAM roof) and peak floating-point performance. The red marker on the graph represents the achieved performance of the 680x680 frame processing.

From the plot, the following metrics are observed:

- Time: 3.0 ms
- Performance (Perf): 62.33 GFLOP/s
- Arithmetic Intensity (AI): 5.62 F/B (FLOP/Byte)

The red dot, representing our application's performance point, falls below the "DRAM roof" line and is located on the rising portion of the roofline. This indicates that the current operation is memory-bound. The arithmetic intensity of 5.62 FLOP/Byte suggests that for every byte transferred from memory, 5.62 floating-point operations are performed. Despite this, the performance is limited by the rate at which data can be fetched from DRAM, rather than the raw computational power of the processor. This finding aligns with the previous profiling results that highlighted memory transfers as a significant contributor to latency. Further optimizations should therefore focus on reducing data movement, improving data locality, or increasing the arithmetic intensity of the kernel to move the performance point closer to the computational peak.

Chapter 5

Implementation

While the hybrid architecture defines the logical flow and component selection of the pipeline, its practical utility depends on efficient and portable implementation. This chapter details how the pipeline was implemented in practice, highlighting the major architectural choices, development patterns, and performance engineering efforts that make it robust, modular, and scalable. The following sections cover:

- **General pipeline implementation:** Each processing stage- motion-correction, segmentation, trace extraction, and spike detection, is implemented as a modular, GPU-accelerated component using Python and TensorFlow. The design emphasizes clean stage boundaries and interoperability, enabling both offline and online reuse. (Section 5.1
- **Setup latency:** Initialization-time optimizations, such as GPU-accelerated median filtering and multithreaded pipeline startup, minimize delays during experiment setup and accelerate transition to online operation. (Section 5.2)
- **Profiling-driven optimization:** Key performance bottlenecks identified in Section 5.3 were targeted through algorithmic refinements and efficient use of TensorFlow primitives.
- **Usability and generalizability:** Modular stage control, containerized deployment via Docker, and automated calibration tools improve reproducibility and adaptability across hardware configurations and lab setups. (Section 5.4)

5.1 General Pipeline Implementation

The hybrid pipeline is implemented in Python, with core components written using TensorFlow, NumPy, Cython, Cuda and C++ for performance-critical sections. Both the offline and online variants share a modular architecture centered around a unified configuration system and reusable processing stages. This modularity enables reproducibility, rapid prototyping, and smooth integration of GPU acceleration.

5.1.1 Offline Pipeline

The offline pipeline is launched from a Python entrypoint that loads configuration files, initializes models, and executes processing stages sequentially. Each stage is conditionally enabled via a configuration dictionary and supports output caching. Tagged Image File Format (TIFF) video files, used for high bit depth lossless compression, are loaded using `tifffile`, stored as NumPy arrays and further processed downstream.

Each stage is defined in a dictionary with an `enabled` flag and an associated cache path:

```
stages = {
  "motion_correction": {
    "enabled": True,
    "save_path": "intermediates/registered.npy"
  },
  ...
}
```

Each stage checks its `enabled` flag before running. If disabled, the corresponding output is loaded from disk using `np.load()`, allowing downstream stages to execute normally. For example, if segmentation is disabled but a cached ROI mask is available, the pipeline loads `intermediates/segmentation_mask.npy` and continues with trace extraction. This strategy minimizes recomputation during development or parameter tuning.

Algorithm 1: Offline pipeline stage execution logic

```
1 foreach stage in pipeline do
2     if stage.enabled then
3         run(stage.function)
4     else if stage.cache_path exists then
5         load(stage.output ← np.load(stage.cache_path))
6     else
7         raise Error("Missing cached output and stage disabled")
```

To further reduce preprocessing latency and improve data throughput within TensorFlow stages (e.g., segmentation), the offline pipeline uses a highly optimized input pipeline built with TensorFlows `tf.data` API. Batch generators are constructed using `tf.data.Dataset.from_generator`, combined with prefetching and parallel mapping to overlap data preparation with GPU execution. This ensures the GPU remains consistently fed with ready-to-process batches, minimizing idle time due to I/O stalls or preprocessing delays. These design choices follow TensorFlow performance guidelines [41] and contribute significantly to offline execution efficiency.

Key processing steps:

- **Motion-Correction:** TensorFlow-based layer using batched FFTs, XLA compilation, and Cython-managed memory for speed.
- **Shading & Preprocessing:** NumPy and Cython kernels applied as batch functions, including background subtraction and bandpass filtering.
- **Segmentation:** U-Net model from VOLTAGE, fed with summary projections and postprocessed with SciPy and connected-component labeling.

- **Trace Extraction:** Implemented in multithreaded Cython with OpenMP, accessing CuPy arrays via memory views.
- **Spike Detection:** Batch evaluation using FIOLA's regression-based spike inference.

5.1.2 Online Pipeline

The online pipeline uses the same configuration and model-loading routines but operates on a online loop. Memory-mapped files and ZMQ simulate camera input during development.

Concurrent threads manage the pipeline as described in 5.2.2:

- **Motion-Correction:** TensorFlow layer processes frames and outputs aligned images.
- **Trace Extraction:** Applies masks to incoming frames using multithreaded python.
- **Spike Detection:** Real-time regression inference using a stateful `fit_next()` method.

Data is passed via thread-safe queues to maintain frame order and avoid latency spikes. Online execution is designed to maintain $>$500 FPS on standard GPUs with minimal overhead.

5.1.3 Language and Framework Overview

Table 5.1: Languages and frameworks used for major pipeline components.

Component	**Implementation**
Pipeline orchestration	Python
Configuration and I/O	Python (YAML, NumPy, TIFF)
Motion-correction	Python, TensorFlow (XLA)
Shading & preprocessing	Cython, C++
Segmentation	Python, TensorFlow (U-Net), SciPy
Trace extraction	Python
Spike detection	Python (NumPy, FIOLA)
Streaming infrastructure	Python (ZMQ, `memmap`, `threading`)
Profiling and logging	Python,`psutil`, `tracemalloc`

5.2 Setup latency optimization

For real-time pipelines, minimizing setup latency is critical, not only to reduce downtime between experiments but also to enable rapid prototyping and deployment in fast-paced experimental workflows. This section focuses on optimizations that reduce initialization time, particularly those affecting the offline preprocessing stages that must complete before online execution can begin. These include GPU-accelerated median filtering for motion-correction template generation and parallelized startup procedures that improve resource utilization and responsiveness. Together, these strategies help transition the pipeline from raw input to real-time readiness with minimal delay.

5.2.1 GPU-Accelerated Median Filtering

To generate the motion-correction template, the pipeline computes the per-pixel temporal median across a stack of L frames. While the median provides superior robustness compared to the mean, especially for motion alignment, its computation is significantly more expensive. A naive CPU implementation computes the median at each pixel by sorting L values, resulting in a complexity of $O(d \cdot L \log L)$, where $d = H \times W$ is the number of pixels per frame.

To eliminate this bottleneck, a GPU-accelerated approximation was implemented using TensorFlow. Although TensorFlow lacks a native GPU median operator, the median was computed as the $(\lfloor L/2 \rfloor + 1)$-th largest value using `tf.nn.top_k`, applied after transposing the input tensor from $[L,H,W]$ to $[H,W,L]$. This avoids full sorting and leverages partial sort acceleration on GPU.

The core median function, `fast_median()`, is compiled with XLA using `@tf.function(experimental_compile=True)`, enabling kernel fusion and reducing launch overhead. For high-resolution inputs (e.g. $[15{,}000, 464, 498]$), this approach alone can exceed 32 GB of available device memory . To manage this, a chunked strategy was introduced: the volume is divided along the height dimension into smaller strips (e.g. 50 rows at a time), each processed independently using `fast_median()`. The results are stitched together to form the full template. This method maintains GPU residency for all computation while bounding memory usage.

The final implementation includes two key components:

- `fast_median(frames)`: Transposes and computes the approximate pixelwise median.
- `median_chunked(frames, chunk_h)`: Splits input along height, applies `fast_median()` to each chunk, and concatenates outputs.

Alternative methods, such as using the mean or a rolling average, were tested but yielded worse F1 scores in segmentation and spike detection, and were therefore not adopted. The final design achieves a more scalable

$$O(d \cdot L) \tag{5.1}$$

complexity while enabling template computation at full resolution without out-of-memory failures.

5.2.2 Multithreaded Model Loading and Pipeline Parallelism

One major optimization involves launching key tasks concurrently during pipeline startup and execution. As shown in Figure 5.1, a multithreaded design was implemented to load models, buffer frame batches, and initiate downstream modules in parallel rather than sequentially. Threading was deliberately chosen over multiprocessing, as the pipeline is predominantly I/O-bound rather than compute-bound. Initial experiments with multiprocessing, particularly in the spike detection setup, introduced unnecessary overhead due to interprocess communication and memory duplication, resulting in slower startup times. Using threads avoids these issues and enables faster, more efficient initialization.

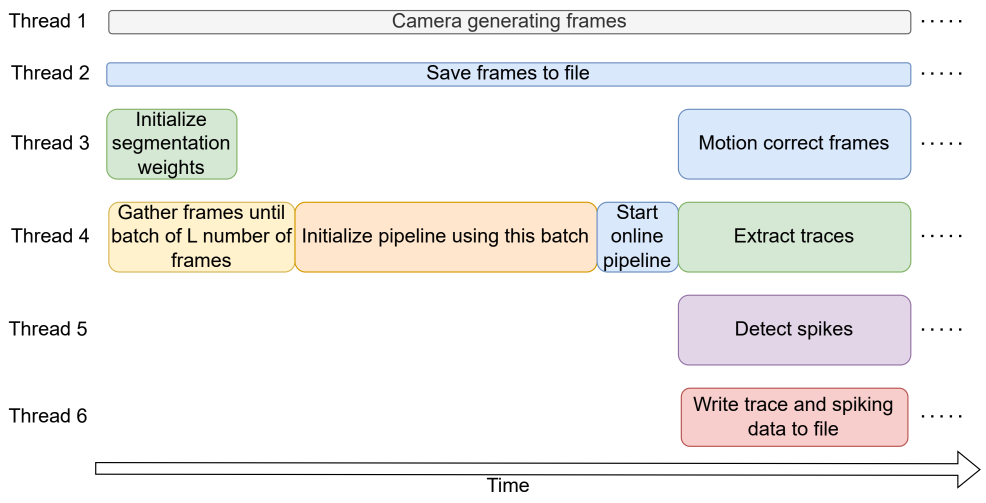


Figure 5.1: Timeline of the hybrid pipeline's startup and online execution using multiple threads. Early stages, including data buffering and model loading, are run concurrently. Downstream components operate in parallel, synchronized through shared memory buffers.

Specifically:

- **Thread 2** continuously receives frames from the camera, saving them to disk and populating an in-memory buffer until a minimum batch size L is reached.
- **Thread 3** initializes the segmentation model weights.
- **Thread 4** performs pipeline initialization on the buffered frames (e.g., template calculation, shading correction, and mask generation).

Once initialization completes, the system enters its online mode, where the following stages are run in a pipelined fashion:

- Motion-correction, trace extraction and spike detection are executed concurrently in separate threads, consuming frames from the shared buffer.
- Output (traces and spikes) is written to disk asynchronously, decoupling computation from I/O latency.

Shared memory buffers are used between stages to decouple the producer and consumer threads. This ensures that temporary delays (e.g., in writing to disk or during GPU-inference steps) do not stall upstream modules. The use of fixed-size buffers and thread-safe queues also enables dynamic flow control and prevents data loss during peak frame arrival times.

5.3 Profiling-driven optimization

The motion-correction component was originally implemented using normalized cross-correlation between each incoming frame and a static reference template, computed either in the spatial domain (`tf.nn.conv2d`) or the frequency domain (via FFTs). While effective for early testing, the initial implementation was not viable for real-time operation due to redundant computation, inefficient tensor operations, and limited GPU utilization.

Following the profiling analysis presented in Section 4.2.5, the motion-correction layer was restructured around a GPU-optimized `MotionCorrect` TensorFlow module. This refactoring focused on eliminating bottlenecks, improving batch throughput, and enabling static graph execution. A visual summary of the architectural differences is shown in Figure 5.2.

The most significant implementation changes include:

- **Template FFT reuse:** The FFT of the static template is now computed once during layer construction and reused across all input batches. This prevents redundant recomputation and reduces memory bandwidth usage.
- **Depthwise Vignette Gaussian filtering:** Input frames can optionally be smoothed using separable 1D Gaussian filters, implemented with `tf.nn.depthwise_conv2d`. This reduces the cost of smoothing from quadratic to linear in kernel size and enables efficient GPU execution over entire batches.
- **Optimized local normalization:** The `normalize_image` routine was rewritten to compute local mean and variance using depthwise separable convolutions. This replaces reduction-based methods with fused operations and improves memory locality.
- **Subpixel shift estimation:** Gaussian interpolation is used to refine the correlation peak and extract fractional-pixel motion vectors. This improves alignment accuracy without increasing the computational burden.
- **Graph compilation and type constraints:** The entire operation is compiled with `@tf.function` to enable XLA graph optimizations such as kernel fusion and operation reordering. To support this, the internal FFT data type was changed from `complex128` to `complex64`, which was necessary for compatibility but did not degrade alignment performance.

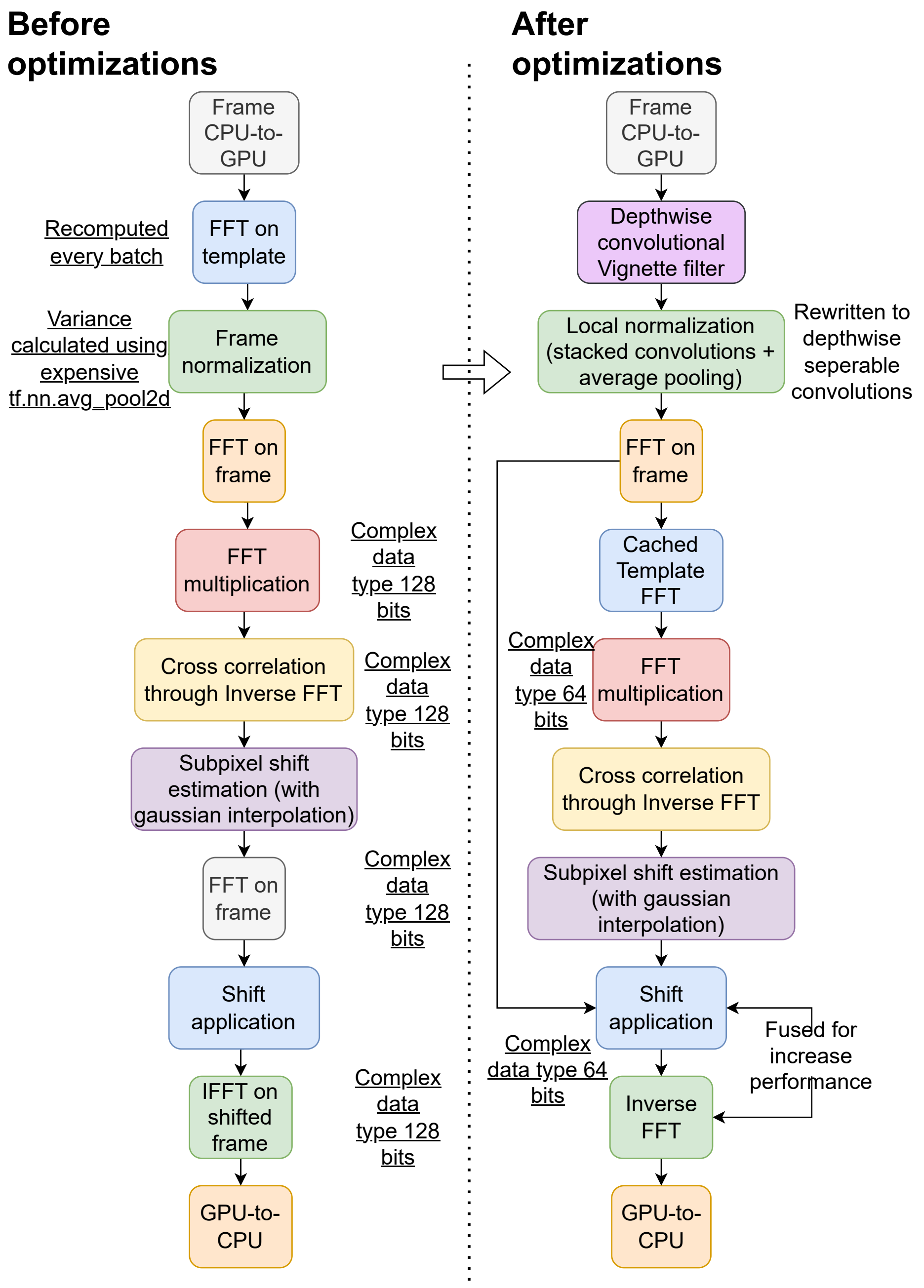

Before optimizations
Frame CPU-to-GPU
Recomputed every batch
FFT on template
Variance calculated using expensive tf.nn.avg_pool2d
Frame normalization
FFT on frame
FFT multiplication
Complex data type 128 bits
Cross correlation through Inverse FFT
Complex data type 128 bits
Subpixel shift estimation (with gaussian interpolation)
FFT on frame
Complex data type 128 bits
Shift application
IFFT on shifted frame
Complex data type 128 bits
GPU-to-CPU
After optimizations
Frame CPU-to-GPU
Depthwise convolutional Vignette filter
Local normalization (stacked convolutions + average pooling)
Rewritten to depthwise seperable convolutions
FFT on frame
Cached Template FFT
Complex data type 64 bits
FFT multiplication
Cross correlation through Inverse FFT
Subpixel shift estimation (with gaussian interpolation)
Shift application
Fused for increase performance
Complex data type 64 bits
Inverse FFT
GPU-to-CPU

Figure 5.2: Schematic comparison of the motion-correction pipeline before and after profiling-driven optimization. The new implementation enables FFT reuse, optional GPU-side vignette filtering, fused normalization, and shift application within a compiled TensorFlow graph.

5.4 Usability and generalizability

Beyond raw performance, a practical imaging pipeline must be easy to deploy, adapt, and maintain across varied lab environments. This section highlights implementation features that enhance usability and extend the pipelines applicability to different hardware, workflows, and development contexts. Key contributions include a fully functional demonstration environment that simulates real-time acquisition without physical hardware, containerized deployment for reproducibility, modular control over individual processing stages, and an automatic calibration module that adapts runtime parameters, such as batch size and thread count, to the available system. Together, these features ensure robust, hardware-agnostic deployment in both research and production settings.

5.4.1 Demonstration Environment and Simulation

To support testing, validation, and demonstration without requiring a physical imaging setup, a full-featured simulation environment was implemented. This demo replicates the behavior of a live camera by generating synthetic frame streams and writing them to a shared memory-mapped file, which is then consumed by the online pipeline running inside a Docker container.

The simulation framework uses a `sender_thread` to emulate camera behavior by writing frames incrementally to disk and transmitting metadata (e.g., timestamps) via ZMQ sockets. The pipelines `receiver_thread` monitors the shared file and processes each frame in real time, allowing seamless substitution of the simulated stream for actual camera input.

This design mirrors the camera-to-GPU data flow used in production systems, ensuring that all components, memory buffers, synchronization primitives, and GPU pipelines, are tested under realistic timing conditions. As shown in Figure 5.3, this architecture enables the entire online pipeline to operate as if connected to a real acquisition device.

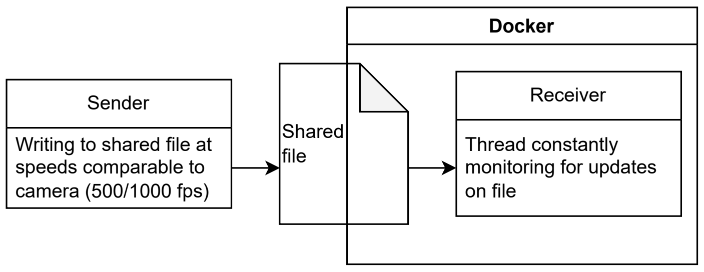

Docker
Sender
Writing to shared file at speeds comparable to camera (500/1000 fps)
Shared file
Receiver
Thread constantly monitoring for updates on file

Figure 5.3: Overview of the demo simulation setup. A sender process emulates the camera by writing to a memory-mapped file, which is mounted into the Docker container. The receiver inside the container reads frames in near real time.

The demo environment plays a critical role in achieving the goals outlined in Section 4.2.4 on generalizability and usability:

- **Portability:** It enables rapid testing of the pipeline on new systems or GPU configurations without requiring specialized hardware.

- **Debugging and benchmarking:** Developers can replicate exact conditions across systems, simplifying profiling and regression testing.
- **Demonstration:** The demo supports live presentations or tutorials in scenarios where deploying actual hardware is impractical or infeasible.

Together, these features make the pipeline easier to deploy, extend, and validate in diverse research and development contexts.

5.4.2 Containerized Deployment via Docker

To ensure reproducibility across systems and simplify setup, the entire processing pipeline was containerized using Docker. The container bundles all dependencies, including CUDA, TensorFlow, Python libraries, and model weights, into a portable runtime image that can execute identically across lab desktops, high-performance clusters, and cloud nodes.

The Docker build process is organized into multiple stages to ensure modularity and minimize image size:

- **Base stage:** Begins from `nvidia/cuda:12.1.0-base-ubuntu22.04`, providing the required CUDA runtime and a compatible Linux distribution.
- **System dependency installation:** A separate build stage installs core packages such as `gcc-9`, `libomp-dev`, `ffmpeg`, and `python3-pip`, required for both compilation and runtime support.
- **Dual CUDA support:** To ensure compatibility with both TensorFlow (requiring CUDA 12.1) and legacy VOLTAGE components (built against CUDA 11.0), two CUDA toolkits are installed side-by-side. Symlinks and environment variables allow runtime switching between toolchains.
- **Miniconda environment:** A dedicated stage installs Miniconda and creates a custom environment (`rtvi`) using an `environment.yml` file. This isolates Python dependencies and facilitates version control. The container automatically activates the environment for all subsequent commands via the `SHELL` directive.
- **XLA and GUI support:** Environment variables are set to support GUI features (e.g., OpenCV via Qt on XCB) and TensorFlow's XLA compiler. These settings ensure compatibility when executing precompiled models or visual tools.
- **Sanity checks:** The final stages include diagnostic calls (e.g., `import numpy`) to verify that all layers were successfully built and that GPU acceleration is enabled within the activated environment.
- **Entrypoint:** The container launches into a Bash shell with the environment fully configured, allowing developers to run the offline or online pipeline directly without additional setup.

The final image runs via `nvidia-docker` and supports GPU passthrough. All CUDA and cuDNN libraries are statically bundled, ensuring independence from host drivers and enabling consistent behavior regardless of the execution platform.

5.4.3 Hardware-aware calibration

To prepare the pipeline for real-time deployment under diverse hardware constraints, a systematic calibration procedure was developed. This process involves iteratively varying key parameters, including batch size, number of processing threads, and spatial scaling factor, and evaluating their impact on runtime performance and stability.

The full calibration logic is formalized in Algorithm 2. For each candidate configuration, the pipeline is benchmarked on a fixed-length input stream, and the average per-frame latency is measured. Configurations that exceed a predefined latency threshold (e.g., 50 ms) are discarded, and the setup with the lowest valid latency is selected.

This approach enables automated tuning of the pipeline for a wide range of GPU and CPU configurations, ensuring low-latency operation and sustained throughput. It also exposes performance ceilings under constrained conditions, guiding future hardware choices and system scaling.

Algorithm 2: Hardware-Aware Calibration Procedure

```
Require: Candidate batch_sizes, thread_counts, scaling_factors, performance
    threshold latency_max
Ensure: Optimal configuration: (batch_size, thread_count, scale)
  best_config ← None, min_latency ← $\infty$
  for all scale in scaling_factors do
    Increase image size based on base resolution and scale factor scale
    for all batch_size in batch_sizes do
      for all thread_count in thread_counts do
        Configure pipeline with current scale, batch_size, and thread_count
        Run profiling on a fixed-length input stream
        Measure average per-frame latency $\ell$
        if $\ell$ < latency_max and $\ell$ < min_latency then
          min_latency ← $\ell$
          best_config ← (batch_size, thread_count, scale)
        end if
      end for
    end for
  end for
  return best_config
```

Chapter 6

Results and Evaluation

Having detailed the implementation of the hybrid voltage imaging pipeline in the previous chapter, we now turn to evaluating its real-world performance. This chapter presents a comprehensive assessment of the system, focusing on both computational efficiency and signal fidelity.

The evaluation is structured around the pipelines two operational modes, offline and online, and reflects realistic workloads and hardware constraints. Key metrics include segmentation and spike detection accuracy, per-frame latency, scalability, and GPU memory usage. The results are presented in the following order:

- **Experimental setup:** Description of hardware, benchmarking procedures, and datasets used throughout the evaluation.
- **Trace extraction comparison:** Assessment of using FIOLA's trace extraction with VOLTAGE masks in place of FIOLAs native strategy, highlighting the impact on spike detection accuracy.
- **Effect of vignette filtering:** Analysis of Gaussian vignette filtering (Section 4.2.4), showing its impact on segmentation F1 scores and motion-correction performance.
- **Offline implementation timeline:** Chronological, by implementation, evaluation of implementation-level optimizations in the offline pipeline, beginning with:
  - TensorFlow data pipeline acceleration,
  - GPU-accelerated median filtering (Section 5.2.1),
  - Followed by an assessment of individual component performance as a function of spatial resolution, batch size, and thread count.
- **Offline pipeline performance:** End-to-end runtime performance and throughput analysis of the complete offline pipeline under various scaling conditions.
- **Initial online evaluation:** Evaluation of the unoptimized online pipeline, which does not yet meet the 500-1000 FPS target, followed by performance on a high-end GPU as an upper-bound baseline.
- **Post-optimization online performance:** Results after integrating motion-correction optimizations described in Section 4.2.5, demonstrating improved latency and throughput.

- **GPU memory profiling:** Detailed breakdown of memory usage across online pipeline stages, varying spatial resolution and batch size to expose hardware limits and guide configuration.

Together, these results validate the hybrid pipelines practical viability and demonstrate how targeted design and implementation strategies yield a system that balances accuracy, speed, and generalizability across real-world neuroscience workloads.

6.1 Experimental Setup

All primary performance evaluation experiments were conducted on the CUBE server configuration described in Table 4.1, using representative recordings from the HPC2 dataset [1]. Offline testing was performed using the complete hybrid pipeline, including all retained algorithmic enhancements (e.g., vignette filtering). The wavelet-based trace filtering described in Chapter 4.2.4 was not included in the final evaluation, as it was found to have limited impact on spike detection accuracy and was therefore removed from the active pipeline configuration.

To assess performance across different spatial resolutions, synthetic inputs were generated by vertically stacking a single recording. Specifically, the original frame of size 116×498 pixels was duplicated n times along the vertical axis, where n is the scaling factor. This method preserved the input structure while simulating increased resolution workloads. A scaling factor of 1 corresponds to the original 116×498 frame; a factor of 2 yields a 232×498 frame; and so on. This setup enabled controlled increases in pixel count, allowing fair comparisons of GPU throughput and memory usage over a known range of resolutions. Table 6.1 summarizes the scaling factors, including total pixel counts and the side length of an equivalent square image (for easier interpretation).

Table 6.1: Spatial scaling factors used in evaluation, showing the resulting frame dimensions, total pixel count, and side length of an equivalent square image.

Scaling Factor	**Frame Size (H $\times$ W)**	**Total Pixels**	**Square Equivalent ($\sqrt{d}$)**
1	116×498	57,768	240
2	232×498	115,536	340
4	464×498	231,072	480
8	928×498	462,144	680
16	1856×498	924,288	960
32	3712×498	1,848,576	1359

Preliminary evaluation of motion-correction was also conducted on an alternative hardware setup (the Lab PC), detailed in Table 4.4. While the Lab PC was useful in testing GPU performance across different architectures (Volta vs. Ampere), it was unable to support motion-correction above a n = 4 spatial scaling factor due to VRAM limitations (12 GB vs. 32 GB on the CUBE server). As a result, all high-resolution and final performance evaluations were conducted exclusively on the CUBE server, which is shown in Table 4.1, to ensure consistency and eliminate memory-related constraints

6.2 Trace Extraction

As established in Section 4.1.6, FIOLA's ridge-regression method significantly outperforms VOLTAGE's z-score-based spike detection, particularly in low-SNR conditions. To ensure that this advantage is retained in the hybrid pipeline which uses VOLTAGE's trace extraction instead of FIOLA's, F1 performance was re-evaluated using FIOLA's detector operating on traces extracted via VOLTAGE's modular segmentation. Spike detection was evaluated on both the HPC2 and L1 datasets using amplitude-binned F1 scores. The results showed that the hybrid configuration reproduced FIOLA's original accuracy characteristics, with no observable drop in sensitivity or precision.

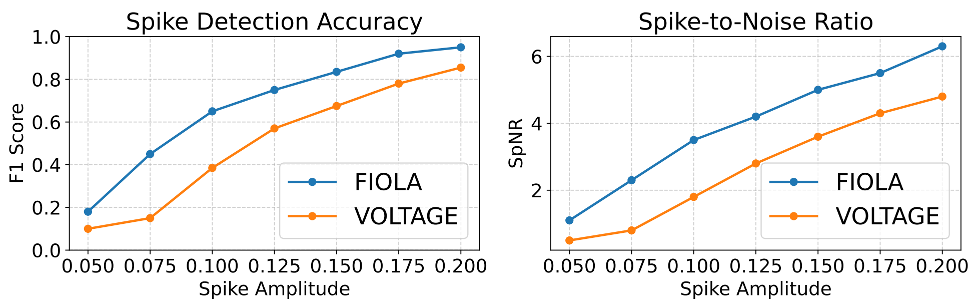


Figure 6.1: Spike detection F1 scores across spike amplitude bins. The hybrid pipeline retains the accuracy benefits of FIOLA's adaptive regression model.

This confirms that FIOLA's spike inference generalizes well to downstream integration in the hybrid pipeline, and that the use of VOLTAGE's trace-extraction step does not compromise detection fidelity. .

6.3 Vignette Filtering

To understand how the Vignette filter contributes to improved downstream accuracy, we next evaluate its specific impact on segmentation performance. By isolating the effects of motion-correction and pre-filtering in otherwise identical pipelines, we can attribute gains in segmentation quality to individual components with greater confidence.

6.3.1 Segmentation Accuracy

To evaluate the impact of motion-correction enhancements on segmentation quality, F1 scores were compared across three pipeline configurations:

1. The original VOLTAGE pipeline with its default motion-correction and U-Net segmentation.

2. The hybrid pipeline using FIOLA's motion-correction and VOLTAGE's U-Net segmentation.

3. The hybrid pipeline using FIOLA motion-correction preceded by vignette filtering (Section 4.2.4).

Evaluations were performed on two datasets published by the same group: HPC2 and L1 [1], both containing ground truth masks and pre-aligned frames suitable for benchmarking. Each configuration was tested on both datasets using identical settings, and average F1 scores were computed per dataset by comparing frame-level segmentation outputs to ground truth ROIs.

Table 6.2: Segmentation F1 scores across pipeline configurations and datasets. The hybrid pipeline consistently improves accuracy, with additional gains from vignette filtering.

Pipeline Configuration	**HPC2 (F1)**	**L1 (F1)**
VOLTAGE (baseline)		
Average F1	0.58	0.71
Best F1	0.78	0.86
Worst F1	0.36	0.61
Hybrid (FIOLA motion-correction)		
Average F1	0.62	0.82
Best F1	0.86	0.88
Worst F1	0.41	0.68
Hybrid + Vignette Filter		
Average F1	0.67	0.85
Best F1	0.87	0.91
Worst F1	0.55	0.76

As shown in Table 6.2, both the HPC2 and L1 datasets show consistent improvements in segmentation accuracy when switching from the VOLTAGE pipeline to the hybrid pipeline with FIOLA motion-correction. The addition of vignette filtering yields further gains of approximately 0.01-0.06 F1 points.

Importantly, the benefit of vignette filtering was not uniform across all recordings. In cases where baseline segmentation F1 scores were already high (e.g., above 0.85), the improvement from vignette filtering was modest. However, in recordings with blurred or low-contrast frames, where baseline F1 scores were significantly lower (e.g., 0.65-0.75), the vignette-enhanced pipeline consistently raised scores by 0.10-0.15 points.

6.3.2 Performance Trade-Off of Vignette Filtering

While vignette filtering significantly improves segmentation accuracy, particularly in low-contrast or blurry recordings, it introduces a measurable performance overhead in the motion-correction stage. This trade-off was quantified by comparing the total motion-correction time with and without vignette filtering, using otherwise identical configurations.

As shown in Figure 6.2, the vignette-enhanced pipeline consistently requires more time per frame during motion-correction. This increase is due to the additional convolution on each incoming frame. While the cost varies slightly with batch size and resolution, the overhead ranges from 15-25% in typical configurations. Despite this cost, the improvement in segmentation F1 score (up to 0.10–0.15 in worst-case conditions) justifies the inclusion of the vignette filter in most scenarios.

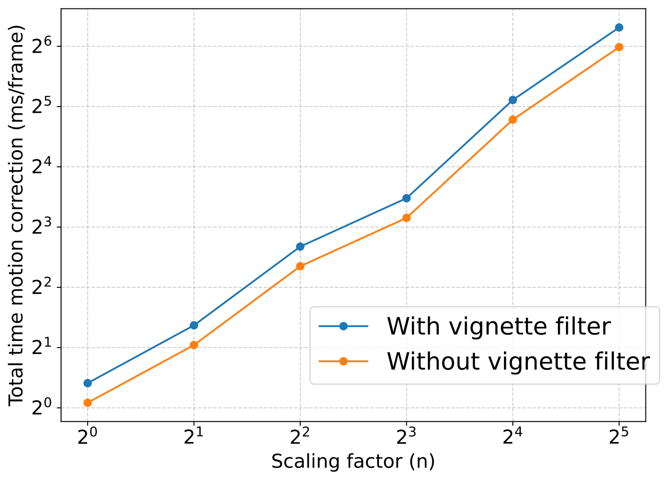


Figure 6.2: Comparison of motion-correction runtime with and without vignette filtering. The added cost stems from pre-filtering each frame prior to alignment.

6.4 Offline pipeline

The following sections examine the performance characteristics of the offline pipeline and document the implementation-driven optimizations applied throughout its development. These changes were introduced incrementally and validated through stepwise profiling. We begin with a key improvement to throughput using TensorFlows data pipeline API, followed by enhancements to GPU-accelerated median filtering, and continue with a stage-by-stage evaluation of each pipeline component. This is followed by an analysis of the full offline pipeline performance under varying conditions.

6.4.1 TensorFlow Data Optimization

To assess the effectiveness of the TensorFlow data pipeline improvements (Section 5.1.1), vignette-filtered runs were compared with and without the optimized `tf.data` configuration. The evaluation focused on per-frame latency during motion-correction, as motion-correction was the only affected component. As shown in Figure 6.3, the optimized pipeline results in a speed up between 1.78 and 3.32, having more impact on bigger figures. This gain is primarily attributed to overlapping CPU-side preprocessing and GPU-side execution, enabled by asynchronous data loading and multi-threaded mapping functions.

Importantly, this improvement was achieved without modifying the computational graph or changing model logic. Instead, the benefit derives purely from more efficient input delivery to the GPU, which helped maintain near-constant GPU occupancy, even during trace-heavy or template-initialization phases. Given the relatively low implementation complexity and general applicability of this optimization, it was adopted as a standard configuration for all subsequent experiments. This change is especially beneficial in online or streaming

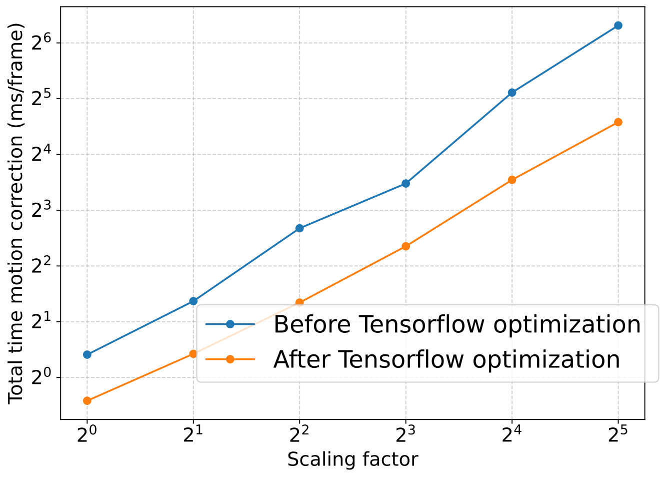


Figure 6.3: Per-frame runtime of vignette-filtered pipeline runs with and without TensorFlow data pipeline optimization. Improvements are primarily due to reduced data-loading overhead.

modes, where idle GPU time directly increases latency and impacts real-time responsiveness.

6.4.2 GPU-Accelerated Median Filtering

The GPU-based implementation of the median filter led to substantial improvements in template generation time. Figure 6.4 compares the total runtime of the CPU and GPU implementations on representative datasets from the HPC2 collection. The median was computed across 15,000 frames with a resolution of 116 $\times$ 496 pixels, representative of typical use cases.

The CPU-based median computation required over 25 seconds to complete the full template calculation, consistent with its $O(d \cdot L \log L)$ complexity. In contrast, the GPU-accelerated implementation completed in just 2.7 seconds, yielding a speedup of over 9$\times$. This improvement is attributable to the highly parallel nature of the reduction-style computation and optimized GPU memory access patterns used in the TensorFlow-based implementation. Despite the increased complexity of implementing a GPU-based approximation to true median computation, the runtime savings are substantial enough to justify its use even in single-batch or offline modes. This improvement also opens the door to recomputing the motion-correction template dynamically within online pipelines, something previously infeasible under the CPU-based approach.

6.4.3 Runtime Analysis

To understand the performance characteristics of the hybrid pipeline under varying operational conditions, a series of evaluation experiments were conducted as part of the calibra-

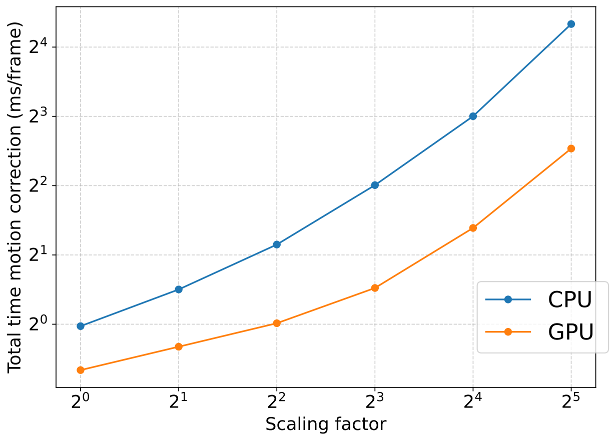


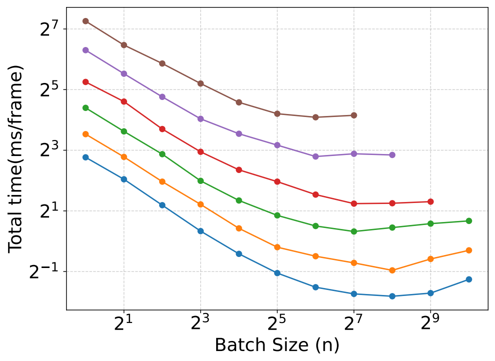
Figure 6.4: Runtime comparison of CPU vs. GPU median filtering for template generation. GPU implementation reduces computation time by over 90%.

tion process (see Section 5.4.3). These tests evaluated how runtime scales with batch size, thread count, and spatial resolution across key computational stages. The evaluation was carried out on the CUBE server using the HPC2 dataset unless otherwise noted.

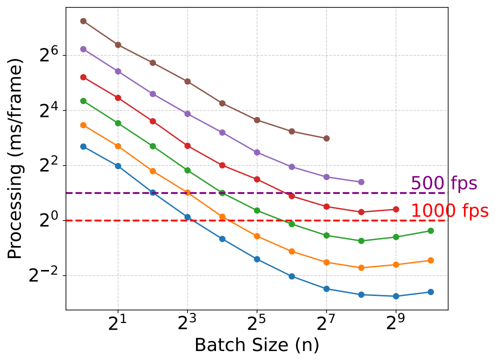
Motion-Correction Evaluation

Motion-correction was the most computationally intensive streaming-stage component and was therefore profiled in detail. Tests were conducted across a range of batch sizes and spatial scaling factors, with measurements taken for:

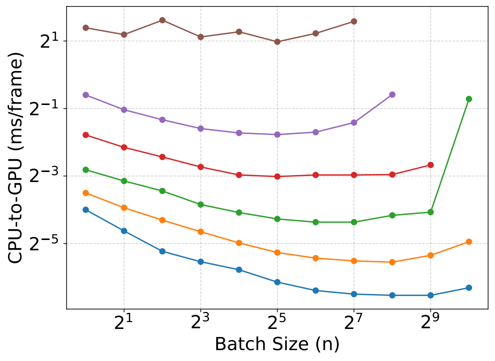
- Total runtime per batch,
- GPU kernel execution time,
- CPU-to-GPU memory transfer time,
- GPU-to-CPU output retrieval time.



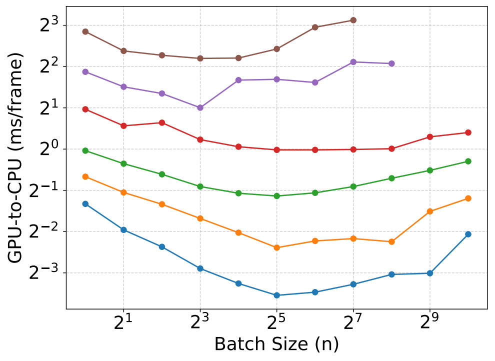
(a) Total runtime per batch.



(b) GPU kernel execution time.



(c) CPU-to-GPU transfer time.



(d) GPU-to-CPU readback time.

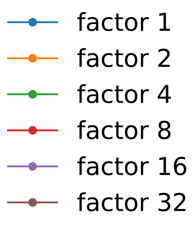


Figure 6.5: Detailed profiling of motion-correction across batch sizes and spatial scaling factors. Each quadrant shows a different performance metric: (a) total runtime, (b) GPU kernel time, (c) input transfer time, and (d) output readback time. Batching yields significant speedups, with optimal ranges depending on resolution.

Figure 6.5 presents these measurements across four subplots. Across all conditions, processing frames individually results in the highest latency and poorest GPU utilization. Batching significantly improves throughput, particularly between 64-256 frames. However, extremely large batches can regress in performance due to memory or kernel scheduling overhead.

These results informed the default batching configuration for the hybrid pipeline and reinforce the importance of tunable execution in real-time environments.

Shading Correction Performance

Shading correction is a lightweight but necessary preprocessing step applied to each frame following motion-correction. During calibration, this stage was evaluated on its scalability across thread pool sizes and spatial resolutions.

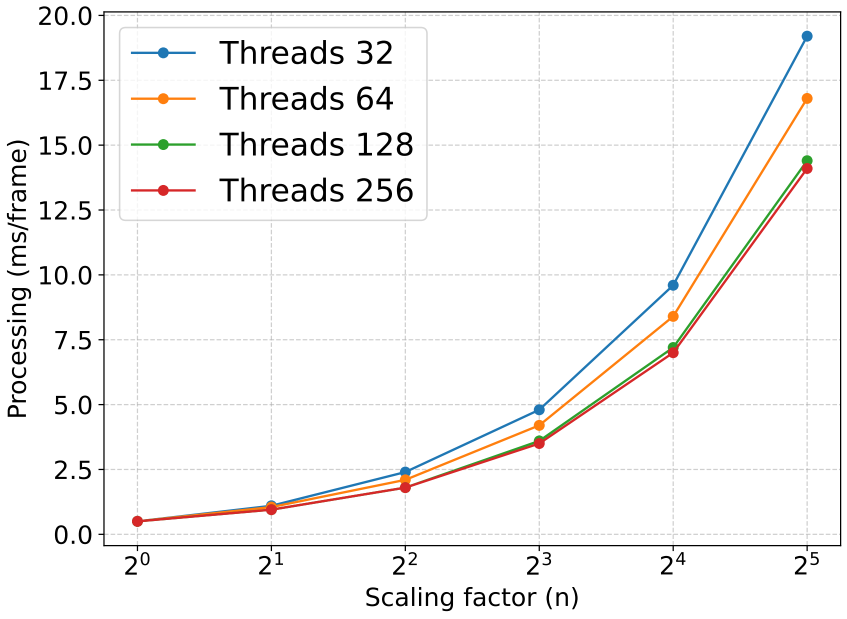


Figure 6.6: Shading correction runtime across thread counts and spatial resolutions. Runtime is measured per frame.

As shown in Figure 6.6, performance improves when increasing thread count up to 128 threads. However, beyond this point, further gains become negligible across all tested image sizes, indicating a saturation in CPU-side parallelism, likely due to memory access limits or thread-management overhead.

Notably, although the shading correction module is not a major bottleneck, it was found to be *slower* in the hybrid pipeline than in the original VOLTAGE implementation. In VOLTAGE, shading and motion-correction were fused into a single CUDA kernel, resulting in a total runtime of approximately 1.11 seconds for the full shading operation. In contrast, the hybrid pipeline separates these stages, and the standalone shading correction step alone required 5.18 seconds at the default spatial resolution, using 15000 frames. This regression stems from a modular restructuring: the decoupling was necessary to allow FIOLA's motion-correction to be integrated in place of VOLTAGE's original kernel. However, the resulting shading pass uses a more generic, unsophisticated CUDA implementation, incurring additional memory transfers between stages. This trade-off was accepted to preserve modularity and maintain cross-pipeline compatibility, but it identifies a clear area for future kernel-level optimization.

Preprocessing Performance

In addition to motion-correction, upstream preprocessing steps, including shading correction, temporal filtering, and summary image generation, were also profiled during the calibration phase. These operations are primarily CPU-bound and were evaluated as a function of thread pool size and spatial resolution.

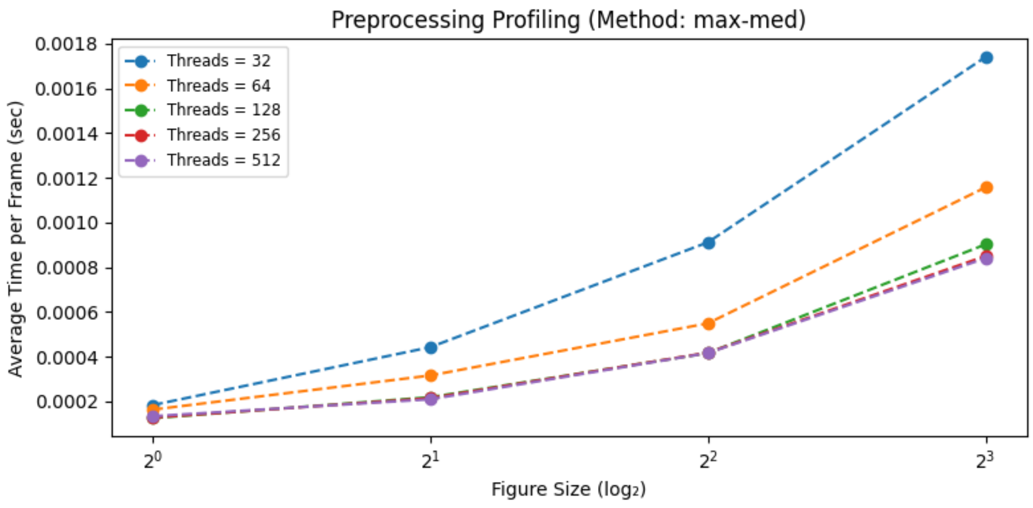


Figure 6.7: Preprocessing runtime across different thread pool sizes and spatial resolutions. Runtime is normalized per frame.

As shown in Figure 6.7, increasing the thread count significantly improves performance up to approximately 128 threads. The improvement from 32 to 128 threads nearly doubles preprocessing throughput across all tested resolutions. However, gains diminish beyond that point, particularly for higher-resolution inputs, indicating that memory access and scheduling overheads begin to dominate past this threshold.

At large spatial resolutions, multi-threading becomes increasingly important to keep per-frame latency below the thresholds required for real-time operation. These findings informed the choice of thread pool sizes used in the hybrid pipeline's initialization phase and highlight the need for hardware-aware tuning in CPU-intensive components.

Segmentation Performance

Segmentation was profiled to evaluate how batch size affects the throughput of the U-Net inference model used in the VOLTAGE segmentation stage. While segmentation is GPU-accelerated and benefits from batched execution, excessive batch sizes can introduce memory contention or kernel scheduling inefficiencies.

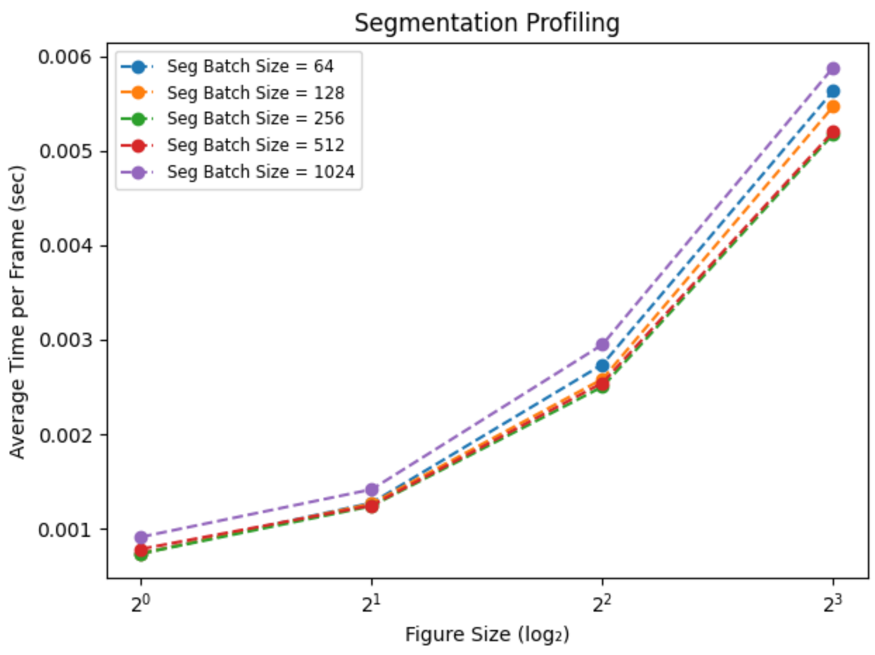


Figure 6.8: Segmentation runtime as a function of batch size. Runtime shown per frame, averaged over entire input volume.

As shown in Figure 6.8, runtime performance improves with batch size up to a point, but does not follow a linear scaling trend. A batch size of 64 serves as a baseline, and increasing to 128 and 256 results in lower per-frame processing time. However, performance begins to degrade beyond this point: batch size 512 is slightly slower than 256, and batch size 1024 is noticeably worse than all smaller configurations.

These results indicate that extremely large batches are suboptimal for this U-Net model, likely due to GPU memory saturation or internal overhead related to tensor allocation and execution graph complexity. In particular, large batches may exceed the cache capacity of shared memory or saturate tensor core pipelines, resulting in degraded parallel efficiency.

Based on these findings, batch sizes between 128 and 256 are recommended for segmentation tasks, offering the best balance of throughput and memory stability without oversaturating GPU resources.

Mask computation, trace extraction and Spike Detection

The ROI mask generation, trace extraction and spike detection were also included in the overall evaluation, but no performance calibration was performed on these components. This is because they do not expose configurable runtime parameters such as batch size or thread count within the current implementation.

Demixing is a lightweight post-segmentation operation that generates ROI masks from predicted segmentation maps. It completes in under 1 ms per frame across all tested inputs and resolutions. Given its simplicity and negligible runtime, further optimization or evaluation was deemed unnecessary. Similarly, spike detection, both in offline batch mode and online streaming mode, showed stable and efficient performance under default settings. The offline mode performs batch inference after trace extraction, while the online mode uses an incremental ridge-regression model with minimal per-frame cost following initialization. In both cases, the absence of tunable runtime parameters made these stages straightforward to deploy and excluded them from parameter sensitivity analysis.

As such, these stages were omitted from the detailed calibration figures but are included in total pipeline runtime measurements reported in later sections.

Calibration Summary and Impact

To consolidate the effects of runtime parameter tuning, the pipeline's per-frame performance was compared across three representative configurations.

- **Worst case:** Unreasonably small batches (e.g., single-frame processing), minimal threads, and suboptimal scaling, used to illustrate the lower bound of performance.
- **Default case:** Reasonable defaults similar to VOLTAGE's original configuration, without calibration.
- **Best case:** Fully calibrated configuration based on evaluation results, using optimized batch sizes, thread counts, and resolution settings.

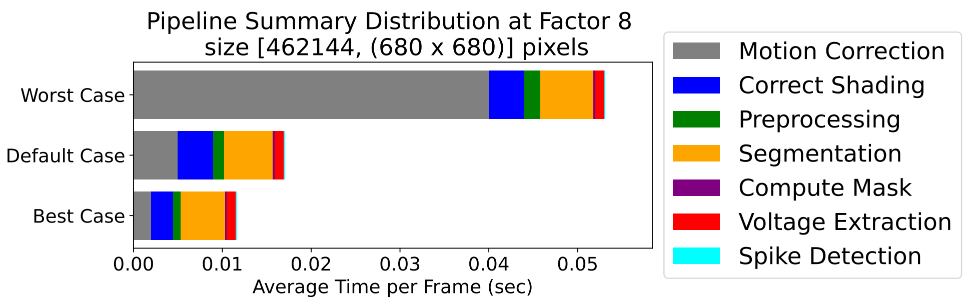


Figure 6.9: Per-frame runtime comparison between worst-case, default, and calibrated (best-case) pipeline configurations. Runtime is decomposed by stage.

As shown in Figure 6.9, using the wrong runtime settings can result in over a 5 × increase in processing time per frame compared to the best-case configuration. Even the default setup, while functionally correct, is approximately 1.5 × slower than the fully optimized version. This illustrates how critical calibration is for systems that operate under real-time constraints.

The breakdown of best-case runtime also reveals the dominant cost components in the optimized pipeline. When correctly configured, segmentation, motion-correction and shading correction consume the largest share of compute time, followed by preprocessing.

Meanwhile, demixing, trace extraction, and spike detection remain lightweight, confirming that further optimization effort for the offline stage is best focused on the preceding pipeline stages.

6.4.4 Total Offline Pipeline Performance

To quantify the impact of algorithmic and implementation improvements on full pipeline execution, the offline pipeline was benchmarked across a range of spatial resolutions and input durations (frame counts). Results are compared against the original, unoptimized version of the pipeline to highlight practical runtime gains under realistic conditions.

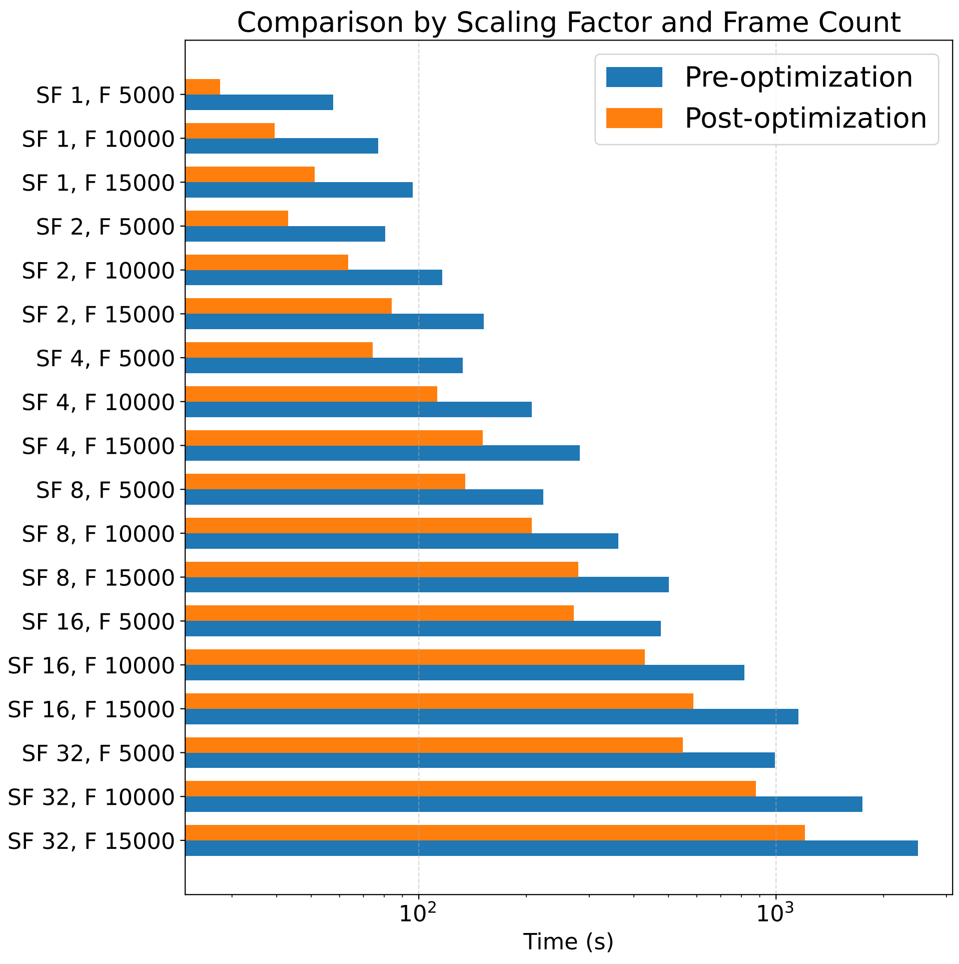


Figure 6.10: Total runtime of the offline pipeline before and after optimization.

As shown in Figure 6.10, the optimized pipeline consistently outperforms the original implementation across all tested configurations. Speedups are especially pronounced at lower resolutions and shorter video lengths, where computational overhead was previously an issue.

In some cases, total runtime was reduced by more than factor 4. While for longer videos with higher resolutions it is a factor 2.

6.5 Online pipeline

This section evaluates the hybrid pipelines performance under real-time streaming conditions, where frame-by-frame responsiveness and minimal latency are essential. Unlike the offline mode, which emphasizes throughput and batch processing, the online pipeline must balance GPU efficiency with tight timing constraints. The focus here is on quantifying latency, analyzing how batch size and resolution affect performance, and identifying which components limit real-time feasibility.

We begin by profiling the baseline performance of the core online stages, motion-correction, trace extraction, and spike detection, under various batch sizes. This is followed by resolution scaling tests and hardware comparisons to determine where further optimizations are needed.

6.5.1 Initial Performance

To evaluate the responsiveness and throughput of the hybrid pipeline in streaming mode, the core online stages, motion-correction, trace extraction, and spike detection, were profiled. These components represent the minimal processing required to deliver real-time feedback. Performance was measured as a function of batch size, ranging from 1 (strict frame-by-frame streaming) to 512, increasing in powers of two.

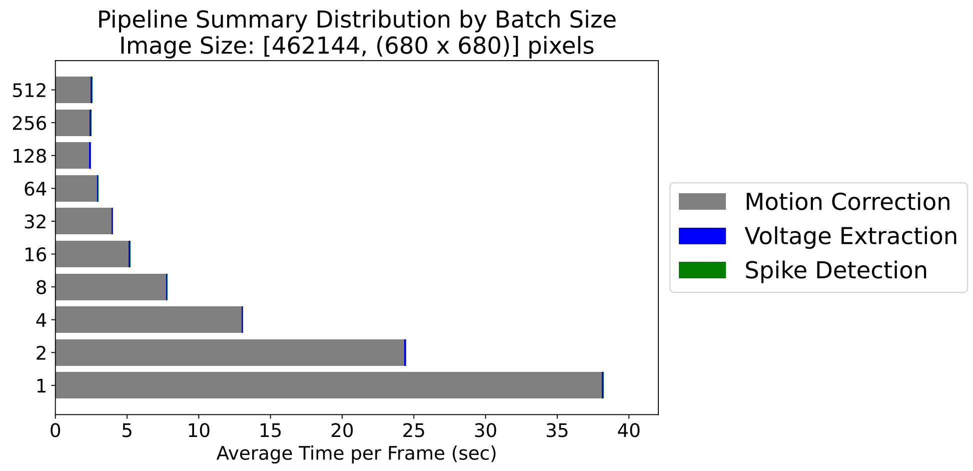


Figure 6.11: Per-frame latency for the online pipeline stages (motion-correction, trace extraction, spike detection) across batch sizes. Results are shown for a spatial scaling factor of 8 but are representative of overall trends across different resolutions.

As shown in Figure 6.11, single-frame processing incurs the highest per-frame latency, largely due to repeated kernel launches and I/O overhead that cannot be amortized. Increasing the batch size to 4 or 8 yields an immediate reduction in latency per frame, as operations are batched more efficiently on the GPU.

Although the figure reflects performance at a spatial scaling factor of 8, this latency vs. batch size trade-off is consistent across all tested scaling factors. In each case, small batches favor low-latency real-time responsiveness, while larger batches offer greater computational efficiency at the cost of delayed feedback.

To further illustrate this relationship, Figure 6.12 shows the per-frame latency of the online pipeline as a function of scaling factor, using the batch size that yielded the best latency for each resolution. As expected, latency increases with spatial resolution due to the growing number of pixels per frame. However, the trend remains smooth and within real-time constraints when batching is tuned appropriately.

Importantly, the results also show that motion-correction dominates the total latency in the online processing stage. Trace extraction and spike detection contribute only a small fraction of the total per-frame cost, confirming that further latency reduction efforts should prioritize optimizing the motion-correction stage, either through kernel improvements or more aggressive batching strategies.

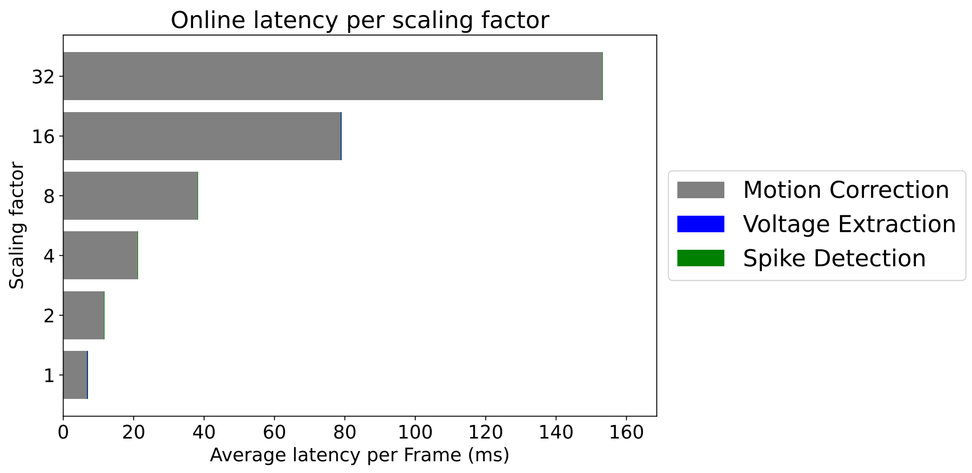


Figure 6.12: Best-case online pipeline latency per frame as a function of spatial scaling factor. Each point uses the optimal batch size for that resolution.

Real-Time Constraints

In many experimental and closed-loop neuroscience applications, real-time performance is defined in terms of throughput targets, such as sustaining 500 or even 1000 FPS. These thresholds correspond to maximum per-frame latencies of 2 ms and 1 ms, respectively. However, achieving such low latencies is challenging due to the computational cost of motion-correction and data transfer operations, particularly at high spatial resolutions.

To explore this constraint, online pipeline latency was measured across both batch sizes and spatial scaling factors. Figure 6.13 overlays two reference lines at 2 ms and 1 ms to indicate the performance boundaries for 500 FPS and 1000 FPS operation, respectively.

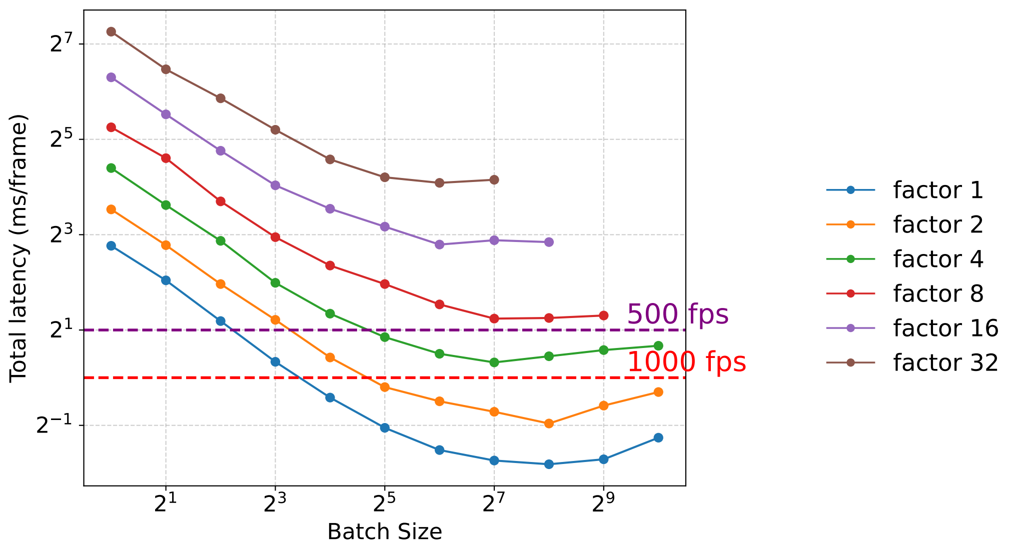


Figure 6.13: Per-frame latency for the online pipeline across batch sizes and scaling factors on the Tesla V100. Dashed horizontal lines represent 2 ms (500 FPS) and 1 ms (1000 FPS) thresholds.

The results highlight an important constraint: while single-frame processing offers the lowest theoretical latency, it is not computationally efficient enough to meet high-throughput requirements. Instead, achieving 500 FPS or 1000 FPS typically requires bigger batch sizes, which increase the per-frame latency extensively but enable much higher GPU efficiency.

At lower spatial resolutions (scaling factors ≤ 4), the pipeline can achieve 1000 FPS performance using batch sizes of 8–16. However, at higher resolutions (scaling ≤ 8), only 500 FPS is sustainable without exceeding the 2 ms frame budget. Even then, performance must be carefully tuned: too small a batch wastes compute resources, while too large a batch introduces excessive latency due to delayed output availability.

These findings reinforce the importance of calibration in real-time deployments. In practice, a compromise in per-frame latency is necessary to sustain high frame rates under realistic hardware constraints.

Hardware Scaling

To evaluate how the hybrid pipeline's online performance scales with hardware improvements, the evaluation experiments were repeated on a third system featuring a newer-generation GPU: the NVIDIA H100 NVL. Key hardware specifications of this system are shown in Table 6.3.

Table 6.3: Hardware specifications for H100-based evaluation system.

Component	**H100 System**
CPU	AMD EPYC 9334 (32-core, 3.9 GHz)
GPU	NVIDIA H100 NVL, 96 GB
RAM	755 GB DDR5
Disk	Dell PM9A3 NVMe SSDs (2 $\times$ 3.84 TB + 1 $\times$ 447 GB)
OS	Ubuntu 22.04.3 LTS

Despite the substantial GPU upgrade compared to the CUBE server's Tesla V100, the H100 system was still unable to execute motion-correction in strict frame-by-frame mode (batch size = 1) at any of the tested spatial resolutions. This finding underscores the computational intensity of the frequency-domain motion-correction algorithm and highlights the limits of single-frame efficiency, even on top-tier hardware.

However, when modest batching (e.g., batch size 4–16) was introduced, the H100 system outperformed the V100 setup across all spatial resolutions. Specifically:

- Real-time operation ($\leq$ 2 ms per frame) was sustained up to a spatial scaling factor of 16.
- Even higher scaling factors (e.g., 20–24 $\times$) incurred only minor latency overhead and may still be acceptable for less time-critical online applications.

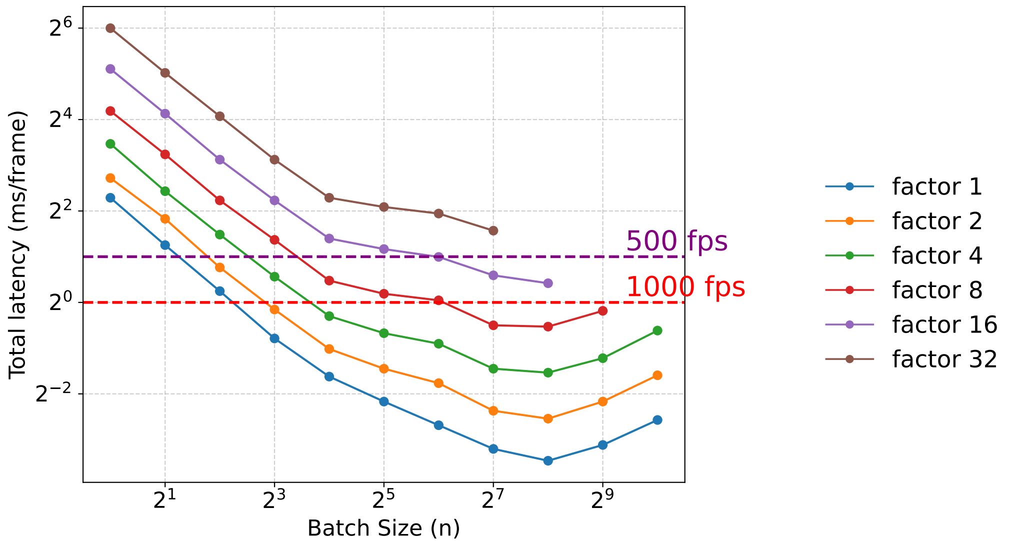


Figure 6.14: Online motion-correction latency on the H100 system across batch sizes and spatial scaling factors. Dashed lines indicate 1 ms and 2 ms latency thresholds for 1000 FPS and 500 FPS operation, respectively.

As shown in Figure 6.14, even though frame–by–frame motion-correction remains out of reach, the H100 significantly expands the operational envelope of the hybrid pipeline. With modest batching, it achieves real-time throughput at higher resolutions than previously possible. These results suggest that modern hardware and future hardware, can bring the system closer to true real-time feedback operation at scale.

6.5.2 Profiling-Driven Optimization Outcomes

The batch and kernel–level optimizations described in Section 4.2.5 produced substantial reductions in the per-frame processing time of the online pipeline. In particular, switching to single–precision FFTs, precomputing the template FFT, and fusing FFT $\rightarrow$multiply $\rightarrow$IFFT into a single XLA–compiled kernel all contributed to this speedup.

Figure 6.13 showed the latency vs. batch size trade-off before these changes. To highlight the impact of our optimizations, Figure 6.15 compares the total per-frame latency for the core online stages both before and after profiling-driven improvements on the CUBE server. Similarly, Figure 6.15 isolates the motion-correction stage to show its raw speedup.

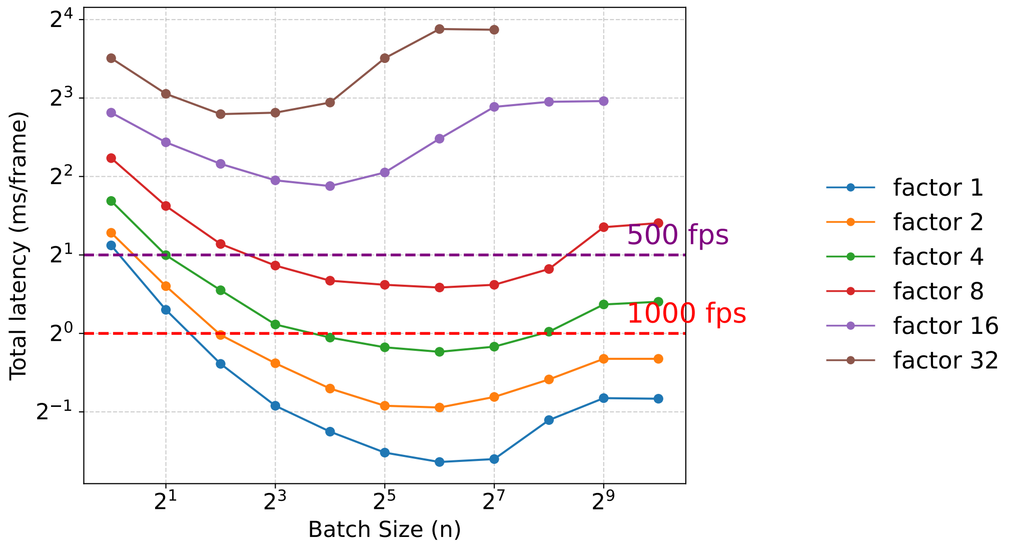


Figure 6.15: Per-frame latency for the online pipeline across batch sizes and scaling factors after applying motion-correction optimizations. Dashed horizontal lines represent 2 ms (500 FPS) and 1 ms (1000 FPS) thresholds.

As shown, the end–to–end per-frame latency (Figure 6.16) drops by over 50 % in the small-batch regime, bringing batch sizes of 2-256 within the 2 ms frame budget for most spatial factors. The core motion–correction kernel alone (Figure 6.15) now executes in under 2 ms for all resolutions when using a batch size bigger than 1.

With computational overhead largely addressed, the dominant remaining bottleneck is now the GPU–to–CPU memory transfer in strict frame-by-frame operation. This transfer

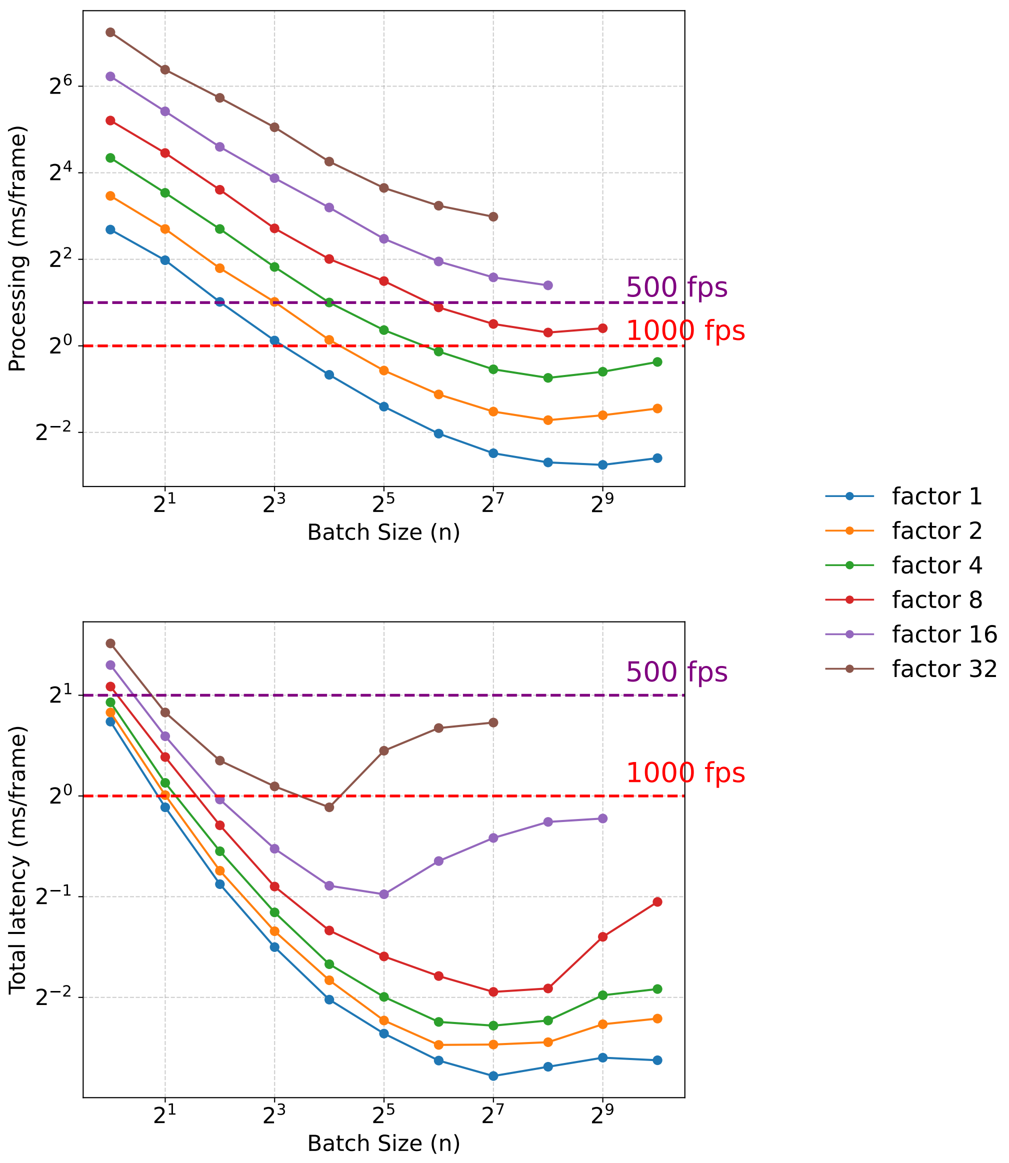


Figure 6.16: Comparison of per-frame processing latency across batch sizes and scaling factors (top), and post-optimization latency (bottom). Dashed horizontal lines in the top panel indicate the 2 ms (500 FPS) and 1 ms (1000 FPS) thresholds.

cost, quantified in Figure 6.5d, underscores the necessity of further optimizing data movement, via pinned memory, true asynchronous transfers, or deeper kernel fusion to push towards true 1000 FPS responsiveness.

6.6 Memory evaluation

Real-time voltage imaging pipelines must not only meet strict latency budgets but also operate within the memory limits of commonly available workstations. Excessive GPU or host memory usage can lead to out-of-memory errors or force swapping, both of which dramatically increase per-frame latency and undermine closed-loop experiments. To ensure our hybrid pipeline remains accessible to laboratories with modest hardware, such as a single 8 GB or 12 GB GPU a detailed memory evaluation across key online parameters was performed.

Figure 6.17 plots peak resident memory (in GB) as a function of batch size for several spatial scaling factors. Each curve corresponds to a different input resolution (see Table 6.1 for scaling definitions). As batch size increases, memory consumption grows approximately linearly on the GPU because larger tensors must be allocated on the GPU.

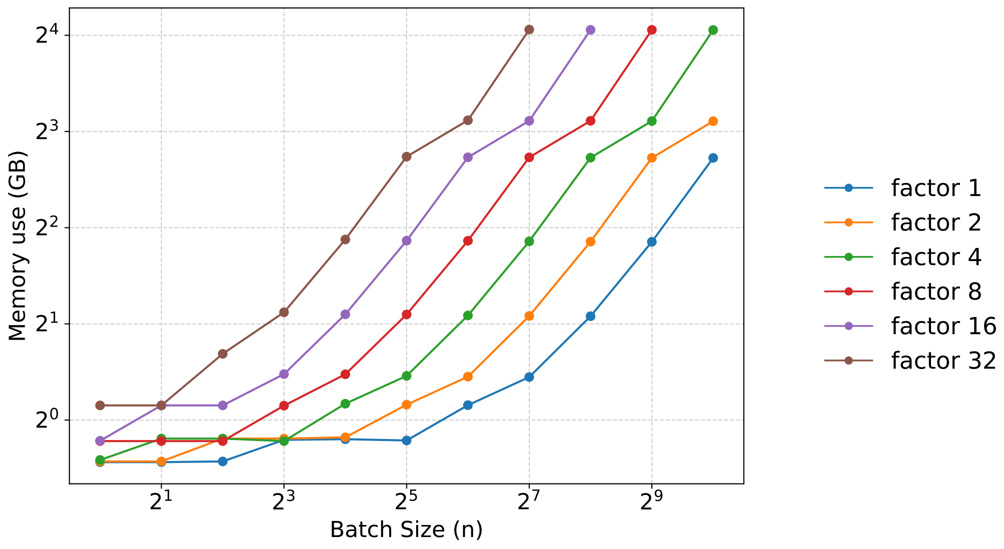


Figure 6.17: Peak memory usage versus batch size for different spatial scaling factors. Higher resolution and larger batch sizes both drive up GPU memory requirements.

There is an inevitable trade-off between latency, batch size, spatial resolution, and memory availability. Larger batches amortize kernel launch and transfer overhead, lowering per-frame latency, but simultaneously raise the memory bar (Figure 6.17). Likewise, increasing the spatial scaling factor inflates tensor sizes, further tightening memory constraints. These results underscore the importance of jointly evaluating compute and memory to identify Pareto-optimal operating points that satisfy both low-latency and resource-availability requirements.

Chapter 7

Conclusions

Voltage imaging offers a powerful window into fast neural dynamics, but existing pipelines rarely meet the dual demands of real-time performance and practical usability. Many prior systems have focused on optimizing individual components, such as motion-correction or segmentation, in isolation, often overlooking whole-system latency, initialization overhead, and deployment complexity. As a result, even high-performance methods have struggled to integrate seamlessly into closed-loop experimental workflows.

This thesis set out to address that gap. The central objective was to design and evaluate a hybrid voltage imaging pipeline that combines the real-time capabilities and motion-correction performance of FIOLA with the modularity and segmentation accuracy of VOLTAGE. A strict real-time budget was established, capping per-frame latency at 50 ms for spatial resolutions up to 300Œ300, to ensure closed-loop compatibility. The resulting system delivers real-time feedback, high-throughput performance, and end-to-end reproducibility, all within a flexible, GPU-accelerated architecture suitable for in-lab use.

Through careful architectural design, modular integration, and profiling-driven optimization, this work demonstrates that targeted system-level improvements can substantially enhance both the performance and usability of neural imaging pipelines. The final design is practical, accurate, and readily deployable, while leaving a clear and modular path forward for future improvements.

This final chapter has summarized the thesis contributions, quantified empirical performance across hardware configurations, and discussed the trade-offs and limitations encountered during development. Together, these results highlight the importance of full-system thinking in neuroscience software engineering and establish a foundation for continued advancement in real-time imaging pipelines.

7.1 Thesis Contributions

The work culminated in a hybrid voltage imaging pipeline that merges the GPU-based motion-correction and spike detection of FIOLA with the U-Net segmentation and modular trace extraction of VOLTAGE. The design emphasizes real-time capability, modularity, and reproducibility, bridging the gap between offline analysis and live experimental demands.

- **Pipeline Bottleneck Analysis:** A detailed evaluation of runtime performance in existing and hybrid pipelines, identifying motion-correction as the primary bottleneck for real-time spike extraction. (Chapter 4)

- **Hybrid Optimization Framework:** The development of a modular hybrid pipeline combining FIOLAs motion-correction and spike inference with VOLTAGEs segmentation and trace extraction, integrated with GPU-based enhancements such as XLA-compiled median filtering and TensorFlow data pipeline optimization. (Chapter 4.2.4)
- **Signal Quality Enhancements:** Accuracy-focused methods such as vignette filtering and wavelet-based denoising that improve spike detection consistency and segmentation precision, especially in noisy or low-contrast recordings. (Chapter 4.1.6)
- **Scalable Real-Time System:** An online-capable pipeline architecture that supports real-time spike inference at 500-1000 FPS, with tunable batching for low-latency feedback. (Chapter 6.5.2)
- **Live Deployment Capability:** A Dockerized, streaming-compatible execution model with shared-memory input and camera simulation, enabling seamless integration into live experimental setups without the need for file-based preprocessing. (Chapter 4)
- **Open Benchmark Toolkit:** A reproducible benchmarking suite supporting calibration, evaluation, and profiling across different datasets, resolutions, and GPU configurations. (Chapter 5.4.3)
- **Empirical Validation:** Comprehensive results validating throughput, latency, and spike detection accuracy using both synthetic and real data, including application to closed-loop contexts. (Chapter 6)

7.2 Performance Achievements

The primary evaluation system was a CUBE server equipped with an NVIDIA Tesla V100 GPU. On this hardware, the optimized pipeline reached real-time throughput of 500 FPS for spatial resolutions up to 680×680, albeit with 8 ms latency.

Additionally, startup time was significantly reduced through preloading optimizations and parallelized initialization routines. In contrast to the original FIOLA and VOLTAGE pipelines, both of which required extensive manual setup and environment management, the final system can be deployed from a single Docker container with minimal configuration. This usability improvement lowers the barrier to adoption in real-world lab environments and is on itself a major contribution.

7.3 Discussion

While performance and accuracy were improved, several trade-offs and observations emerged:

- **Unified workflow and reproducibility:** A key advantage of this work is its streamlined, modular architecture. It supports offline and online modes using the same codebase and enables partial execution with cached data, a feature particularly helpful for testing and iterative development.

- **Latency vs. frame rate trade-offs:** To achieve higher frame rates, moderate batching is required. However, batching inherently introduces latency. The system's configurability makes it possible to tune this trade-off per experiment, but low-latency, high-resolution operation remains challenging.
- **Latency vs. Memory Footprint:** Memory profiling (Section 6.6) shows that increasing batch size or spatial resolution improves GPU throughput but rapidly inflates VRAM usage and device-host transfer overhead. Since spike detection must complete within a 50 ms window, we must balance batch-driven latency gains against the risk of out-of-memory errors and longer transfer times, especially on GPUs with limited memory.
- **Ease of use vs. internal complexity:** The Dockerized deployment and camera simulation make the system easy to install and run. However, maintaining modularity (e.g., separating shading and motion-correction) comes at the cost of some performance overhead, most evident in the shading correction runtime.

7.4 Limitations and Future Work

While the proposed hybrid pipeline achieves substantial improvements in real-time spike inference and modularity, several limitations remain and offer directions for future development:

- **Lack of live hardware testing:** Although the system was evaluated extensively using synthetic camera streams and memory-mapped simulation, it has not yet been validated with real acquisition hardware in a live experimental setup.
- **Frame-by-frame streaming limitations:** True frame-by-frame (batch size = 1) execution in the online pipeline was not achievable due to persistent GPU-to-CPU memory transfer overhead. Addressing this would require one or more of the following:
  - Direct streaming of camera video to GPU memory,
  - Lower-level memory pinning and data transfer optimizations,
  - GPU-based trace extraction to reduce host-device communication entirely, potentially converting this into a sparse GPU computation pipeline.
- **Shading correction overhead:** The current shading correction step is implemented using a combination of C++, CUDA, and Cython, which limits maintainability and performance tuning. A reimplementation using fully GPU-native operations (e.g., CuPy or TensorFlow) could eliminate this bottleneck.
- **Limited optimization of offline components:** While motion-correction and median filtering were heavily optimized, other offline stages, such as segmentation and trace extraction, did not receive comparable acceleration. If significantly higher spatial resolutions become standard in future experiments, these components may require additional tuning or GPU acceleration.

- **No automatic configuration tuning:** Runtime parameters such as batch size, thread count, and chunk size are currently set manually. Future versions could incorporate automated calibration based on real-time hardware profiling to dynamically select optimal settings.
- **Lack of multi-GPU or distributed support:** The pipeline is designed for single-device execution. Extending it for multi-GPU or cloud-deployed use cases could allow much higher throughput in parallel imaging scenarios.

These limitations do not undermine the systems current utility but highlight concrete opportunities for future refinement as deployment contexts and experimental demands evolve.

Bibliography

[1] Yosuke Bando, Ramdas Pillai, Atsushi Kajita, Farhan Abdul Hakeem, Yves Quemener, Hua-an Tseng, Kiryl Piatkevich, Changyang Linghu, Xue Han, and Edward S. Boyden. Real-time Neuron Segmentation for Voltage Imaging [dataset], oct 2023.

[2] Antic Lab, University of Connecticut Health. Sequential dendritic voltage imaging and dendritic calcium imaging from the same region of interest on a basal dendrite of a layer 5 cortical pyramidal neuron. Antic Lab Image Gallery, n.d. Recolored version.

[3] Darcy S. Peterka, Hiroto Takahashi, and Rafael Yuste. Imaging voltage in neurons. *Neuron*, 69(1):9–21, 2011.

[4] A. Giovannucci, J. Friedrich, P. Gunn, J. Kalfon, B. L. Brown, S. A. Koay, J. Taxidis, F. Najafi, J. L. Gauthier, P. Zhou, B. S. Khakh, D. W. Tank, D. B. Chklovskii, E. A. Pnevmatikakis, and L. Paninski. Volpy: Automated and scalable analysis pipelines for voltage imaging. *bioRxiv*, 2021.

[5] M. Pachitariu, C. Stringer, M. Dipoppa, S. Schröder, L. F. Rossi, H. W. P. Dalgleish, M. Carandini, and K. D. Harris. Suite2p: beyond 10,000 neurons with standard two-photon microscopy. *bioRxiv*, 2017.

[6] Y. Bando, M. Chen, N. J. Hill, R. Zhou, H. Kawsar, G. Hobbs, M. Glenn, A. D. Douglass, D. B. Chklovskii, and E. A. Pnevmatikakis. Real-time neuron segmentation and spike extraction from voltage imaging data. *arXiv preprint*, arXiv:2403.16438, 2024.

[7] Logan Grosenick, James H. Marshel, and Karl Deisseroth. Closed-loop and activity-guided optogenetic control. *Neuron*, 86(1):106–139, 2015.

[8] M. B. Bouchard, V. Voleti, C. S. Mendes, C. Lacefield, W. B. Grueber, R. S. Mann, R. M. Bruno, and E. M. C. Hillman. Swept confocally-aligned planar excitation (scape) microscopy for high-speed volumetric imaging of behaving organisms. *Nature Photonics*, 9:113–119, 2015.

[9] A. S. Abdelfattah, T. Kawashima, A. Singh, O. Novak, H. Liu, H. Shuai, A. Y. Hu, P. Borden, H. A. Day, R. O. Rasmusson, A. D. Snyder, D. J. Blake, A. P. Chambers, R. Y. Tsien, and M. Inoue. Bright and photostable chemigenetic indicators for extended in vivo voltage imaging. *Science*, 365(6454):699–704, 2019.

[10] Thomas Knöpfel and Chenchen Song. Optical voltage imaging in neurons: moving from technology development to practical tool. *Nature Reviews Neuroscience*, 20(12):719–727, 2019.

[11] R. U. Kulkarni and E. W. Miller. Voltage imaging: Pitfalls and potential. *Biochemistry*, 56(39):5171–5177, 2017.

[12] Wikipedia contributors. Refractory period (physiology). `https://en.wikipedia.org/w/index.php?title=Refractory_period_(physiology)&oldid=1257739036`, Nov 2024. Accessed: 2025-06-07. Text available under CC BY-SA 4.0.

[13] G. J. Gage, C. R. Stoetzner, A. B. Wiltschko, F. H. Gage, E. O. Boateng, S. Ribeiro, J. N. Meta, K. Zebrowski, and G. G. Koch. The real-time experimental interface: A tool for studying closed-loop neuroscience. *Frontiers in Neural Circuits*, 9:44, 2015.

[14] Adrien Jouary and Germán Sumbre. Real-time imaging and optogenetics in neuroscience. *Current Opinion in Neurobiology*, 79:102708, 2023.

[15] John L. Gustafson. Amdahls law. In David Padua, editor, *Encyclopedia of Parallel Computing*, pages 53–60. Springer, Boston, MA, 2011.

[16] Rui Silva. Personal communication, 2025. PhD candidate at Erasmus MC, advisor to the author.

[17] Changjia Cai, Johannes Friedrich, Abhinav Singh, Mohammad H Eybposh, Eftychios A Pnevmatikakis, Kaspar Podgorski, and Andrea Giovannucci. Fiola: An accelerated pipeline for fluorescence imaging online analysis. *bioRxiv*, 2021.

[18] Eftychios A. Pnevmatikakis and Andrea Giovannucci. Normcorre: An online algorithm for piecewise rigid motion correction of calcium imaging data. *Journal of Neuroscience Methods*, 291:83–94, 2017.

[19] Philippe Thévenaz. Turboreg and other imagej plugins at the biomedical imaging group of epfl. In *Proceedings of the Euro-BioImaging First Workshop on Bioimage Analysis Software: Is There a Future Beyond ImageJ?*, page 23, Barcelona, Spain, 2012.

[20] Jean-Yves Tinevez, Nick Perry, Johannes Schindelin, Genevieve M. Hoopes, Gregory D. Reynolds, Emmanuel Laplantine, Sebastian Y. Bednarek, Spencer L. Shorte, and Kevin W. Eliceiri. Trackmate: An open and extensible platform for single-particle tracking. *Methods*, 115:80–90, 2017.

[21] Ljubisa Platisa and et al. Roi-accelerated motion correction in voltage imaging, 2022. Closed-source method described in unpublished technical documentation.

[22] George Adam. Manual annotation in neural imaging studies, 2019. Unpublished dataset annotation protocol.

[23] Patrick Kaifosh, Michael Lovett-Barron, Georg Turi, Thomas R. Reardon, and Attila Losonczy. Sima: Python software for analysis of dynamic fluorescence imaging data. *Frontiers in Neuroinformatics*, 8:80, 2014.

[24] R. Kannan and collaborators. Signal-informed segmentation using gevi polarity features, 2022. Technical report.

[25] Y. Liu and collaborators. Template matching for segmentation in gevi recordings, 2022. Unpublished method description.

[26] L. Wang and collaborators. Real-time neural network-based segmentation for voltage imaging, 2023. Manuscript in preparation.

[27] H. Brooks and collaborators. Probabilistic refinement for segmentation in noisy voltage data, 2024. Preprint in submission.

[28] M. Weber and collaborators. Integrated volumetric segmentation pipelines for neural imaging, 2023. Technical documentation.

[29] J. Tian and collaborators. Segmentation within high-speed volumetric imaging systems, 2022. Conference poster.

[30] Xin Fan and collaborators. Exponential fitting for photobleaching correction in calcium imaging, 2020. Unpublished protocol or internal documentation.

[31] David Pedrosa and collaborators. High-pass filter based baseline correction in neural fluorescence recordings, 2024. Manuscript in review.

[32] Emily Jackson and collaborators. Baseline stabilization using rolling averages in voltage imaging, 2024. Conference presentation.

[33] Chang Liu, Jing Lu, Yiyang Wu, Xin Ye, Allison M. Ahrens, Jelena Platisa, Vincent A. Pieribone, Jerry L. Chen, and Lei Tian. Deepvid v2: Self-supervised denoising with decoupled spatiotemporal enhancement for lowphoton voltage imaging. *Neurophotonics*, 11(4):045007, 2024. E-published Oct 29, 2024.

[34] V. Villette, M. Chavarha, I. Dimov, J. Bradley, N. Pradhan, C. E. Pizoli, A. Tanguay, I. Khan, S. Chen, D. Font-Rodriguez, Y. Kim, A. G. Richardson, C.-F. Latchoumane, E. A. Pnevmatikakis, M. J. Schnitzer, F. Gao, W. Grueber, K. Holthoff, C. A. Barnett, J. T. Vogelstein, H. Shimazaki, Z. Knox, O. Parnas, J. Echegoyen, K. Deisseroth, and G. Feng. Ultrafast two-photon imaging of a high-gain voltage indicator in awake behaving mice. *Cell*, 179(7):1590–1608.e23, 2019.

[35] A. Shu and collaborators. Low-complexity synchrony-based spike detectors for real-time imaging, 2021. Technical report.

[36] R. Zhang and collaborators. A hybrid thresholding strategy for robust spike detection in voltage imaging, 2023. Preprint under review.

[37] J. Sabater and collaborators. Global-local hybrid thresholding in neural signal extraction, 2021. Presented at IEEE EMBC 2021.

[38] D. Kim and collaborators. Compressed neural imaging with spatially adaptive thresholding, 2023. Manuscript in preparation.

[39] Rodrigo Quian Quiroga, Zoltan Nadasdy, and Yoram Ben-Shaul. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural computation*, 16(8):1661–1687, 2004.

[40] Miroslaw Latka, Zbigniew Was, Andrzej Kozik, and Bruce J West. Wavelet analysis of epileptic spikes. *Physical Review E*, 67(5):052902, 2003.

[41] Google. Better performance with the `tf.data` api. `https://www.tensorflow.org/guide/data_performance`, 2023. Accessed: 2025-06-08.

[42] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, 2017.

[43] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015.

Glossary

This appendix provides an overview of frequently used terms and abbreviations.

Graph: In GPU computing, a "graph" often refers to a computational graph or dataflow graph, which represents a sequence of operations and their data dependencies. This structure allows the GPU to efficiently parallelize and execute these operations by identifying independent tasks and optimizing memory access patterns.objects.

TensorFlow: An open-source machine learning framework developed by Google. TensorFlow allows users to build and train models using computational graphs, where operations are represented as nodes and data (tensors) flow along edges. It supports both CPU and GPU execution and is widely used for deep learning, signal processing, and other numerical tasks.

Appendix A

GPU benchmark comparison

This appendix presents benchmarking results for key GPU-accelerated operations used in the hybrid voltage imaging pipeline. The goal of these benchmarks is to compare performance across two representative hardware setups: an NVIDIA RTX 3080 (consumer-grade GPU) and an NVIDIA Tesla V100 (data center-class GPU). The focus is on critical operations such as DFT-based shift correction, FFT execution, cross-correlation, and image normalization. These comparisons guided decisions around kernel configurations, batching strategies, and hardware deployment scenarios during pipeline development.

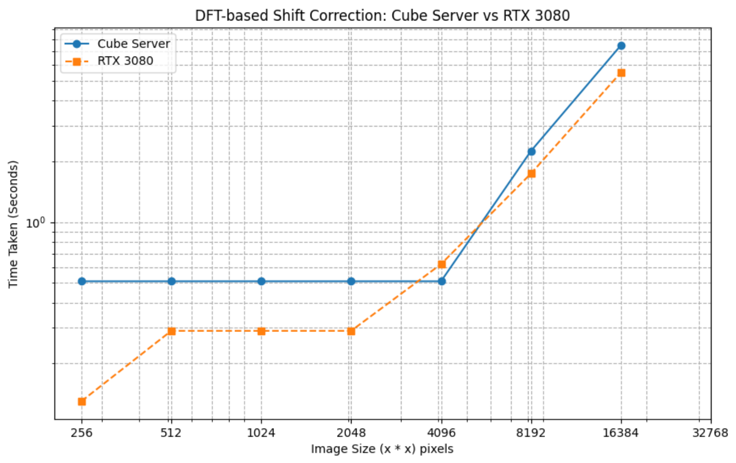


Figure A.1: Benchmark results for FIOLA's dft-based shift correction

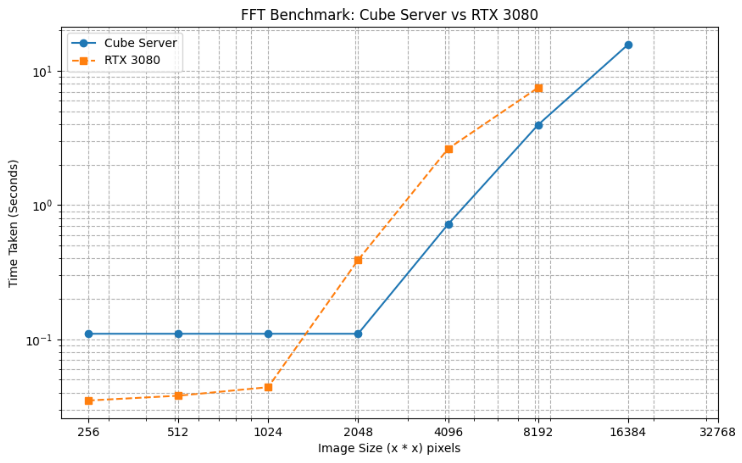


Figure A.2: Benchmark results for general FFT sizes

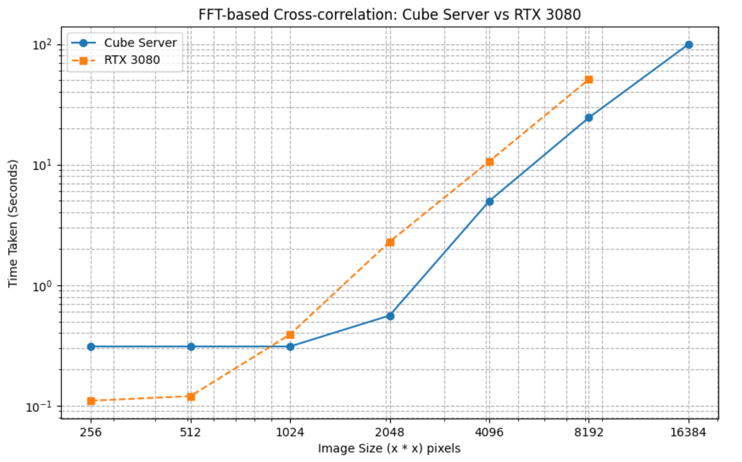


Figure A.3: Benchmark results for FFT cross-correlation

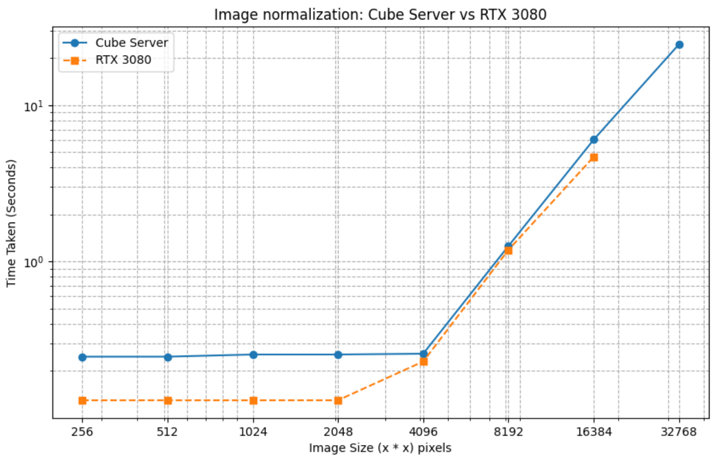


Figure A.4: Benchmark results for image normalization

Appendix B

In-depth Analysis of FIOLA and VOLTAGE

B.1 FIOLA: Fluorescence Imaging OnLine Analysis

FIOLA is an online voltage imaging pipeline designed for low-latency, real-time processing. After an initial offline calibration, it processes frames independently using a GPU-resident computational graph, achieving millisecond-scale latencies. FIOLA supports frame-by-frame motion-correction, fluorescence trace extraction, and spike inference without full video buffering, making it suitable for closed-loop experimental paradigms.

The structure and execution flow of FIOLA's offline phase is illustrated in Figure 2.4, which highlights each step leading up to online operation and the data artifacts generated at each stage.

While FIOLA is described as an online pipeline, it does not operate in a fully streaming mode. Specifically, after offline calibration, the full imaging file is first loaded into GPU memory. Only then does FIOLA process each frame independently or in small batches without further buffering delays. This distinction is important in the context of real-time systems that must operate continuously on incoming data. Figure 2.5 illustrates the sequence of operations following offline initialization, capturing the data flow and GPU execution model during the online phase.

B.1.1 Motion-Correction

FIOLA performs rigid motion-correction by aligning each incoming frame to a fixed reference template (typically obtained during initialization) using frequency-domain normalized cross-correlation (ZNCC). Crucially, once the required subpixel shift is determined, it is applied directly in the Fourier domain via phase modulation, avoiding interpolation in the spatial domain.

The algorithm proceeds as follows:

1. **Fourier Cross-Correlation:** The incoming frame I_t and the template T are first zero-mean normalized and transformed via a 3D FFT, although the complex values remain at 0. Their normalized cross-correlation is computed in the frequency domain using:

$$\text{ZNCC} = \left|\mathcal{F}^{-1}\left(\mathcal{F}(I_t) \cdot \mathcal{F}(T)^*\right)\right|$$

   where $\mathcal{F}$ denotes the FFT and * is the complex conjugate.

2. **Subpixel Shift Estimation:** The peak of the ZNCC map is extracted, and Gaussian interpolation is applied in log-space to estimate subpixel displacements $(\Delta x, \Delta y)$.

3. **Phase-Based Shift Application:** The shift is then applied in the frequency domain by modulating the phase of the FFT of I_t. This approach exploits the Fourier shift theorem and avoids explicit interpolation:

$$I'_t = \mathcal{F}^{-1}\left(\mathcal{F}(I_t) \cdot e^{-2\pi i(u\Delta x + v\Delta y)}\right)$$

   where (u, v) are spatial frequency coordinates.

Figure 2.6 illustrates this pipeline, including the FFT-based cross-correlation, subpixel estimation, and frequency-domain shift correction.

Template Construction: The accuracy of the motion-correction step depends on the quality of the reference template. FIOLA constructs this during an initialization phase by computing the pixelwise temporal median over a batch of frames (usually 5000 or more). This median operation is performed on the CPU and has a complexity of $O(NL\log L)$ for N pixels and L frames.

Computational Complexity: The per-frame cost is dominated by FFT operations and subpixel refinement:

- FFT and inverse FFT: $O(N\log N)$
- Phase-based shift: $O(N)$
- Subpixel peak estimation: $O(1)$ per frame

All steps are efficiently parallelized on GPU using TensorFlow's 'fft3d' and 'ifft3d' operations.

Implementation Details: Motion-correction is implemented as a custom TensorFlow layer using 'tf.signal.fft3d' and 'ifft3d'. The cross-correlation, phase shifting, and inverse FFT are executed on the GPU for high throughput. Precomputed frequency-domain template values are reused across frames, and the system maintains a batched pipeline for real-time operation. Shift values are computed per frame and passed downstream for trace alignment. The FFT-based correction supports subpixel accuracy without the artifacts that typically arise from spatial-domain interpolation.

B.1.2 Trace Extraction

FIOLA does not perform neuron segmentation online. Instead, it relies on pre-generated ROI masks, which must be computed during an offline initialization phase. These masks define the spatial footprints of individual neurons and can be obtained using methods such as manual annotation, external segmentation models (e.g., Mask R-CNN [42]), or low-rank matrix factorization applied to a batch of frames. Once these masks are defined, FIOLA uses them for fast online trace extraction.

The motion-corrected video is represented as:

$$Y \in \mathbb{R}^{d\times T}, \tag{B.1}$$

where d is the number of pixels and T the number of frames. This is approximated as:

$$Y \approx AC + B, \tag{B.2}$$

where:

- $A \in \mathbb{R}^{d\times K}$ contains spatial neuron footprints (one column per ROI),
- $C \in \mathbb{R}^{K\times T}$ holds the corresponding temporal activity traces,
- B models background fluorescence.

Offline Initialization: FIOLA initializes A, C, and B from a batch of frames ($T_{\text{init}} \sim 1000$) using projected gradient descent (PGD) on the following objective:

$$\min_{A,C,B} \|Y_{\text{init}} - AC - B\|_F^2 \quad \text{s.t.} \quad A \geq 0,\ C \geq 0. \tag{B.3}$$

This produces spatial masks in A, which serve as neuron segmentation for later trace extraction.

Online Trace Extraction: Once initialized, FIOLA fixes A and B. For each incoming frame y_t, the temporal activity vector c_t is computed via:

$$c_t = \arg\min_{c\geq 0} \| y_t - Ac - b_t\|^2, \tag{B.4}$$

where b_t is the background at time t. In regions without ROI overlap, this reduces to:

$$c_k[t] = \sum_{i\in R_k} \big(y_t(i) - b_t(i)\big), \tag{B.5}$$

where R_k is the support of ROI k. In overlapping regions, small nonnegative least squares (NNLS) problems are solved per cluster (e.g., 2×2), which introduces negligible overhead.

Computational Complexity:

- **Offline Initialization:** $O(N_{\text{iter}}\, d\, K\, T_{\text{init}})$, amortized over time. GPU acceleration and sparse operations make this feasible.
- **Online Trace Extraction:** $O(d + K)$ per frame with optional local NNLS for overlapping ROIs.

Implementation Details:

- A, B, and incoming frames are kept on GPU.
- PGD and online updates use GPU kernels (cuBLAS, cuDNN, or custom CUDA).
- Minimal memory transfer overhead; only spike traces c_t are stored.

Summary: FIOLA's online trace extraction is fast and accurate but depends on external segmentation. The framework assumes fixed masks and does not dynamically discover or update ROIs. This constraint simplifies online inference but limits adaptability to shifting neural activity patterns or new spatial regions during acquisition.

B.1.3 Spike Detection and Photobleaching Correction

FIOLA identifies spike events in real time by applying matched filtering and adaptive thresholding to each neuron's trace. To maintain accuracy throughout long recordings, photobleaching correction is integrated directly into the filtering steps.

Photobleaching Correction (DC Blocker & Running Median): To compensate for baseline decay and low-frequency drift, FIOLA applies a two-stage high-pass filtering sequence:

1. **DC-Blocking Recursive Filter:** For each fluorescence trace $F[n]$, apply:

   $$X[n] = F[n] - F[n-1] + R \cdot X[n-1], \tag{B.6}$$

   where $R \approx 0.995$ defines a cutoff at approximately 0.5 seconds (200 frames at 400 Hz). This suppresses slow transients while retaining spike-scale dynamics.

2. **Running Median Filter:** The signal $X[n]$ is then centered using a running median of window size w, typically 15 frames:

   $$\hat{F}[n] = \text{median}(X[n-w:n]). \tag{B.7}$$

   This removes residual drift and subthreshold fluctuations. Due to the sparsity of spikes, the median is not affected by sharp transients.

3. **Adaptive Recalibration:** Every M frames:

   - Recompute a long-term baseline using the median of the past M samples.
   - Update the detection threshold using the 95^{th} percentile of the last 100 detected spike amplitudes.

Template Matching: Once filtered, each trace $X[n]$ is convolved with a predefined spike template $h[n]$ of length L (usually 5–10 frames):

$$S[n] = (X * h)[n] = \sum_{\tau=0}^{L-1} X[n+\tau] \cdot h[\tau]. \tag{B.8}$$

This matched filter amplifies true spike events by integrating over time. The template is obtained during initialization by averaging clean spike waveforms. Spike template extraction during initialization is performed explicitly in parallel using multiple CPU cores.

Adaptive Thresholding and Peak Selection: Spikes are identified as local maxima in $S[n]$ exceeding a dynamically updated threshold:

- **Adaptive Threshold:** Every L frames, the threshold is reset to 50% of the 95th percentile of recent spike amplitudes.
- **Refractory Enforcement:** A refractory period of L frames prevents overcounting. Once a spike is detected at n_0, further detections are suppressed until $n > n_0 + L$.

Post-Processing:

- Merge adjacent detections within L frames into a single event.
- Assign the spike time to the frame with the highest value of $S[n]$ in the detection window.

Optional Subthreshold Trace Extraction: If desired, spikes can be removed from $X[n]$ to reveal underlying subthreshold membrane fluctuations. This is done by masking spike regions and applying a smoothing filter to the residual trace.

Computational Complexity:

- DC-blocking and median filtering: $O(1)$ per frame per trace.
- Matched filtering: $O(L)$ per frame per neuron.
- Thresholding and post-processing: $O(1)$ per frame per neuron.
- **Total:** $O(K \cdot L)$ per frame, efficiently parallelizable.

Implementation Details:

- All filtering and template matching are implemented using sliding windows or simple vector operations and run efficiently on the CPU.
- Initialization routines, including spike template extraction and percentile threshold estimation, are parallelized across multiple CPU cores.
- Threshold updates and signal filtering are integrated into the same frame-processing loop, minimizing latency.

Summary: FIOLA's spike detection pipeline achieves robust performance through a combination of baseline correction, template matching, and dynamic thresholding. Its modular, online-friendly architecture allows for consistent, high-speed operation suitable for real-time feedback systems.

B.2 VOLTAGE: A Fast, Modular Voltage Imaging Pipeline

VOLTAGE (Voltage imaging pipeline) is an open-source framework developed by the MIT Media Lab and collaborators for high-speed analysis of voltage imaging data [6]. It is designed to operate in real time or faster-than-recording-time on a single high-end workstation equipped with GPUs.

The pipeline consists of six main stages:

1. **Motion-Correction:** Stabilizes video data by removing global 2D translational shifts.
2. **Shading Correction:** Compensates for lighting intensity changes due to movement and illumination variability.
3. **Preprocessing:** Aggregates raw frames into summary images to suppress noise.
4. **Segmentation:** Applies a CNN to summary images and demixes overlapping ROIs via local NMF.
5. **Trace Extraction:** Computes fluorescence traces for each ROI over time.
6. **Spike Detection:** Identifies spikes on the extracted traces using thresholding or template matching.

Motion-correction, shading-correction and segmentation are designed for GPU acceleration and multi-GPU parallelism. VOLTAGE is implemented in C++/CUDA and Python and can be extended or replaced at any stage.

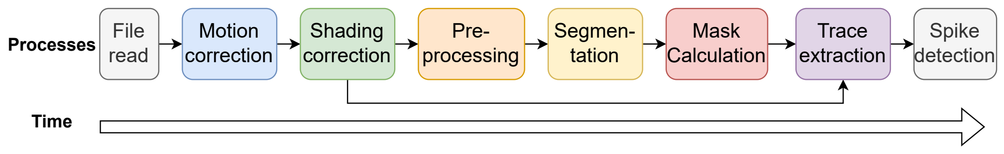


Figure B.1: Timeline of VOLTAGE's pipeline execution. Each stage uses the output of the previous one as its input, creating a sequential dependency. The only exception is trace extraction, which depends not only on the ROI mask but also directly on the output of shading correction.

B.2.1 Motion-Correction

The first stage of the pipeline performs rigid motion-correction using patch-based Zero-mean Normalized Cross-Correlation (ZNCC). For each incoming frame I_t, the goal is to estimate the optimal shift $(\Delta x, \Delta y)$ relative to a fixed reference template T. ZNCC is computed for candidate shifts across small patches to determine this displacement.

$$\text{ZNCC}_{(\Delta x,\Delta y)} = \frac{1}{|P|}\sum_{i\in P}\frac{(I_t(i+\Delta x,\Delta y)-\mu_I)(T(i)-\mu_T)}{\sigma_I\,\sigma_T},$$

where P denotes a patch of pixels (e.g., 21×21), and μ, σ are the patch-wise means and standard deviations of the current frame and template, respectively.

Algorithm Steps:

1. Divide each frame into $m \times n$ patches.
2. For each patch, evaluate ZNCC over a grid of M shift candidates (e.g., ±5 pixels in each direction).
3. Aggregate patch ZNCC scores across the frame.
4. Select the shift with the maximum aggregated ZNCC.
5. Translate the frame using the negative of the estimated shift (e.g., via bilinear interpolation).

B.2.2 Shading Correction

To account for illumination variability, such as vignetting or motion-induced shading changes, VOLTAGE integrates normalization into the motion-correction phase. Because ZNCC inherently removes differences in mean and contrast, it provides illumination-invariant alignment:

- Patches are zero-meaned and normalized by their standard deviation.
- No additional shading correction step is necessary.

Implementation Details:

- **CUDA Parallelism:** Patchwise ZNCC computations and shift evaluations are implemented via custom CUDA kernels or tensor libraries, allowing all patches and candidate shifts to be processed in parallel.
- **Integral Images:** Patch means and variances are computed using summed-area tables (integral images), which allow $O(1)$ lookup.
- **Memory Optimization:** Double-buffered GPU memory and pinned CPU memory ensure uninterrupted frame acquisition from NVMe storage.

- **Parallel GPU Strategy:** One GPU is allocated to motion and shading correction; others may handle segmentation and trace extraction.

Computational Complexity: Let:

- F be the number of frames,
- N be the number of pixels per frame,
- P be the number of patches per frame,
- M be the number of candidate shifts evaluated per patch.

The total complexity over all frames is:

$$O(F \cdot P \cdot M)$$

Each ZNCC evaluation within a patch is $O(s^2)$, where $s \times s$ is the patch size. Thus, the full complexity per frame becomes:

$$O(P \cdot M \cdot s^2) = O(N \cdot M)$$

assuming patch coverage over the full image. For example, a 512×512 image with 21×21 patches, sliding every 16 pixels, results in roughly 1024 patches, with $M \sim 441$ candidate shifts. This totals over 450,000 comparisons per frame.

B.2.3 Preprocessing

In VOLTAGE, preprocessing is performed without explicit denoising. Instead, the pipeline constructs two types of summary images over short temporal windows of L motion- and shading-corrected frames. These summaries reduce noise, enhance salient structures, and serve as input to the segmentation model.

Summary Types: Two complementary summary statistics are computed per L-frame segment:

- **Temporal Average Image:**

$$A(x) = \frac{1}{L}\sum_{t=1}^{L} V(x,t),$$

where $V(x,t)$ denotes the motion-corrected intensity at pixel x and time t. This smooths out uncorrelated noise and highlights persistent spatial features (e.g., cell bodies).

- **Max-minus-Median Image:**

$$M(x) = \max_{1\le t\le L} V(x,t) - \text{median}_{1\le t\le L}\big(\tilde{V}(x,t)\big),$$

where $\tilde{V}(x,t)$ is the result of Gaussian blurring each frame with standard deviation $\sigma = 3$ pixels. This highlights transient events, such as spike-evoked fluorescence, while suppressing slow drift and local noise.

Noise Suppression: These summary strategies are designed to attenuate noise without sacrificing spike visibility:

- *Averaging* attenuates random noise and accentuates stable neuron outlines.
- *Gaussian blurring* in the max-minus-median suppresses high-frequency pixel-level noise that might otherwise corrupt the median baseline.
- The *max* operation preserves transient peaks, making this summary ideal for identifying neurons with sparse firing.

Implementation Notes:

- Raw frames are streamed and buffered in GPU memory for each L-frame segment.
- Gaussian blurring is implemented via separable convolutions using either CUDA or cuDNN-accelerated PyTorch ops.
- The median is computed per pixel over time, with exact methods costing $O(L)$ per pixel. Approximate medians (e.g., bin-based quantile estimation) can be used for large L to reduce runtime.
- Only the two summary images $A(x)$ and $M(x)$ are stored, greatly reducing memory overhead compared to keeping raw frame buffers.

Computational Complexity: Let d be the number of pixels per frame. The computational cost of summary image extraction is:

$$O(d \cdot L),$$

as each pixel requires L operations for mean, max, and median across frames.

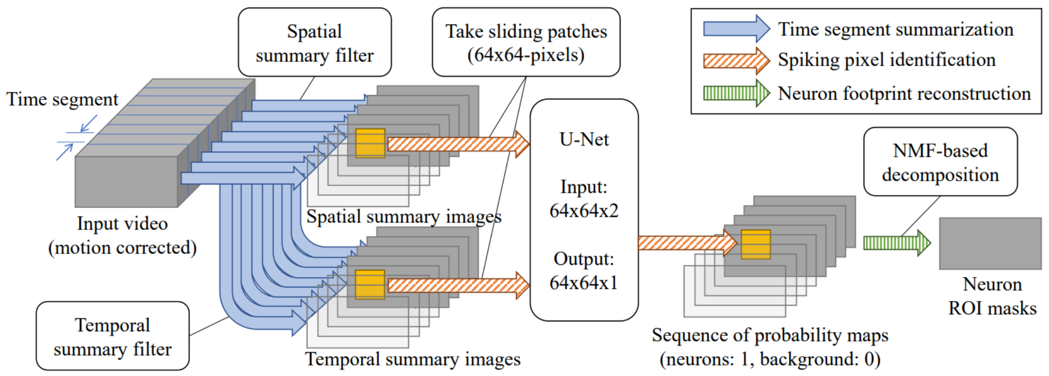

Spatial summary filter
Take sliding patches (64x64-pixels)
Time segment summarization
Spiking pixel identification
Neuron footprint reconstruction
Time segment
Input video (motion corrected)
Spatial summary images
U-Net
Input: 64x64x2
Output: 64x64x1
NMF-based decomposition
Neuron ROI masks
Temporal summary filter
Temporal summary images
Sequence of probability maps (neurons: 1, background: 0)

Figure B.2: Illustration of the two summary images used for neuron segmentation in VOLTAGE. The average image highlights stable spatial structures, while the max-minus-median emphasizes sparse, transient events such as spikes. Image from [6]

These summary images are then passed to the U-Net-based segmentation model, where they act as spatial priors for neuron identification.

B.2.4 Segmentation

Following preprocessing, VOLTAGE applies a convolutional neural network based on the U-Net architecture to the summary images. This model identifies the spatial footprints of individual neurons by learning to map image features to binary masks indicating regions of interest (ROIs).

Model Input and Output: The network receives two-channel input:

- Channel 1: the temporal average image.
- Channel 2: the max-minus-median summary image.

The output is a single-channel probability map where each pixel denotes the likelihood of being part of a neuron. This is subsequently thresholded and post-processed to obtain discrete, labeled ROIs.

Architecture Overview: The U-Net consists of an encoder-decoder structure with skip connections:

- The encoder path progressively downsamples the input through convolutional and max-pooling layers, capturing hierarchical spatial features.
- The decoder path upsamples feature maps using transposed convolutions and merges them with corresponding encoder features via skip connections. This recovers spatial resolution while retaining learned context.
- A final sigmoid activation produces a dense probability map suitable for segmentation.

This structure is particularly well-suited for biomedical image segmentation tasks, where both coarse context and fine detail are needed to distinguish overlapping or faint features.

Training and Inference: The model is trained offline using manually annotated masks and data augmentation (e.g., rotation, scaling). During inference, it runs on the GPU and operates on summary images generated in the preprocessing stage. Predictions are made segment-wise and are immediately available for ROI extraction.

Implementation Notes:

- All computations are GPU-accelerated using PyTorch or TensorFlow, enabling efficient inference even for moderate-sized images.
- Summary images are normalized before being passed to the network.

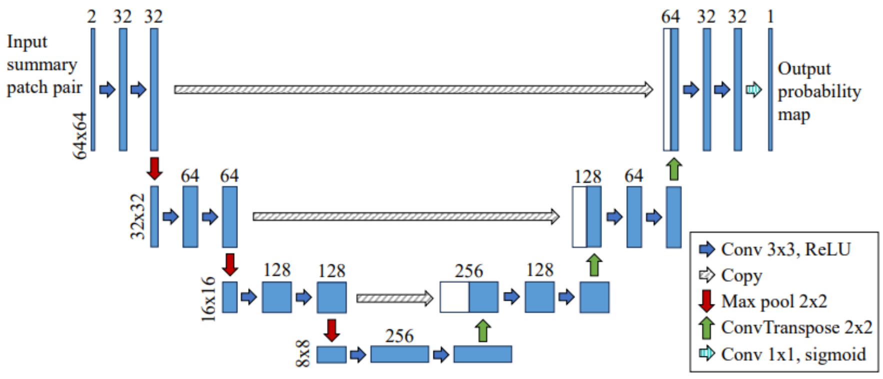


Figure B.3: Structure of the U-Net model used for segmentation in the VOLTAGE pipeline. The two summary images serve as input channels, and the output is a pixelwise probability map indicating neuron locations. Image from [6]

- Output probability maps are thresholded and filtered (e.g., by area or shape) to remove false positives.

Once segmentation masks are obtained, they are used in downstream processing to extract fluorescence traces in combination with the motion and shading corrected frames.

B.2.5 Trace Extraction

Once final ROI masks R_k are obtained, compute each neuron's fluorescence trace in the motion-corrected video:

$$F_k[t] = \frac{1}{|R_k|} \sum_{i \in R_k} I_t(i),$$

where $I_t(i)$ is the intensity of pixel i at frame t.

Implementation Options:

- **Direct Summation:** Loop over frames in C++ and sum ROI pixels per neuron.
- **Matrix Multiplication:** Let A be the binary (or weighted) mask matrix (pixels $\times$ neurons) and Y be frames (pixels $\times$ time). Then $A^\top Y$ yields (neurons $\times$ time).
- **GPU Reduction:** Use masked parallel reduction per neuron, producing all $F_k[t]$ in one GPU pass.

Photobleaching Note: The pipeline does not perform explicit photobleaching correction at this stage. Raw traces may exhibit slow decay; users can apply post-hoc baseline subtraction (e.g., high-pass filter or fitting an exponential) if desired.

B.2.6 Spike Detection

VOLTAGE does not prescribe a single spike detection algorithm but provides a baseline z-score thresholding approach:

1. **Z-score Normalization:** For each trace $F_k[t]$, compute a sliding-window mean $\mu_k[t]$ and standard deviation $\sigma_k[t]$ over recent frames, then

$$Z_k[t] = \frac{F_k[t] - \mu_k[t]}{\sigma_k[t]}.$$

2. **Thresholding:** Mark times where $Z_k[t] > \theta$ (typically $\theta \approx$ 2-4) as candidate spikes.
3. **Refractory Enforcement:** Enforce a minimum inter-spike interval (e.g., no two spikes within 5-10 ms) by ignoring peaks within that window.
4. **Optional Refinements:** Users may convolve $F_k[t]$ with a spike template or apply high-pass filtering before thresholding for improved SNR.

Computational Complexity:

- Trace extraction is $O(K\bar{r})$ per frame, where $\bar{r}$ is average ROI size.
- Spike detection (z-scoring + thresholding) is $O(K)$ per frame.

Implementation Notes:

- Sliding-window statistics (mean, std) can be updated in $O(1)$ per frame per neuron using exponential moving averages or running accumulators.
- Thresholding and refractory logic are trivial to implement in C++ or Python.
- Advanced methods (template matching, deconvolution) can replace the baseline z-score step without modifying upstream stages.

Summary: By decomposing processing into six GPU-friendly stages, motion-correction, shading-correction, preprocessing, segmentation, trace extraction, and spike detection, VOLTAGE achieves real-time, accurate voltage imaging analysis. Each stage is optimized for parallelism and minimal latency, while allowing modular replacement or extension.

B.3 Comparison to FIOLA:

While FIOLA employs FFT-based motion-correction and assumes fixed illumination, VOLTAGE integrates shading normalization directly via its patch-wise ZNCC formulation. This makes VOLTAGE more robust to dynamic brightness changes but also more tightly coupled in its preprocessing stages. Unlike FIOLA's modular NMF-centric model, VOLTAGE's front-end is optimized as a tightly integrated CUDA pipeline.

B.3.1 Preprocessing and Summary Image Extraction

The VOLTAGE pipeline does not include an explicit denoising stage. Instead, it achieves noise reduction indirectly through a summary-based preprocessing step that condenses temporal information into representative 2D images. These summary images highlight neuron activity and suppress unstructured noise, enabling efficient and robust segmentation.

Temporal Summarization: VOLTAGE divides the video into short temporal segments (typically 250-1000 frames) and computes two types of summary images:

- **Temporal Average:** A per-pixel mean over the segment:

  $$A(x) = \frac{1}{L}\sum_{t=1}^{L} V(x,t)$$

  This reduces uncorrelated noise and reveals persistent spatial structures (e.g., cell bodies, vasculature).

- **Max-minus-Median (Max-Med):** A transient-sensitive projection computed as:

  $$M(x) = \max_t V(x,t) - \text{median}_t(\tilde{V}(x,t))$$

  where $\tilde{V}(x,t)$ is the spatially blurred version of $V(x,t)$ (Gaussian blur with $\sigma = 3$ pixels). This operation emphasizes fluorescence transients (spikes), as neurons that fired during the segment produce bright blobs, while background and noise yield smaller or zero values.

Noise Suppression Strategy: The summary images serve to denoise by aggregation rather than per-frame filtering. The average suppresses Gaussian-like pixel noise, and the max-med removes baseline and slow fluctuations while retaining spike peaks. Gaussian blurring before computing the median reduces pixel-level noise and stabilizes the baseline estimate.

Implementation Details:

- Raw frames are buffered in GPU memory and processed in batches.
- Gaussian blurring is applied using small 2D convolutions, optimized via CUDA or PyTorch backends.
- Median computation may be approximated using fast quantile estimation or selection algorithms. While the exact median over L frames per pixel is $O(L)$, approximation techniques (e.g., temporal subsampling or histogram-based methods) reduce overhead.
- Only two 2D images (average and max-med) are retained per segment, minimizing memory.

Computational Complexity: The per-segment cost is $O(dL)$, where d is the number of pixels and L the segment length. With modern GPUs, this step takes $\sim$0.5-0.7 s for $L = 1000$ frames at 450×138 resolution (i.e., $\sim$62k pixels), amounting to $\sim$60 million operations. Operations are trivially parallelized across pixels.

Design Rationale and Comparison to FIOLA: Whereas FIOLA performs low-rank matrix factorization to explicitly separate signal and noise per frame, VOLTAGE sidesteps traditional denoising entirely. Instead, it compresses activity-rich temporal data into high-SNR summary images and uses a convolutional neural network to separate neurons from noise. This reduces compute time, simplifies implementation, and enables real-time operation. Moreover, because max-med is computed per segment, it implicitly mitigates photobleaching by anchoring baseline estimates to short time windows.

B.3.2 Segmentation

The segmentation module in VOLTAGE is central to the pipeline's performance. It identifies the spatial footprints of active, spiking neurons from summary images, using a multi-stage pipeline combining learned inference (U-Net CNN), classical image processing, and lightweight matrix factorization for demixing. This section outlines each stage in detail.

1. CNN-Based Probability Mapping: VOLTAGE uses a compact convolutional neural network (U-Net) to process pairs of summary images (average and max-minus-median) and generate per-pixel spiking probability maps.

- **Patch-based U-Net:** Input images are divided into 64×64 patches, and the U-Net is applied independently to each. This constrains model size, allows local focus, and enables full GPU parallelism.
- **Architecture:** A standard encoder-decoder U-Net [43] with skip connections is employed. The model is lightweight to ensure < 10 ms inference per patch on RTX-class GPUs.
- **Training:** The U-Net is trained on both synthetic and real datasets with annotated neuron ROIs, using data augmentation (e.g., rotations, brightness shifts) for generalization.

Each patch's output is a 64×64 probability map $P_i(x)$ indicating the likelihood of each pixel x belonging to a spiking neuron in segment i.

2. Thresholding and Morphological Filtering: To obtain discrete candidate neuron regions:

- A threshold is applied to each $P_i(x)$ to form binary maps $B_i(x)$.
- Morphological filters are used to remove small, spurious regions based on area, eccentricity, and concavity criteria.

- Overlapping or irregular regions are retained if their shape resembles a plausible neuron.

This stage outputs a clean binary mask B_i for each time segment i containing likely active ROIs.

3. Temporal Aggregation: To form a unified set of ROIs across the entire recording, VOLTAGE performs a temporal logical OR across all segment masks:

$$B_{\text{all}}(x) = B_1(x) \vee B_2(x) \vee \cdots \vee B_N(x)$$

This step merges all pixel locations that were part of any detected neuron in any time segment, producing a single mask of candidate spatial neuron footprints.

4. Demixing with Local Non-negative Matrix Factorization (NMF): To separate overlapping neurons:

1. Each connected region in B_{all} is treated independently.
2. A data matrix $P_{\text{region}} \in \mathbb{R}^{p \times N}$ is constructed, where p is the number of pixels in the region and N the number of time segments.
3. Local NMF is applied:
   $$P_{\text{region}} \approx F \cdot T$$
   where $F \in \mathbb{R}^{p \times K}$ contains spatial footprints, and $T \in \mathbb{R}^{K \times N}$ contains segment-wise activation weights.
4. K (number of neurons in the region) is chosen heuristically or via error minimization.

This step is crucial for demixing spatially overlapping neurons that spike in different segments.

5. Final ROI Assembly: The final output of segmentation is a set of clean, non-overlapping neuron masks derived from the NMF step. These masks are then used in downstream fluorescence trace extraction.

Computational and Implementation Notes:

- Patch-based U-Net inference is highly parallelizable. For a 450×138 image divided into $\sim$24 patches, total inference takes $\lesssim 50$ ms on a modern GPU.
- Morphological filtering and temporal aggregation are $O(d)$, with negligible cost.
- NMF demixing is performed on small regions (e.g., 100–200 pixels), and typically requires only a few thousand operations per region. It is performed on the CPU or GPU depending on matrix size.
- The overall segmentation runtime (including CNN and NMF) is under 1 second per segment in practice.

Comparison to FIOLA: FIOLA segments neurons via matrix decomposition of the entire video and separates spatial/temporal components globally. VOLTAGE instead performs spatial segmentation via a learned CNN on temporally compressed summary data, followed by localized demixing using NMF. This modular approach enables faster processing and real-time capability at the cost of requiring supervised model training.

Summary of Algorithm:

- **Step 1:** Generate summary images (average, max-minus-median).
- **Step 2:** Predict neuron probability maps via U-Net CNN on 64×64 patches.
- **Step 3:** Threshold and clean each prediction to produce binary ROI masks.
- **Step 4:** Combine all segment masks using logical OR.
- **Step 5:** For each connected region, apply local NMF to resolve overlapping neurons.
- **Output:** A set of spatial neuron masks for trace extraction.

This multistage segmentation pipeline is optimized for high-throughput, high-SNR detection with minimal human intervention, and is suitable for both online and offline applications.

B.3.3 Voltage Trace Extraction and Spike Detection

After segmentation, VOLTAGE extracts fluorescence (or voltage) traces for each identified neuron and optionally detects spikes. While the authors describe this stage as rudimentary, it provides a real-time baseline implementation with opportunities for extensibility.

Trace Extraction: Each neuron's ROI mask, produced during segmentation, is used to compute its fluorescence trace from the motion-corrected video.

- **Per-frame intensity average:** For each neuron k, its trace is:

  $$F_k[t] = \frac{1}{|R_k|} \sum_{i \in R_k} I_t(i)$$

  where R_k is the set of pixels in the ROI, and $I_t(i)$ is the intensity of pixel i at time t.
- **Weighted or soft masks:** Optionally, traces may be computed as a weighted sum using soft masks from the segmentation network.
- **Implementation:** This can be computed via:
  - Simple frame-wise pixel summation on CPU or GPU.
  - Matrix multiplication: using the mask matrix A (pixels $\times$ neurons) and video matrix Y (pixels $\times$ time), compute $A^T Y$.
  - ROI gather/reduce operations using CUDA or C++ for fast streaming computation.

Photobleaching Considerations: Photobleaching correction is not explicitly implemented:

- **Segmentation phase:** Each segment uses local median subtraction (e.g., for max-minus-median), mitigating bleaching within that segment.
- **Trace phase:** The extracted raw traces often show baseline drift due to photobleaching. Users are expected to apply post-hoc corrections such as:
  - High-pass filtering or baseline subtraction (e.g., moving average, exponential decay fit).
  - Computing $\Delta F/F$ using percentile-based baselines.
- **Pipeline extensibility:** A simple real-time baseline subtractor (e.g., FIR/IIR filter or DC blocker) could be added with negligible cost.

Spike Detection (Optional): While not the focus of the pipeline, basic spike detection is feasible and lightweight.

- **Z-score thresholding:** For each trace $F_k[t]$, compute:

  $$Z_k[t] = \frac{F_k[t] - \mu_k[t]}{\sigma_k[t]}$$

  using a sliding window mean μ_k and standard deviation σ_k. Spikes are detected where $Z_k[t] > \theta$, with $\theta \approx 2$–4.
- **Optional refinements:**
  - Apply high-pass filtering to remove slow drifts.
  - Require local maxima and enforce refractory period.
  - Use convolution with a spike template for enhanced SNR.
- **Advanced extensions (not implemented):**
  - Matched filtering or AR(1) deconvolution (e.g., OASIS).
  - Learning-based classifiers on trace snippets.
  - Temporal integration of CNN outputs from segmentation.

Computational Complexity:

- Trace extraction: $O(K \cdot \bar{r})$ per frame, where K is the number of neurons and $\bar{r}$ the average ROI size.
- Spike detection: $O(K)$ per frame for z-scoring and thresholding.
- Total: $O(K \cdot T)$ over T frames; trivially parallelizable on CPU or GPU.

Implementation Notes:

- Trace computation can be implemented in C++ or CUDA with parallel reduction.
- Median filtering or quantile-based baseline estimation can run in background threads.
- The default pipeline stops at trace extraction; spike detection is a post-processing step.

Comparison to FIOLA: FIOLA incorporates spike detection into its matrix decomposition framework using matched filtering and adaptive thresholds, suitable for streaming data. VOLTAGE, in contrast, adopts a modular design with optional spike detection. This separation simplifies real-time processing and allows flexible post hoc analysis.

Summary:

- VOLTAGE trace extraction is a fast, linear-time process using simple ROI averaging.
- Spike detection, while optional, can be performed with lightweight methods such as z-scoring or threshold crossing.
- Photobleaching is not corrected during extraction but mitigated in segmentation and easily addressed post hoc.
- Users may substitute this stage with any preferred method once clean ROI traces are obtained.

B.4 Summary and Comparative Analysis

FIOLA and VOLTAGE represent two complementary design philosophies for real-time analysis of high-speed voltage imaging data. Both pipelines aim to extract spike information from fast, noisy recordings in (near) real-time, but differ markedly in architecture, scope, and algorithmic emphasis.

Speed vs. Scope:

- **FIOLA** is engineered for low-latency closed-loop experiments, with a single-GPU pipeline capable of processing frames at millisecond resolution. However, segmentation is less resilient to suboptimal image conditions and is not as optimized as the other components are.
- **VOLTAGE**, in contrast, is optimized for throughput and flexibility. It decouples the analysis into modular stages that can be parallelized or swapped independently. The focus is on real-time processing of large batches of frames, with emphasis on deep learning-based segmentation and general-purpose design.

Motion and Shading Correction:

- FIOLA uses frequency-domain normalized cross-correlation (FFT-based) to perform fast translation correction. It assumes consistent illumination and does not address vignetting or spatially varying shading.
- VOLTAGE performs joint motion and shading correction using patchwise Zero-Normalized Cross-Correlation (ZNCC) in the spatial domain. This method is more robust to nonuniform illumination and small local deformations. GPU-1 handles this step in real time using integral image tricks to accelerate the patch correlation computation.

Segmentation:

- FIOLA uses a model-based approach: it identifies neurons by performing a low-rank matrix factorization on the incoming video stream. While this yields interpretable components, it may struggle with overlapping neurons or variable morphology, and initialization can be sensitive to hyperparameters.
- VOLTAGE applies a U-Net CNN trained on synthetic and real data to segment neurons from summary images (average and max-minus-median projections). The segmentation supports overlapping and irregular ROIs and is further refined by combining detections across time (via OR logic) and performing local nonnegative matrix factorization (NMF) within overlapping regions to demix multiple neurons.

Spike Detection:

- FIOLA uses matched filtering with template waveforms and adaptive thresholds to infer spikes with temporal precision directly on each frame. This enables fast, robust detection even in low SNR regimes, and includes photobleaching compensation via high-pass filtering.
- VOLTAGE, by contrast, leaves spike detection as a post-processing step on extracted traces. A baseline method using z-score thresholding is suggested, but users are expected to provide their own methods. This design reflects the pipeline's modularity: segmentation identifies spiking neurons, and the spike detection algorithm can be chosen independently.

Architectural Design and Implementation:

- FIOLA is a tightly integrated GPU pipeline. Each module operates on the output of the previous, and all data resides on the GPU to minimize transfer overhead. It uses a computational graph (e.g., TensorFlow or custom CUDA ops) and asynchronous streaming to hide latency. Memory management and execution order are optimized for minimal delay.

- VOLTAGE is more loosely coupled. It divides the work across two GPUs: GPU-1 performs motion-correction and computes summary images, while GPU-2 performs segmentation via CNN inference. The use of summary images allows major temporal data compression (from tens of thousands of frames to a few dozen 2D images), drastically reducing the computational load on the CNN. Each pipeline stage uses GPU-friendly primitives, and the entire process typically completes faster than the recording duration.

Photobleaching and Baseline Drift:

- FIOLA explicitly corrects for photobleaching during inference using a high-pass filter or DC blocker, ensuring stable baselines for spike detection throughout the recording.
- VOLTAGE does not perform photobleaching correction by default. However, since segmentation is based on short temporal segments with median baseline subtraction (in max-minus-median images), it is resilient to slow drifts during detection. Post hoc correction (e.g., moving average subtraction or $\Delta F/F$ computation) is left to the user during trace analysis.

Use Cases and Scalability:

- **FIOLA** is ideal for experiments requiring real-time feedback or online control, such as closed-loop optogenetics. Its low-latency, all-GPU architecture ensures minimal delay between signal and response.
- **VOLTAGE** is better suited for high-throughput batch processing of large imaging datasets. It is designed to scale across hardware setups, supports easy integration of new models (e.g., updated CNNs or denoisers), and is conducive to exploratory analyses and deep learning extensions.

Concluding Remarks: FIOLA and VOLTAGE represent two advanced, yet divergent, approaches to real-time neural signal extraction:

- FIOLA integrates motion-correction, ROI discovery, and spike detection into a single end-to-end GPU workflow, with each step optimized for low latency. Its reliance on hand-designed filters and adaptive thresholds ensures interpretability and robustness at the cost of flexibility.
- VOLTAGE breaks the problem into stages and applies modern tools (CNNs, local matrix factorization, temporal projections) to each. It emphasizes modularity, GPU acceleration, and data-driven inference. Although slower in per-frame latency, it achieves full analysis in a fraction of total recording time, with accuracy comparable to manual annotation.

Both pipelines are extensible and scalable. FIOLA's performance can be enhanced by parallelizing its segmentation steps across more cores or GPUs. VOLTAGE explicitly supports multi-GPU usage and can accommodate future modules (e.g., improved spike detectors, photobleaching correction) with minimal disruption.

Together, these pipelines showcase the diversity of design tradeoffs available in high-speed neural imaging: from unified, closed-loop systems like FIOLA to flexible, data-rich frameworks like VOLTAGE. Each provides a compelling blueprint for future developments in real-time neuroimaging analysis.