AIRC-Fusion: Achieving Real-Time Infrared-Visible (IR-VIS) Video Fusion on CPU-based Hardware

Rayeed Aabir Ahsan[a], Rejmin Islam[a], Fariha Rahman Hridi[a], Maria Chowdhury Prianka[a], Rashedur M. Rahman[a,*]

[a]*Department of Electrical and Computer Engineering, North South University, Dhaka-1229, Bangladesh*

* Corresponding author: Rashedur M. Rahman, *E-mail address:* rashedur.rahman@northsouth.edu

Abstract

The fusion of infrared (IR) and visible (VIS) video is a critical technology for enabling robust perception in challenging environments such as autonomous driving and surveillance. However, the computational intensity of state-of-the-art deep learning models creates a significant deployment gap, rendering them impractical for real-time, Central Processing Unit (CPU)-only systems. This paper directly challenges the notion that high-performance fusion requires specialized hardware. We introduce Vectorized FCDFusion, a novel, computationally focused implementation of the classical FCDFusion algorithm, a fast low-color deviation method proposed by Li and Fu. By redesigning its core logic for parallel execution on a standard Central Processing Unit (CPU), our method achieves a transformative speedup of over 120x compared to its original pixel-loop form, without sacrificing high-fidelity image quality. This optimization enables real-time performance, with processing speeds exceeding 150 frames per second (FPS), over 3.5 times faster than its closest traditional competitor (Model2024) on a standard CPU, without sacrificing the algorithm's high-fidelity image quality. To validate this, we conducted a rigorous comparative analysis on challenging real-world datasets (CAMEL, FLIR ADAS), revealing a clear trade-off: while our method offers unparalleled speed and excellent structural fidelity (SSIM > 0.74), a base/detail decomposition method (Model2024) provides marginally superior structural preservation, while a saliency-based approach (Model2020) excels at maximizing target contrast. Ultimately, our work provides a data-driven framework for practitioners to select the optimal algorithm for their specific mission and demonstrates that an optimized classical algorithm, validated on a Raspberry Pi 5 prototype, offers a powerful and efficient alternative for real-time multimodal perception.

Keywords: Image Fusion, Multimodal Video, Algorithmic Optimization, Vectorization, Computational Efficiency, Performance Benchmarking, Resource-Constrained Systems

1. Introduction

In an increasingly automated world, achieving reliable environmental perception across diverse and dynamic conditions remains a fundamental challenge for both machines and humans. This challenge is particularly evident when a self-driving car operates in patchy fog at dusk, where perceptual limitations can severely impair situational awareness. Visible-light (VIS) cameras, much like the human visual system, excel at capturing fine details such as road signs, lane markings, and traffic lights; however, their performance deteriorates markedly under adverse conditions. Environmental factors such as fog, rain, and low illumination significantly reduce the contrast between objects and their surroundings, resulting in substantial loss of visual information. Consequently, VIS sensors struggle to detect pedestrians in low-visibility scenarios, especially those wearing dark clothing, greatly increasing the risk of misdetection or complete visual failure in safety-critical situations [43-46]. In contrast, an infrared (IR) camera, which detects heat rather than light, can readily identify the pedestrian's thermal signature against the cool background. Neither sensor alone provides a complete picture; the VIS captures visual detail while the IR reveals hidden thermal information. The objective of image fusion is to digitally combine the complementary strengths of these sensors, creating a single, comprehensive image that is more information-rich than either source alone. This synergy provides a powerful perceptual advantage, enhancing situational awareness for more confident, informed decision-making in critical applications.

While fusing static images is a well-established field, real-world applications involve continuous video streams. This extension to video introduces the significant complexity of temporal consistency. When fusing on a frame-by-frame basis, minor process variations can cause distracting flicker or jitter as the scene changes. For practical applications like a vehicle's dashboard live video feed or a drone's monitor, a smooth, stable, and coherent video output is non-negotiable. This necessitates algorithms that not only merge spatial information within each frame but also consider how that information evolves over time.

Robust video fusion holds transformative potential across multiple domains. In the field of automotive safety and autonomous driving, as exemplified by the FLIR ADAS dataset [1], real-time video fusion plays a crucial role in ensuring reliable perception. A fused video stream allows Advanced Driver-Assistance Systems (ADAS) to sustain situational awareness even when individual sensors encounter limitations. For instance, an IR sensor can detect the heat signature of a pedestrian, while the VIS sensor concurrently verifies that the detected object is indeed a human rather than an irrelevant element such as an exhaust pipe or other non-human object. By integrating these complementary modalities, fused video provides a more complete and reliable understanding of the driving environment in real time, thereby enhancing perception robustness and navigation safety under adverse weather, low illumination, and nighttime conditions. Similarly, video fusion is essential for unmanned aerial vehicles (UAVs) and robotic systems operating in complex and dynamic environments, such as those represented in the CAMEL dataset [2, 3]. In a search-and-rescue scenario, for instance, a drone can leverage thermal imagery to locate a missing person based on body heat, while the fused VIS data provides the contextual detail required for accurate identification, obstacle avoidance, and safe route planning. Considering the energy limitations of battery-powered platforms, the underlying fusion algorithms must also be computationally efficient to ensure extended flight duration and greater operational range without compromising real-time performance.

Although the need for real-time video fusion is clear, the literature reveals a tension between algorithm performance and deployment feasibility. State-of-the-art solutions often rely on deep learning, using complex architectures that achieve remarkable quality but necessitate substantial Graphics Processing Unit (GPU) acceleration and extensive training. This creates a critical deployment gap, as their computational demands preclude use on the resource-constrained, CPU-only systems (e.g., in-vehicle computers, drone flight controllers) where they are most needed. This underscores the need for alternative approaches that prioritize computational efficiency without sacrificing quality.

This work addresses the deployment gap by presenting an optimized, computationally lightweight framework for traditional video fusion algorithms. In contrast to prior studies that focus on deep learning or static image fusion, this paper makes the following novel contributions: First, we introduce an algorithmic vectorization method that significantly accelerates traditional fusion techniques, validating their viability for real-time video processing on standard CPU hardware. Second, we systematically evaluate four distinct families of traditional fusion algorithms, providing a comprehensive performance analysis across key trade-offs: processing speed, structural fidelity, and information content. Finally, we demonstrate the real-world utility of these optimized algorithms by assessing their ability to maintain temporal consistency in dynamic scenes—a crucial factor for practical applications that is often overlooked in existing literature.

The primary goal of this project is to address the critical deployment gap between the demands of real-time video fusion and the capabilities of traditional algorithms. We aim to prove that a targeted computational strategy can unlock the potential of these methods for use on standard, CPU-only hardware. To achieve this, our study is guided by the following objectives:

- *To Quantify the Impact of Vectorization:* Determine the extent to which a pixel-wise fusion algorithm can be accelerated through vectorization and verify that this speedup does not come at the cost of fusion quality.
- *To Analyze the Performance Trade-offs:* Conduct a rigorous comparative analysis of three different traditional fusion families to map out the key trade-offs between processing speed, structural fidelity, and the enhancement of image contrast and information content.
- *To Validate Viability for Real-Time Applications:* Assess whether these optimized traditional algorithms are truly viable for sustained, real-time video processing (>30 FPS) on CPU hardware and to evaluate how their performance generalizes across different types of dynamic scenes.

To formalize these objectives, our research is structured to answer the following key questions:

- *RQ1: The Impact of Algorithmic Vectorization:* To what extent can algorithmic vectorization improve the computational performance of a traditional fusion algorithm for real-time video, and does this speedup come at the cost of fusion quality?
- *RQ2: The Trade-off Between Speed, Fidelity, and Contrast:* How do different families of traditional fusion algorithms (direct mapping, multi-scale decomposition, saliency-based) perform relative to one another across the key trade-off axes of processing speed, structural fidelity, and information content?
- *RQ3: Viability for Real-Time CPU-Based Video Fusion:* Are optimized traditional fusion algorithms viable for sustained, real-time (>30 FPS) video processing on standard, CPU-only hardware, and how does their performance generalize across different types of dynamic scenes?

2. Related Works

Image and video fusion systems have gained increasing attention in recent years due to their ability to combine complementary visual and thermal information into a single output that enhances scene perception, enabling intelligent processing and real-time decision-making, especially in low-light conditions. This section reviews relevant literature, from foundational image fusion approaches to recent advances in video and deep learning frameworks.

2.1 Image Fusion

Early image fusion methods often used simple pixel-level rules after multi-scale decomposition (e.g., wavelets, Laplacian pyramids), but struggled to preserve features and could introduce artifacts. More recent research developed sophisticated techniques that address specific challenges in the fusion process.

Traditional fusion approaches are broadly categorized into direct intensity mapping, multi-scale decomposition, and gradient-domain methods. Direct intensity mapping scales visible image intensities using thermal data for fast pixel-wise enhancement, but lacks spatial context modeling. Multi-scale decomposition techniques improve adaptability through base-detail separation and guided filtering, but are computationally expensive due to iterative filtering and reconstruction. Gradient-domain methods preserve edges by manipulating spatial derivatives, though their reintegration steps are often unstable and slow. While these methods improved fusion quality, they rely on handcrafted rules and remain limited by computational cost, which restricts their use in real-time video fusion, especially on CPU-based systems. Recent deep learning approaches [8-12] learn fusion rules end-to-end and integrate attention for modality selection, but they often prioritize visual quality over execution efficiency. Consequently, the challenge remains to achieve accurate, temporally consistent, and computationally lightweight fusion suitable for video sequences under CPU constraints.

Effective algorithm development and evaluation rely on diverse, representative datasets. For image fusion [4-23], the most commonly used datasets are TNO, MSRS, RoadScene, and M3FD, with their usage frequencies shown in Figure 1.

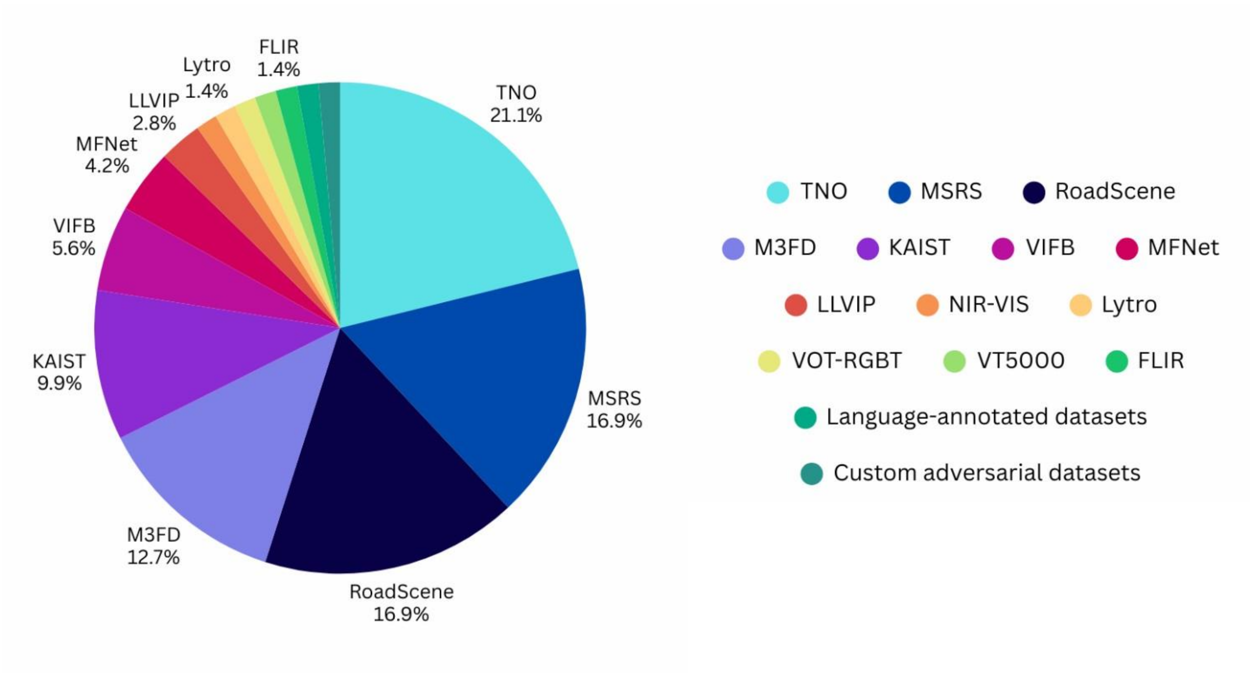


Figure 1. Frequency of public dataset usage in recent Image Fusion literature. This pie chart illustrates the distribution of dataset usage across a survey of recent image fusion research papers [4-23]. The chart highlights that a significant

portion of the field's validation relies on a few key benchmarks, namely TNO (21.1%), MSRS (16.9%), and RoadScene (16.9%). The inclusion of our test datasets, such as FLIR (1.4%), demonstrates that our experimental evaluation is grounded in established, community-recognized standards.

2.2 Video Fusion

While static image fusion is well studied, extending these techniques to video introduces additional challenges. Motion, dynamic lighting, and scene changes can break temporal coherence, making spatiotemporal consistency critical for applications such as surveillance, autonomous navigation, and drone monitoring. Early work often processed videos frame by frame, focusing only on spatial detail. Malviya and Bhirud [24], for example, combined IR and RGB streams for night vision by estimating IR backgrounds to isolate pedestrians and then embedding them into visible frames. While effective for single-frame quality, this method lacked temporal modeling and often produced flicker during object motion.

Recent studies have turned to deep learning for spatiotemporal fusion to address temporal consistency. Wang et al. [25] proposed a Progressive interaction and Temporal-Modal difference fusion Network (PTMNet), a hybrid RGB-Thermal network for video object detection, which combines early “progressive interaction” fusion with a temporal-modal difference module, improving alignment across frames and modalities. Their model reached state-of-the-art detection accuracy while running above 70 FPS. Yet, such deep models often require large, labeled datasets, which are limited for areas like aerial surveillance. To address this, Yang et al. [26] created synthetic IR data using simulation and CycleGAN, paired with an illumination-aware fusion framework for drone detection. Their system also ran on NVIDIA Jetson hardware, showing the growing emphasis on edge-ready solutions. Research has also moved beyond the IR-RGB pair. Spremolla et al. [27] fused RGB, depth, and thermal data for person tracking, relying on calibration and particle filtering. Depth helped scale target areas adaptively, improving tracking under large movements. Ben-Shoushan and Brook [28] instead focused on preprocessing: after registering RGB and thermal inputs, they extracted salient features with a statistical anomaly method, producing more reliable CNN inputs for autonomous driving. Lee et al. [29] expanded into hyperspectral video (HSV), linking it with RGB streams to predict high-resolution hyperspectral frames for water monitoring.

These works mark a progression: from frame-wise overlays [24] to deep temporal models [25], data synthesis [26], multi-modal systems [27], preprocessing [28], and hyperspectral applications [29]. Although these advances improve temporal fusion, they introduce barriers for CPU-based deployment: deep networks are computationally heavy, require large datasets, and often demand strict calibration. This highlights a clear research gap: lightweight video fusion methods that maintain temporal stability and visual quality while enabling real-time processing on CPU-limited systems.

This study tackles that gap by extending classical fusion techniques to video. Instead of deep pipelines, we focus on optimized traditional approaches with temporal filtering and motion heuristics. The aim is to test if CPU-based systems can still achieve real-time IR-Visible fusion, offering a practical benchmark for resource-constrained deployment. To provide a clear synthesis of the reviewed literature, Table 1 presents a comparative analysis of the key methodologies discussed. This summary serves to distill the primary findings and limitations of each approach, thereby highlighting the specific research gap that our proposed AIRC-Fusion system is designed to address.

Table 1 Comparative Analysis of Existing Video Fusion Studies and our Proposed Work

Reference	**Methodology**	**Findings**	**Limitations**	**Comparison with AIRC Fusion**
[24]	Frame-wise overlay using background estimation (IR & RGB).	Demonstrated effective object embedding for night vision enhancement.	Lacks temporal modeling, leading to potential flicker and motion artifacts.	Offers >120x speedup via CPU vectorization for superior real-time performance.
[25]	Hybrid fusion network (PTMNet) with temporal modeling (RGB & Thermal).	Achieved state-of-the-art accuracy and high-speed (>70 FPS) object detection.	Requires GPU acceleration and relies on large, labeled training datasets.	Achieves >150 FPS on a standard CPU, challenging the need for specialized hardware.
[26]	Synthetic data generation (CycleGAN) for an illumination aware fusion model (RGB & IR).	Enabled effective drone-based object detection on edge hardware by solving data scarcity.	Fusion performance is dependent on the quality of the synthetic IR data.	DL-free and data-independent; avoids complexity of training and data synthesis.
[27]	Particle filter with adaptive scaling using three sensors (RGB, Depth & Thermal).	Demonstrated robust person tracking, especially with radial camera motion.	Requires complex calibration for three sensors; particle filtering can be computationally intensive.	A computationally optimized classical method for two modalities; avoids complex calibration.
[28]	Statistical anomaly detection to fuse features for a CNN (RGB & Thermal).	Improved robustness of existing object detectors in varied lighting conditions.	Acts as a pre-processing step for a separate DL model, not a complete fusion system.	A complete, standalone fusion system that generates the final output directly.
[29]	Correlates video streams to predict frames (Hyperspectral & RGB).	Successfully predicted hyperspectral data from RGB video for environmental monitoring.	Highly specialized for prediction, not visual enhancement; specific to HSV-RGB modalities.	Focuses on IR-VIS visual enhancement, not cross-modal prediction.

3. Methodology

This chapter details the framework for our comparative analysis of visible-infrared image and video fusion. We follow a logical progression that mirrors our experimental pipeline. First, we describe the diverse datasets used to ensure robust evaluation. Second, we outline the standardized pre-processing pipeline applied to all data to ensure a fair comparison. Third, we present the fusion algorithms under investigation, with a detailed mathematical exposition of our primary contribution, Vectorized FCDFusion. Subsequently, we define the quantitative metrics used for performance evaluation, describe the end-to-end workflow for video fusion, and conclude with the specifics of our experimental setup.

3.1 Datasets Used in the Study

To comprehensively evaluate the performance and robustness of the fusion algorithms, we utilized a diverse set of five public datasets. These datasets present a range of challenges, including different sensor types, resolutions, and dynamic conditions. The majority of our experiments were conducted on fixed-resolution imagery (either natively or through preprocessing) to ensure a controlled comparison. For the one dataset with variable dimensions, a statistical summary is provided in Table 2. Full data acquisition details for each dataset are available in Appendix 8.2.

- *TNO Image Fusion Dataset [30, 31]:* The TNO dataset contains multispectral nighttime imagery of military-relevant scenarios. As shown in Table-7, the image sizes in the TNO dataset vary, presenting a challenge for consistent processing. The specific image pair used for this study is a 768x576 pixel image from the 'Athena' category, displayed in Figure 2(a) and (d).
- *RGB-NIR Scene Dataset [33]:* The RGB-NIR Scene Dataset contains 477 pairs of RGB and Near-Infrared (NIR) images. The images used for our evaluation are from the “field” category, shown in Figure 2(b) and (e), and have a fixed resolution of 1024x680 pixels.
- *Context-Aware Multi-spectral Light-field (CAMEL) Dataset [2, 3]:* The CAMEL Dataset features synchronized video streams with challenges such as significant resolution differences and handheld camera motion. For our experiments, all visible and thermal frames were standardized to a fixed resolution of 336x256 pixels during preprocessing to ensure consistency.
- *Landsat Dataset [32]:* The Landsat dataset provides large-scale multispectral environmental images. The full Landsat scenes are extremely large, with dimensions of approximately 8811x8851 pixels. Due to the high computational requirements, our comparative analysis was conducted on cropped sections of 1024x1024 pixels. The source bands used are shown in Figure 3.
- *FLIR ADAS Dataset v2.0.0 [1]:* The FLIR Dataset provides time-synchronized video from a vehicle-mounted system. While the thermal camera has a fixed resolution of 640x512 pixels, the visible-light camera's resolution varies, as detailed in Table 2. This presents a real-world challenge of handling inconsistent input sizes.

Dataset	Modality	Mean Dimensions (W x H)	Std. Dev. (W x H)	Min Dimensions (W x H)	Max Dimensions (W x H)

FLIR	Visible (RGB)	1139 x 915	99 x 127	1024 x 768	1224 x 1024

Table 2. Statistical Summary of Image Dimensions for the FLIR Visible-Light Dataset

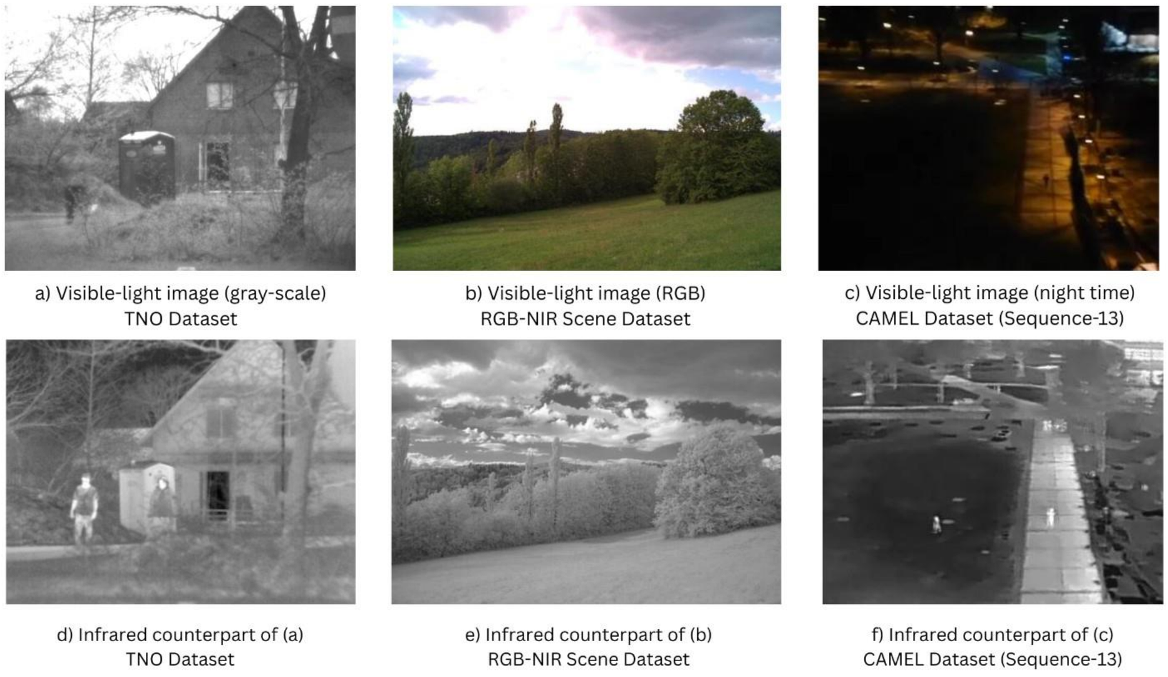

a) Visible-light image (gray-scale)
TNO Dataset
b) Visible-light image (RGB)
RGB-NIR Scene Dataset
c) Visible-light image (night time)
CAMEL Dataset (Sequence-13)
d) Infrared counterpart of (a)
TNO Dataset
e) Infrared counterpart of (b)
RGB-NIR Scene Dataset
f) Infrared counterpart of (c)
CAMEL Dataset (Sequence-13)

Figure 2. Representative image pairs from key datasets. This figure showcases the diversity of the datasets used for evaluating the fusion algorithms, illustrating different environmental conditions and sensor characteristics. The top row (a, b, c) displays the visible-light (VIS/RGB) images, while the bottom row (d, e, f) displays their corresponding time-synchronized infrared (IR/NIR) counterparts, *(a) and (d):* A grayscale image pair from the TNO Dataset, showing a nighttime military-relevant scene with a person camouflaged in the visible spectrum but clear in the thermal image, *(b) and (e):* An RGB and Near-Infrared (NIR) image pair from the RGB-NIR Scene Dataset, demonstrating a well-lit daytime scene with complex natural textures, *(c) and (f):* A nighttime image pair from the CAMEL Dataset (Sequence-13), highlighting a challenging low-light urban environment with high-contrast artificial lighting.

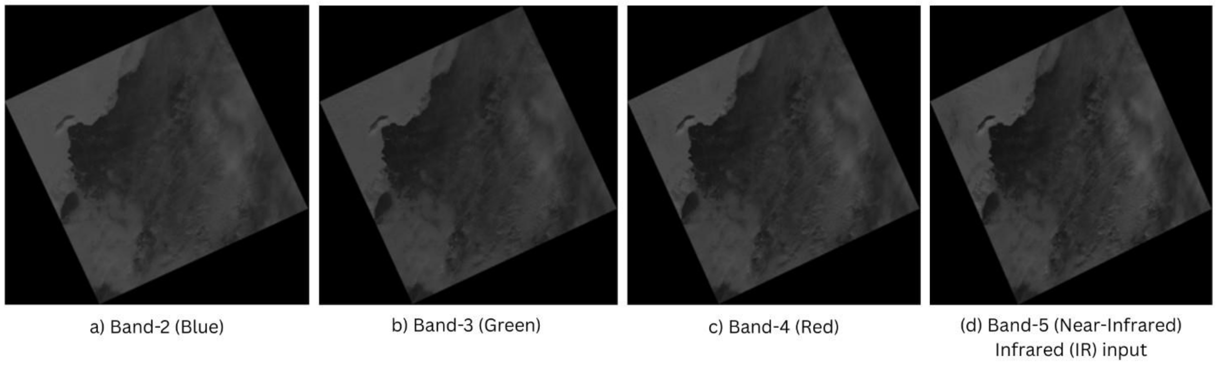


Figure 3. source bands from the Landsat 8 dataset used for fusion. This figure displays the individual multispectral bands that were used to construct the source images for our Landsat experiments. (a) Band-2 (Blue), (b) Band-3 (Green), and (c) Band-4 (Red) were combined to create the 3-channel visible-light (VIS) RGB input. (d) Band-5 (Near-Infrared) served as the source for the infrared (IR) input. These examples illustrate the subtle differences in reflectance across the bands, motivating the use of fusion to create a more information-rich composite image.

3.2 Fusion Algorithms for Comparative Analysis

In this paper, we propose a computationally efficient fusion algorithm, Vectorized FCDFusion, which is a novel, high-performance implementation of the FCDFusion method [4]. To evaluate its efficacy in terms of both speed and quality, we conduct a comparative analysis against four representative fusion algorithms. These benchmarks were specifically chosen to span the primary families of traditional fusion techniques: a direct pixel-level implementation (FCDFusion Pixel), a simple two-scale decomposition method (Model2024) [5], a saliency-based approach (Model2020) [6], and a gradient-domain technique (Model2015) [7]. The details of each algorithm are presented in Section 3.3. The detailed pseudocodes for each algorithm, including parameter specifications, are provided in Appendix 8.1.

- *Baseline FCDFusion Pixel:* This algorithm serves as the direct baseline for our proposed method and is a faithful implementation of the original FCDFusion technique [4]. It applies the same scaling factor logic as described above but calculates it for each pixel individually within a nested for-loop (see Appendix 8.1 for pseudocode). Its inclusion is crucial for directly quantifying the speed and FPS improvements achieved by our vectorization strategy.
- *Base/Detail Decomposition Fusion (referred to as "model2024" [5] in this study):* model2024 represents the class of simple, multi-scale decomposition methods. Source images are decomposed into a low-frequency base layer via Gaussian filtering ($I_{base} = I \otimes G_{\sigma}$) and a high-frequency detail layer ($I_{detail} = I - I_{base}$). The base layers are fused using weighted averaging, while detail layers are fused using a max-absolute selection rule to preserve the most prominent features (see Appendix 8.1.3). model2024 serves as a benchmark for fast decomposition-based fusion. A summary of the procedure is presented in Table 3 as Algorithm 2.

Algorithm 2: High-Level Steps for model2024 Fusion

Input:

- Visible Image I_{vis},
- Infrared Image I_{ir}

Output: Fused Image F

```
1: // Decompose each source image into base and detail layers
2: Base_vis, Detail_vis ← Decompose(I_vis) using Gaussian filter
3: Base_ir, Detail_ir ← Decompose(I_ir) using Gaussian filter
4. // Fuse each layer type separately
5: Fused_Base ← WeightedAverage(Base_vis, Base_ir)
6. Fused_Detail ← FuseDetailsBySaliency(Detail_vis, Detail_ir, method)
7. // Reconstruct the final image
8. F ← Fused_Base + Fused_Detail
9. return F
```

Note: The full implementation details, including the specific rules for detail fusion and preprocessing steps, are available in Appendix 8.1.

- *Hybrid Filtering Fusion (referred to as "model2020" [6] in this study):* model2020, based on the work of Zhang et al. [6], is representative of more complex methods that use saliency to guide the fusion process. It constructs weight maps based on the saliency (visual prominence) of features in both the base and detail layers, using a Guided Filter to ensure the weight maps are spatially consistent (see Appendix 8.1.9). model2020 provides a high-quality, though computationally more intensive, benchmark to evaluate potential trade-offs between speed and fusion quality. The overall procedure is summarized in Table 4 as Algorithm 3.

Algorithm 3: High-Level Steps for Hybrid Filtering Fusion (model2020)

Input:

- Visible Image I_{vis},
- Infrared Image I_{ir}

Output: Fused Image F

```
1: FusedImage_float ← Create empty image of same size as inputs
2: for each color channel c in (B, G, R) do
3. // Decompose source channels
4. Base_vis, Detail_vis ← Decompose(I_vis.channel[c]) using Gaussian filter
5. Base_ir, Detail_ir ← Decompose(I_ir.channel[c]) using Gaussian filter
```

```
6: // Fuse Base Layers
7: SaliencyMap_base ← CreateDecisionMap(Saliency(Base_vis), Saliency(Base_ir))
8: WeightMap_base ← RefineMapWithGuidedFilter(SaliencyMap_base, guide=Base_vis)
9. FusedBase_channel ← WeightMap_base * Base_vis + (1 - WeightMap_base) * Base_ir
10. // Fuse Detail Layers
11. ActivityMap_detail ← CreateDecisionMap(Activity(Detail_vis), Activity(Detail_ir))
12. WeightMap_detail ← SmoothMapWithGaussian(ActivityMap_detail)
13. FusedDetail_channel ← WeightMap_detail * Detail_vis + (1 - WeightMap_detail) * Detail_ir
14. // Reconstruct
15. FusedImage_float.channel[c] ← FusedBase_channel + FusedDetail_channel
16. end for
17. F ← ClipAndConvertToUint8(FusedImage_float)
18. return F
```

Note: The full implementation details, including the specific rules for detail fusion and preprocessing steps, are available in Appendix 8.1.

- *Gradient-Domain Fusion (referred to as "model2015" [7] in this study):* model2015, an implementation of the principles from Connah et al. [7], represents gradient-domain fusion. This method operates by matching the structure tensor (a matrix of local gradient information) of the fused output to that of the combined source images. This is achieved via Singular Value Decomposition (SVD) on the gradient field (see Appendix 8.1.8). Due to the high memory requirements of a fully vectorized implementation for high-resolution images, a tiled processing approach was employed. model2015 serves as a benchmark for an alternative fusion philosophy that focuses on preserving structural gradients rather than pixel intensities. The process is outlined in Table 5 as Algorithm 4.

Algorithm 4: High-Level Steps for Spectral Edge Ansatz Fusion (model2015)

Input:

- Visible Image I_vis (putative Ř)
- Infrared Image I_ir

Output: Fused Image F

```
1. // Prepare inputs
2. IR_intensity ← Grayscale(I_ir)
3. H ← StackChannels(I_vis.R, I_vis.G, I_vis.B, IR_intensity) // High-D source
4. Ř ← I_vis // Putative RGB guide
5. // Compute gradients for all pixels and channels
6. VH ← ComputeGradientMatrixField(H)
```

```
7. ∇Ř ← ComputeGradientMatrixField(Ř)
8. // Decompose gradients via SVD and construct target gradient field
9. // For each pixel, apply: ∇R_target = U(∇Ř) * Λ(∇H) * V^T(∇H)
10. ∇R_target ← ConstructTargetGradientField(∇Ř, ∇H) using SVD
11. // Reintegrate each channel of the target gradient field
12. FusedChannel_R ← Reintegrate(∇R_target.R_gradient)
13. FusedChannel_G ← Reintegrate(∇R_target.G_gradient)
14. FusedChannel_B ← Reintegrate(∇R_target.B_gradient)
// Note: Our implementation uses a placeholder for this step (see Appendix 10.1).
15. // Reconstruct final image
16. F_float ← CombineChannels(FusedChannel_R, FusedChannel_G, FusedChannel_B)
17. F ← ClipAndConvertToUint8(F_float)
18. return F
```

Note: The full implementation details, including the specific rules for detail fusion and preprocessing steps, are available in Appendix 8.1.

3.3 Proposed Method: Vectorized FCDFusion - From Pixels to Matrices

The FCDFusion algorithm proposed by Li et al. [4] offers a fast, low-color deviation fusion approach that avoids complex color space transformations. However, its original formulation is based on per-pixel processing, which is computationally restrictive for real-time video. Our primary contribution is the redesign and implementation of this algorithm using a fully vectorized approach, which we term *Vectorized FCDFusion.*

The original method iterates through every pixel *(x, y)* of the image. For each pixel, it calculates a scaling factor, *k*, based on the infrared intensity $I_{ir}(x,y)$ and the maximum component of the visible pixel $V_{max}(x,y)$. This scalar *k* is then used to scale the visible pixel's RGB values. The inefficiency arises from the nested for-loops, which process pixels sequentially and incur significant overhead in interpreted languages like Python, preventing the use of parallel hardware capabilities.

To unlock real-time performance, we refactor the entire process into a series of parallel matrix operations using the *NumPy* library. This treats the image not as a collection of pixels, but as a set of matrices, allowing calculations to occur on all pixels simultaneously. The transformation proceeds in three steps:

Step 1: Image-wide Intensity Component Extraction: Instead of finding V_{max} in a loop, we compute a 2D matrix V_{max} representing the maximum channel value for every pixel in a single operation:

$$V_{max} = np.max(I_{vis}, axis = 2) \quad (1)$$

$$where\ I_{vis}\ is\ an\ (H, W, 3)\ matrix. \quad (2)$$

Step 2: Computation of the Gain Map: The scaling factor is no longer a scalar value but a 2D matrix, or "gain map," k, of size *(H, W)*. It is computed in a single vectorized equation where all operations are performed element-wise across the matrices:

$$k = \left(\left(\frac{I_{ir}}{255.0} \right)^{\gamma} * \frac{\left(\frac{(V_{max} + 255)}{2} \right)}{(Vmax + \varepsilon)} \right) + 0.5 \tag{3}$$

Here, I_{ir} and V_{max} are *(H, W)* matrices, and a small epsilon ε (1e-6) ensures numerical stability.

Step 3: Fusion via Broadcasting: To apply the *(H, W)* gain map k to the *(H, W, 3)* visible image I_{vis}, we utilize *NumPy*'s broadcasting mechanism. The gain map is reshaped to *(H, W, 1)*, allowing it to be multiplied across all three color-channels of the visible image simultaneously:

$$I_{fused} = I_{vis} * k[:, :, np.newaxis] \tag{4}$$

This line of code replaces the entire inner loop of the original algorithm, resulting in a dramatic performance increase without altering the mathematical output. The high-level pseudocode for this vectorized process is shown in Algorithm 1 as Table 2 with a visual representation shown in Figure 4.

Algorithm 1: High-Level Steps for FCDFusion (Vectorized Implementation)

Input:

- Visible Image I_{vis},
- Infrared Image I_{ir}

Output: Fused Image

```
1. // Preprocessing
2. $Vis_{float}$, $IR_{float}$ ← PreprocessAndAlign($I_{vis}$, $I_{ir}$)
3. // Step 1: Extract Image-wide Arrays
4.  R, G, B ← GetChannels($Vis_{float}$)
5. $Intensity_{ir}$ ← GetIntensityChannel($IR_{float}$)
6. // Step 2: Compute Vmax and Gain Map 'k' (Vectorized)
7. $Vmax_{map}$ ← ElementwiseMax(R, G, B, 1.0)
8. a ← ($Intensity_{ir}$ / 255.0) ^ 2
9. $k_{intermediate}$ ← ($Vmax_{map}$ + 255.0) / 2.0
10. $k_{map}$ ← (a * $k_{intermediate}$ / $Vmax_{map}$) + 0.5
11. // Step 3: Apply Gain Map via Broadcasting
12. $k_{map}$_3D ← ReshapeForBroadcasting($k_{map}$)
13. $Fused_{float}$ ← $Vis_{float}$ * $k_{map}$_3D
14. // Step 4: Postprocessing
15. F ← ClipAndConvertToUint8($Fused_{float}$)
```

```
16. return F
```

Note: The full implementation details, including the implementation of the pixel-loop, as well as the specific rules for detail fusion and preprocessing steps, are available in Appendix 8.1.

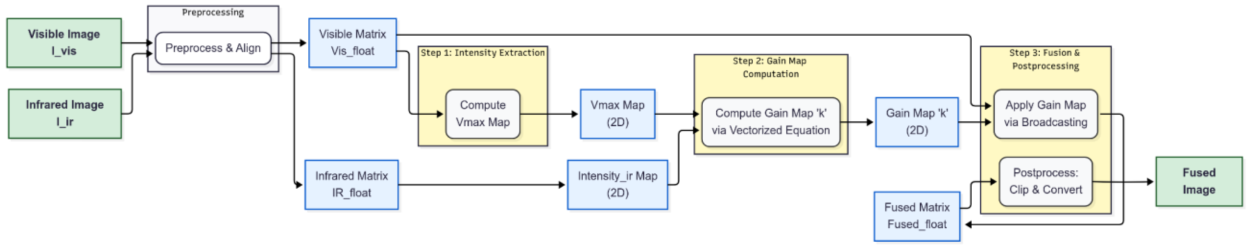
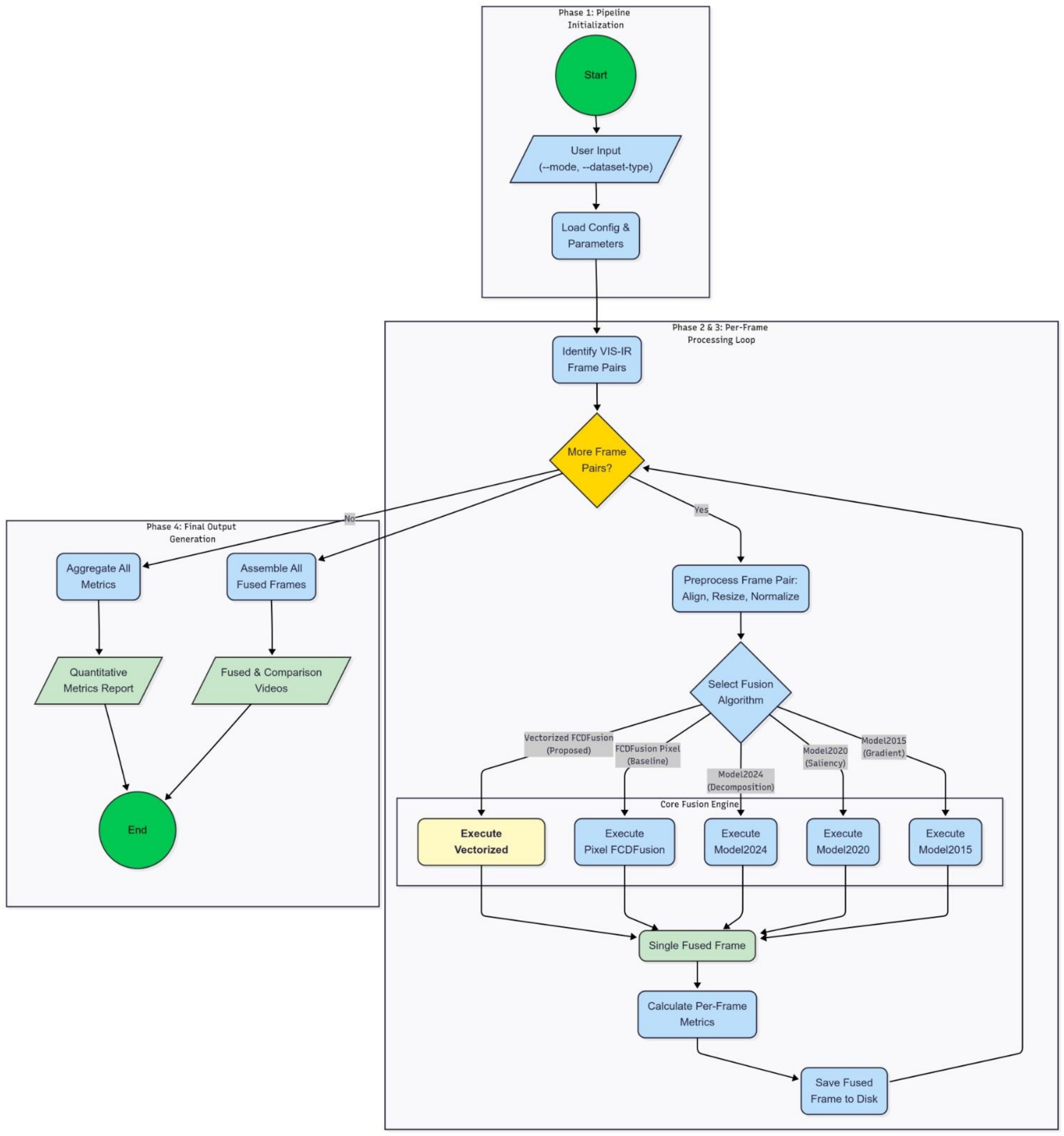


Figure 4. Detailed algorithmic flowchart for the proposed Vectorized FCDFusion method. The process shows the parallel extraction of image-wide arrays, the computation of the 2D gain map, and the final fusion step using NumPy's broadcasting mechanism.

3.4 System Architecture

Figure 5 provides a high-level overview of the software architecture of our experimental fusion pipeline. The primary script, `*main.py*`, orchestrates the entire process, utilizing `*config.py*` to manage dataset paths and hyperparameters. The `*dataset_processor.py*` module loads and pairs image or video frames, which are then passed to `*image_fusion_processor.py*`. This core module contains the implementations of all fusion algorithms, including our proposed Vectorized FCDFusion. Finally, `*evaluation_metrics.py*` computes quantitative performance scores; `*video_utils.py*` assembles the output videos; and `*benchmark_utils.py*` stores and displays the metric values in the terminal.



Phase 1: Pipeline Initialization
Start
User Input (--mode, --dataset-type)
Load Config & Parameters
Phase 2 & 3: Per-Frame Processing Loop
Identify VIS-IR Frame Pairs
More Frame Pairs?
No
Yes
Preprocess Frame Pair: Align, Resize, Normalize
Select Fusion Algorithm
Vectorized FCDFusion (Proposed)
FCDFusion Pixel (Baseline)
Model2024 (Decomposition)
Model2020 (Saliency)
Model2015 (Gradient)
Core Fusion Engine
Execute Vectorized
Execute Pixel FCDFusion
Execute Model2024
Execute Model2020
Execute Model2015
Single Fused Frame
Calculate Per-Frame Metrics
Save Fused Frame to Disk
Phase 4: Final Output Generation
Aggregate All Metrics
Assemble All Fused Frames
Quantitative Metrics Report
Fused & Comparison Videos
End

Figure 5. System Flowchart of the AIRC-Fusion Experimental Pipeline. This diagram illustrates the complete, end-to-end workflow designed for the comparative analysis of fusion algorithms. The process is divided into four sequential phases: *(1) Initialization:* The pipeline begins by loading all necessary configurations and user-defined parameters from the config.py module and command-line arguments; *(2) Data Ingestion & Preprocessing:* The system identifies and pairs corresponding visible (VIS) and infrared (IR) frames. Each pair is then passed through a standardized preprocessing step to align, resize, and normalize the data before fusion; *(3) Core Fusion Engine:* The preprocessed frame pair is fed to a selection module where one of the five benchmarked fusion algorithms is executed. Our proposed Vectorized FCDFusion (highlighted in yellow) is shown alongside the four other traditional methods;

and *(4) Post-Processing & Output Generation:* The single fused frame from the selected algorithm is saved, and its quantitative performance metrics are calculated. After the frame-by-frame loop is complete, all metrics are aggregated into a final report, and the fused frames are assembled into the final output videos.

3.5 Pre Processing

To ensure a fair and consistent comparison across all fusion methods and datasets, a standardized preprocessing pipeline was developed. All visible and infrared image pairs are passed through this pipeline before being supplied to a given fusion algorithm. The pipeline consists of three main stages: frame pair sequencing, spatial alignment & resizing, and color space & data type normalization.

- *Frame Pair Sequencing:* The initial step involves identifying corresponding pairs of visible and infrared frames for a given sequence. Our framework supports two distinct sequencing methods based on the dataset's provided metadata:
  - *JSON Mapping (for FLIR Dataset):* For the FLIR dataset, a provided *rgb_to_thermal_vid_map.json* file is used. This file contains a dictionary mapping each visible image filename to its corresponding time-synchronized thermal image filename. The pipeline iterates through these key-value pairs to load the correct image pairs.
  - *Frame Number Synchronization (for CAMEL Dataset):* For the CAMEL dataset, each sequence provides two annotation files (*SeqX-Vis.txt and SeqX-IR.txt*). The pipeline parses both files to extract the frame numbers listed in each. The set of common frame numbers present in both files is identified. Image pairs are then constructed by using a predefined filename format (e.g., "*{:06d}.jpg*") with these common frame numbers to locate the corresponding visible and infrared image files in their respective directories.

- *Spatial Alignment and Resolution Standardization:* Since the visible and infrared cameras have different sensors and resolutions, spatial alignment is crucial. This two-step process ensures all data entering a fusion algorithm is of a consistent size and format. The process is outlined in Algorithm 5 in Table 6.
  - First, to handle native resolution differences between sensors within a pair (e.g., FLIR visible at 1224x1024 and IR at 640x512), the infrared image is always resized using bilinear interpolation (*cv2.INTER_LINEAR*) to match the spatial dimensions of its corresponding visible image.
  - Second, for pipeline-based experiments, an optional target processing resolution can be specified (e.g., via the *--target-width* and *--target-height* arguments). If a target resolution is defined, *both* the visible image and the already-aligned infrared image are resized to these target dimensions. This step ensures that every frame processed in a given pipeline run is of a uniform size, which standardizes the computational load and guarantees that the resulting fused frames can be assembled into a video without further scaling. If no target resolution is specified, the images proceed at the native resolution of the visible frame.

Algorithm 5: Common Preprocessing Pseudocode

Inputs:

- Raw Visible Image (Vis_{raw}),
- Raw Infrared Image (IR_{raw}),
- Optional Target Width (T_{width}),
- Optional Target Height (T_{height})

Outputs:

- Aligned & Resized Visible (Vis_{out}),
- Aligned & Resized Infrared (IR_{out})

```
1.  // Validation (implicit)
2.  Assert Vis_raw and IR_raw are not None.
3.  // Step 1: Initial Alignment to Visible Image Dimensions
4.  Vis_aligned = Copy(Vis_raw)
5.  IR_aligned = Copy(IR_raw)
6.  IF Dimensions(Vis_aligned) != Dimensions(IR_aligned) THEN
7.      IR_aligned = Resize(IR_aligned, to=Dimensions(Vis_aligned), method=Bilinear)
8.  END IF
9.  // Step 2: Ensure 3-Channel BGR Format
10. IF NumChannels(Vis_aligned) != 3 THEN Vis_aligned = ConvertToBGR(Vis_aligned)
11. IF NumChannels(IR_aligned) != 3 THEN IR_aligned = ConvertToBGR(IR_aligned)
12. // Step 3: Optional Resizing to Target Pipeline Resolution
13. IF T_width is not None AND T_height is not None THEN
14.     IF Dimensions(Vis_aligned) != (T_width, T_height) THEN
15.         Vis_aligned = Resize(Vis_aligned, to=(T_width, T_height), method=Bilinear)
16.     END IF
17.     IF Dimensions(IR_aligned) != (T_width, T_height) THEN
18.         IR_aligned = Resize(IR_aligned, to=(T_width, T_height), method=Bilinear)
19.     END IF
20. END IF
21. Vis_out = Vis_aligned
22. IR_out = IR_aligned
23. RETURN Vis_out, IR_out
```

Table 6: Pseudocode for Common Preprocessing used in our proposed fusion pipeline.

- *Data Type Normalization:* The final preprocessing stage prepares the data type for the specific fusion algorithm. The aligned and resized *uint8* BGR images from the previous stage serve as the input for this step. Different fusion algorithms require inputs in different formats:

- *For model2024 (Base/Detail):* The *uint8* images (pixel values 0-255) are converted to *float32* and normalized to a range of [0.0, 1.0] by dividing by 255.0.
- *For FCDFusion and model2020:* These algorithms are designed to work directly with the aligned *uint8* BGR images (range 0-255), performing their own internal conversions to float for calculations as needed.

This dispatch ensures that each algorithm receives input in its optimal format.

3.6 Evaluation Metrics

To quantitatively assess the performance of the fusion algorithms, we employed a comprehensive suite of established image quality metrics. These metrics, inspired by the Visible and Infrared Image Fusion Benchmark (VIFB) [34], were chosen to evaluate key aspects of the fused image, including information content, image quality, detail preservation, and computational efficiency. For sequential data, we report the average and range (minimum-maximum) of these metrics across each dataset. The full mathematical formulation for each metric is detailed in Appendix 8.1.6, supported by the foundational work cited below:

Speed and Efficiency Metrics: These metrics evaluate the computational cost of each algorithm, a critical factor for assessing real-time viability.

- *Time Taken (ms):* The wall-clock time required to fuse a single image pair. Lower values signify faster computations.
- *Frames Per Second (FPS):* Calculated as the reciprocal of the Time Taken, providing an intuitive measure of throughput. Conversely, higher values signify faster computation periods.

Information Theory and Contrast Metrics: This group of metrics quantifies the amount of information and the level of contrast present in the fused image, based on foundational information theory. [35, 36]

- *Entropy (EN):* Based on Shannon's "A Mathematical Theory of Communication" [35], entropy measures the richness of information in an image by analyzing the distribution of its pixel intensities. A higher value generally signifies more details.
- *Mutual Information (MI):* This metric, also rooted in information theory [35], quantifies the statistical dependency between images. In our study, it is used to characterize the information shared between the original visible and infrared sources (*Mutual_Info_Src*).
- *Standard Deviation (SD) [36]:* Measures the dispersion of pixel intensities around the mean, serving as a straightforward indicator of global image contrast. Higher values are preferred.

Structural and Detail Preservation Metrics: These metrics focus on how well fine details, edges, and structures from the source images are preserved or enhanced in the fused output. [36-39]

- *Structural Similarity Index (SSIM):* As proposed by Wang et al. [37], SSIM is a widely used perceptual metric that evaluates the degradation of structural information by comparing local patterns of luminance, contrast, and structure. We report the average SSIM score against both source images, with values closer to 1.00 indicating higher fidelity.

- *Edge Sum:* This metric quantifies the total edge information by applying the Sobel operator, as described by Vincent & Folorunso [38], and summing the magnitudes of the resulting gradient map. A higher value signifies better edge preservation.
- *Wavelet Standard Deviations (cA, cH, cV, cD):* Following a single-level Haar wavelet decomposition, we analyze the energy in different frequency sub-bands. This is implemented using the PyWavelets library, a standard tool for wavelet analysis in Python [39]. The standard deviation of coefficients in the detail sub-bands (cA, cH, cV, cD) indicates the richness of textural details.
- *Noise Metric [36]:* Estimates the level of high-frequency noise by calculating the mean absolute difference between the fused image and a Gaussian-blurred version of itself, a common technique for noise estimation.

3.7 Implementation & Experimental Setup

To empirically validate our proposed method and rigorously compare it against established benchmarks, we designed a comprehensive experimental framework. This section details the hardware and software environment, the scope of the experiments conducted, and the specific workflow developed for video fusion evaluation.

Experimental Environment and Scope: All experiments were conducted on a system equipped with an AMD Ryzen 5 7600X 6-Core Processor, running Windows 10 Pro. The fusion algorithms and the evaluation pipeline were implemented in Python version 3.13.5, leveraging standard scientific libraries including OpenCV for image/video handling and *NumPy* for accelerated matrix computations. Our study encompassed a total of nine distinct experimental runs: four focusing on static, single-pair image fusion and five on dynamic video fusion, evaluating five different traditional algorithms. The complete results for all experiments are provided in Appendixes 8.2 and 8.3. For clarity and conciseness in the main body of this paper, we present a detailed analysis of four representative experiments selected to highlight key performance trade-offs:

- *For Image Fusion:* We analyze performance on a well-lit RGB-NIR scene from the NIRScene Dataset [33] and a low-light night scene from the CAMEL Dataset [2, 3].
- *For Video Fusion:* We focus on the challenging real-world automotive FLIR ADAS Dataset [1] and the dynamic, motion-rich Sequence-2 from the CAMEL Dataset [2, 3].

Implementation and Evaluation Workflow for Video Fusion: While the comparative analysis of individual image pairs is extensively studied, few, if any, rigorous comparative evaluations of fusion algorithms exist for dynamic video. This is largely due to challenges that extend beyond static image analysis, such as maintaining temporal consistency and efficiently managing large data streams. To address this, and deriving from the approach proposed by Matzner et al. [40], we reconfigured and automated our image fusion pipeline to handle sequences of frames, thereby creating a robust and versatile framework for video fusion. This workflow, detailed below, ensures a consistent and reproducible evaluation of each algorithm's performance in a dynamic setting. The process is broken down into four distinct, sequential stages:

- *Frame Ingestion and Synchronization:* The process begins by identifying all corresponding infrared and visible frame pairs for a given video sequence. As detailed in our preprocessing methodology (Section 3.5), this is handled by parsing either a JSON map (for the FLIR dataset) or synchronized text files (for the CAMEL dataset) to create an ordered list of frame pairs.
- *Sequential Frame-by-Frame Fusion:* The pipeline iterates through the ordered list of frame pairs. For each pair, the selected fusion algorithm (e.g., Vectorized FCDFusion) is executed to produce a single fused output frame. This step is performed sequentially to preserve the temporal order of the video.
- *Metrics Aggregation:* For each generated fused frame, a full suite of quality metrics (SSIM, Entropy, etc.) and the precise processing time are calculated and stored. After the entire video sequence is processed, these per-frame metrics are aggregated to compute the average, minimum, and maximum values, providing a statistical overview of the algorithm's performance and stability over time.
- *Video Assembly:* The sequence of individual fused frames is collected. Upon completion, these frames are encoded into a standard video file (e.g., *.mp4*) using *OpenCV's VideoWriter* library, set to match the source video's frame rate. This ensures correct motion playback and produces the final fused video output, which can be analyzed for visual quality and temporal artifacts like flicker.

4. Results

4.1 Single Pair Image Fusion Evaluation Metrics

For Single Pair Image Fusion, we evaluated the five fusion algorithms against one pair of images from the RGB-NIR Dataset, and one pair of images from the CAMEL Dataset, for evaluation.

- *RGB-NIR Scene Dataset:* The Evaluation Metrics of the five Traditional Python Scripts, for RGB inputs, are given in Table 7 below:

Evaluation Metrics	Model2015	Model2020	Model2024	FCDFusion (Pixel)	Vectorized FCDFusion
Proc Time	0.47889	0.03166	0.02117	0.27315	**0.00624**
Fps	2.08816	31.58195	47.23528	3.66097	**160.17964**
Entropy	4.97132	7.45509	**7.58171**	7.42699	7.42699
Mutual Info Src	**0.78505**	**0.78505**	**0.78505**	**0.78505**	**0.78505**
Std Dev	18.63863	49.31013	55.97450	**73.36079**	**73.36079**
Noise	2.42609	1.73020	2.40472	**1.72975**	**1.72975**
Ssim Accuracy	0.09507	0.71751	**0.77895**	0.77571	0.77571
Wavelet Std Ca	34.27473	98.02744	111.07870	**146.13742**	**146.13742**
Wavelet Std Ch	**10.80329**	8.00317	9.20726	8.16942	8.16942
Wavelet Std Cv	**8.77140**	6.32918	7.85322	6.15838	6.15838
Wavelet Std Cd	**4.15408**	2.24870	3.42937	2.37077	2.37077
Edge Sum	5178.58549	6470.18875	**6898.05037**	5974.74675	5974.74675

Table 7. Quantitative Evaluation of Fusion Algorithms on the RGB-NIR Scene Dataset. The best result for each metric is highlighted in bold. Mutual Information is calculated on the source images and is therefore identical for all methods.

The fused outputs of the models, with the input images in RGB and NIR forms, are given in Figure 6 below:

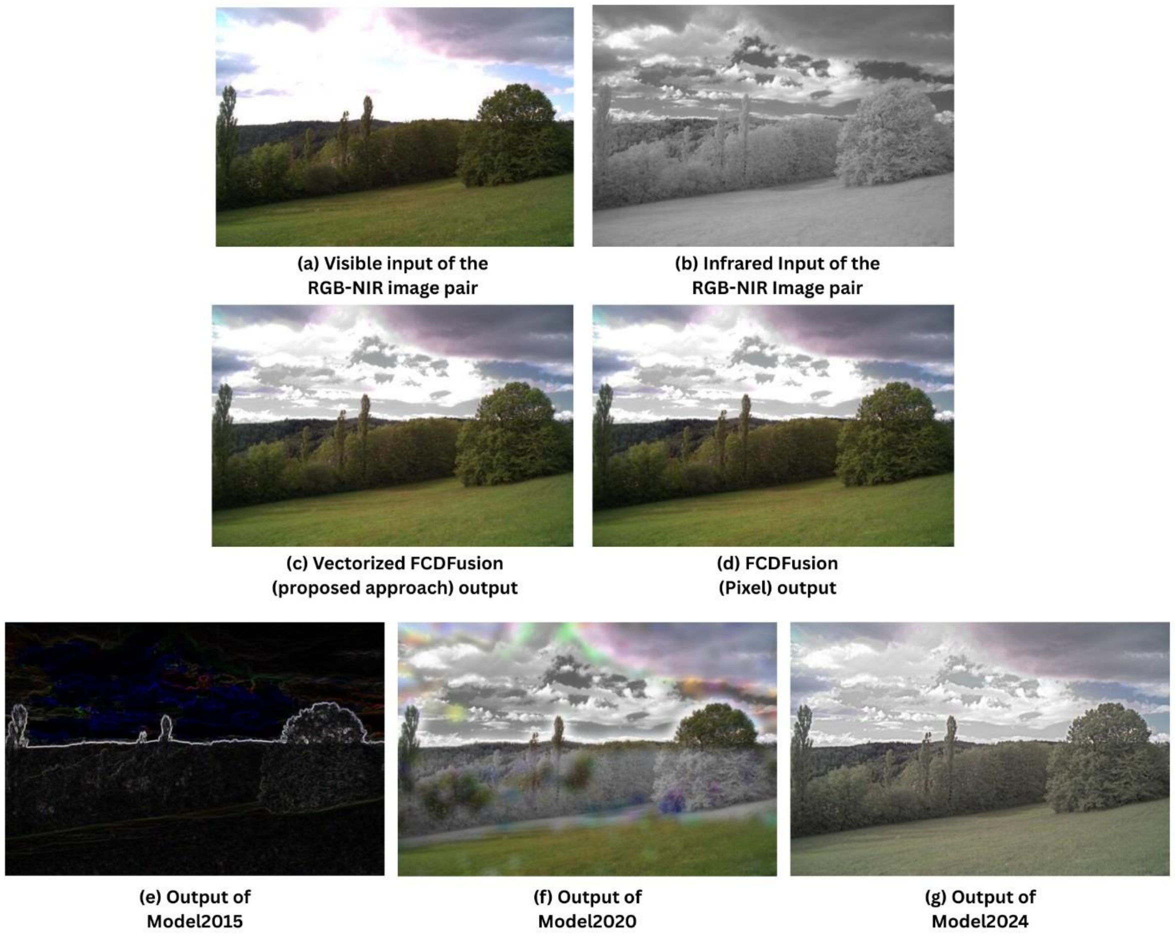

(a) Visible input of the RGB-NIR image pair
(b) Infrared Input of the RGB-NIR Image pair
(c) Vectorized FCDFusion (proposed approach) output
(d) FCDFusion (Pixel) output
(e) Output of Model2015
(f) Output of Model2020
(g) Output of Model2024

Figure 6. Qualitative Comparison of Fusion Algorithms on the RGB-NIR Scene Dataset. This figure provides a visual comparison of the fusion outputs for a well-lit daytime scene, allowing for an assessment of color fidelity and artifact introduction, (a) Source Visible (RGB) Input and (b) Source Near-Infrared (IR) Input, (c) *Vectorized FCDFusion (Proposed):* Produces a high-fidelity output with natural color rendition and clean structural lines, consistent with its high SSIM score, (d) *FCDFusion (Pixel) [4]:* The output is visually and numerically identical to the proposed vectorized version, confirming the fidelity of our optimization, (e) *Model2015 [7]:* The output shows severe artifacts and a loss of photorealism, confirming the failure of the placeholder gradient reintegration step, (f) *Model2020 [6]:* The output exhibits higher local contrast, particularly in the clouds and foliage, but introduces noticeable color artifacts (e.g., rainbow-like halos). This aligns with its high Entropy score but lower SSIM, (g) *Model2024 [5]:* The output shows exceptionally sharp details but has a slightly desaturated or grayish appearance compared to the original RGB input.

Analysis of the RGB-NIR Single Pair Evaluation: To establish a baseline for objective fusion quality, the algorithms were evaluated on a standard RGB-NIR image pair from the TNO dataset. The quantitative results, presented in Table 7, highlight a key trade-off between the different fusion philosophies. Our proposed Vectorized FCDFusion achieves the second highest Structural Similarity (SSIM) score (0.776), indicating a superior ability to preserve the structural integrity of the source visible image. This is a direct result of its intensity-scaling mechanism, which modifies pixel brightness while maintaining the original color and structural ratios, but has the fastest Time Taken (6.24ms) and FPS (160.18), thereby sacrificing minimal Accuracy but compensating that with nearly four times faster outputs compared to model2024, which has the highest SSIM score (0.779).

The qualitative results in Figure 6 visually confirm these metrics. The output from Vectorized FCDFusion (c) is notable for its natural color rendition and clean structural lines, especially on the building facade. In contrast, Model2020 (f) scores second highest on metrics of information content like Entropy (7.456). This translates visually to an image with higher local contrast, making the clouds and foliage appear to have overlaying spots. Model2024 (g) demonstrates the highest Edge Sum and Entropy, and its output shows exceptionally sharp details, but has a more grayish output, possibly due to the input IR image.

- *CAMEL Dataset (Sequence-13):* The Evaluation Metrics of the five Traditional Python Scripts, for CAMEL inputs, are given in Table 8 below:

Evaluation Metrics	**Model2015**	**Model2020**	**Model2024**	**FCDFusion (Pixel)**	**Vectorized FCDFusion**
Proc Time	0.18745	0.01080	0.01133	0.10832	**0.00230**
Fps	5.33468	92.55680	88.23798	9.23213	**434.32785**
Entropy	4.76684	**7.04158**	6.65599	5.96099	5.96099
Mutual Info Src	**0.22122**	**0.22122**	**0.22122**	**0.22122**	**0.22122**
Std Dev	19.51036	**40.25075**	33.70955	28.87264	28.87264
Noise	2.35226	1.82164	2.58071	**1.30133**	**1.30133**
Ssim Accuracy	0.21121	0.56694	**0.58470**	0.55746	0.55746
Wavelet Std Ca	36.14087	**79.45817**	65.55477	56.92588	56.92588
Wavelet Std Ch	12.71908	11.11609	**13.41253**	8.13283	8.13283
Wavelet Std Cv	6.50846	6.25632	**7.50981**	4.97658	4.97658
Wavelet Std Cd	**3.50594**	2.04339	3.40743	1.74229	1.74229
Edge Sum	2121.01770	2758.43734	**3028.22667**	1838.44728	1838.44728

Table 8. Quantitative Evaluation of Fusion Algorithms on the CAMEL Dataset (Sequence-13). This table presents the performance metrics for the five fusion algorithms on a challenging single image pair from a low-light, nighttime urban scene. The results highlight the consistent performance trends across different datasets, showcasing the trade-off between the superior speed of Vectorized FCDFusion (434.33 FPS), the high structural fidelity of Model2024 (SSIM of 0.585), and the strong contrast enhancement of Model2020 (Entropy of 7.042). The best result for each metric is highlighted in bold. Mutual Information is calculated on the source images and is therefore identical for all methods.

The fused outputs of the models, with the input images in RGB and IR forms, are given in Figure 7 below:

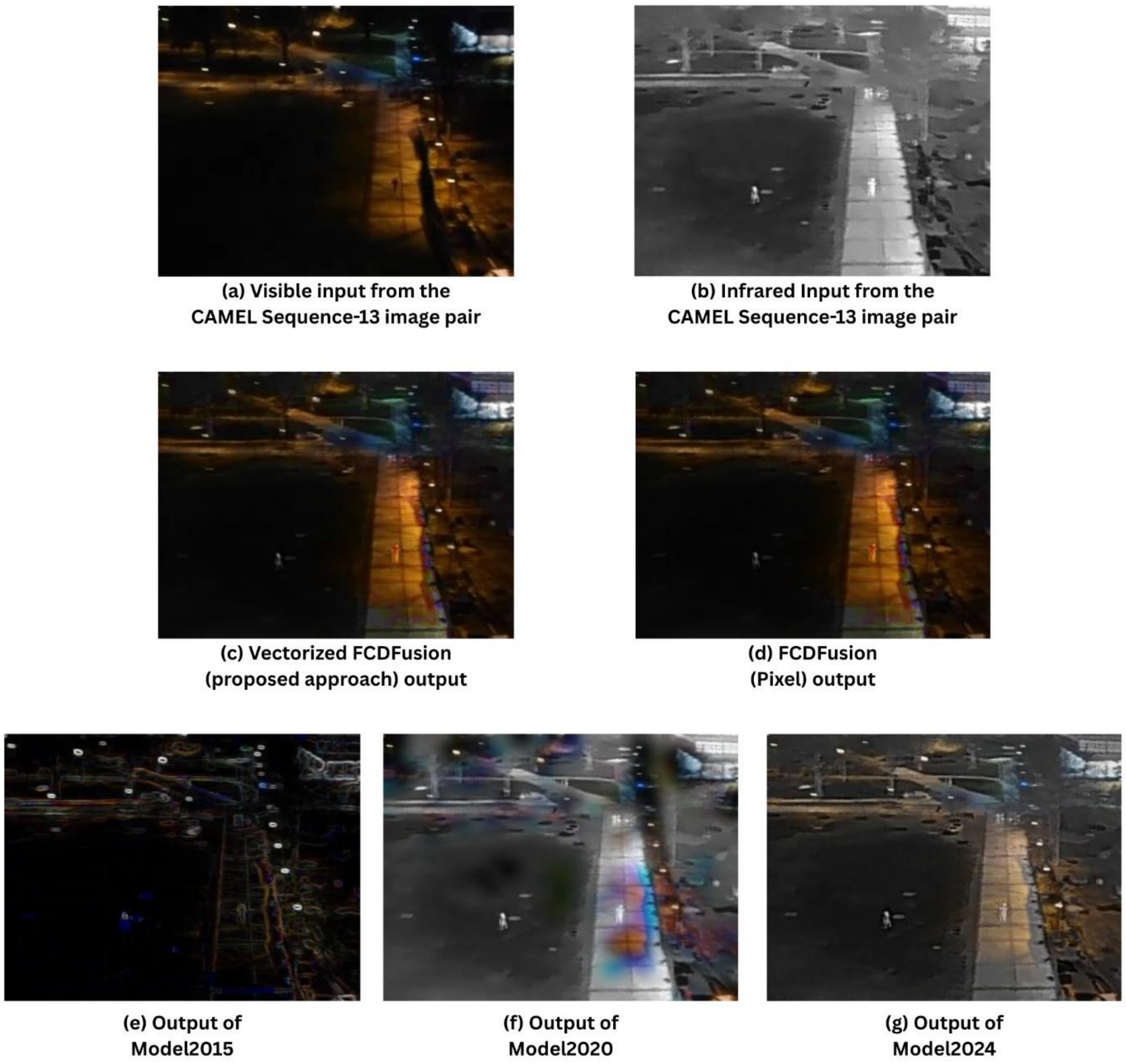

(a) Visible input from the CAMEL Sequence-13 image pair
(b) Infrared Input from the CAMEL Sequence-13 image pair
(c) Vectorized FCDFusion (proposed approach) output
(d) FCDFusion (Pixel) output
(e) Output of Model2015
(f) Output of Model2020
(g) Output of Model2024

Figure 7. Qualitative Comparison of Fusion Algorithms on a Low-Light Night Scene from the CAMEL Dataset. This figure showcases the performance of the fusion algorithms in a challenging nighttime environment with high-contrast lighting from streetlights and deep shadows: (a) *Source Visible (RGB) Input:* Shows the color information from the lights but loses significant detail in the dark, unlit areas, (b) *Source Infrared (IR) Input:* Clearly reveals the presence of human figures and the structure of the pavement, which are obscured in the visible image, (c) *Vectorized FCDFusion (Proposed):* Effectively preserves the challenging lighting conditions of the original scene, accurately rendering the color of the streetlights while integrating the thermal signatures of the pedestrians, (d) *FCDFusion (Pixel) [4]:* The output is visually and numerically identical to the proposed vectorized version, (e) *Model2015 [7]:* Fails to produce a coherent image, resulting in an output dominated by high-contrast edges and artifacts, (f) *Model2020 [6]:* Excels in contrast, making the human figures "pop" brightly against the background, consistent with its high Entropy score, but introduces some color distortion, (g) *Model2024 [5]:* Produces a structurally sound image but struggles to preserve the vibrant color of the artificial lighting, resulting in a more desaturated appearance.

Analysis of the CAMEL Single Pair Evaluation: To validate the performance on a different source, a single image pair from the CAMEL dataset, specifically from Sequence-13, was evaluated. The results, shown in Table 8, demonstrate that the performance trends observed on the RGB-NIR Scene dataset are robust and consistent across varying scenes. Vectorized FCDFusion once again leads in time taken and Frames Per Second, securing the fastest time achieved throughout the experiment at 2.30ms or 434.33 FPS, whilst having the highest Noise (1.301).

The accompanying visual analysis in Figure 7 reinforces this conclusion. The Vectorized FCDFusion output (c) preserves the challenging lighting conditions of the original scene, accurately rendering the color and contrast of the lighted pavements. The Model2020 image (f) again excels in contrast, which is consistent with its high Entropy score (7.042). These results confirm that Vectorized FCDFusion reliably produces high-fidelity fusions, while Model2020 is a strong alternative for applications where maximizing detail visibility is the primary objective.

4.2 Video Fusion

For Video Fusion Evaluation, we have evaluated the FLIR Dataset and four Sequences from the CAMEL Dataset (Sequence-1, 2, 8, and 13), with the Average values of Sequence-2 of the CAMEL Dataset, as well as the 95% Confidence Intervals, being evaluated below. The other sequences, as well as the Tables for Ranged Maximum and Minimum values and the table of average values of the other four datasets with their 95% confidence intervals, are given in Appendix 8.3, with video sequences uploaded in GitHub [41].

a) CAMEL Dataset (Sequence-2): The Average Evaluation Metrics of the five Traditional Python Scripts, for the video processing of Sequence-2, as well as the 95% Confidence Intervals are given in Table 9 below:

Evaluation Metrics	**Model2015**	**Model2020**	**Model2024**	**FCDFusion (Pixel)**	**Vectorized FCDFusion**
proc_time	0.36435 [0.36257, 0.36613]	0.03928 [0.03884, 0.03972]	0.02276 [0.02253, 0.02299]	0.28048 [0.26914, 0.29183]	**0.00663 [0.00652, 0.00675]**
fps	2.75760 [2.74486, 2.77034]	26.07000 [25.80147, 26.33854]	44.70918 [44.33432, 45.08404]	4.69507 [4.52993, 4.86022]	**159.26041 [156.62526, 161.89556]**
entropy	6.52083 [6.51939, 6.52227]	**7.59131 [7.59089, 7.59172]**	7.37546 [7.37474, 7.37619]	7.07726 [7.07607, 7.07845]	7.07726 [7.07607, 7.07845]
mutual_info_src	**0.16535 [0.16473, 0.16597]**	**0.16535 [0.16473, 0.16597]**	**0.16535 [0.16473, 0.16597]**	**0.16535 [0.16473, 0.16597]**	**0.16535 [0.16473, 0.16597]**
std_dev	33.05826 [33.02281, 33.09372]	**54.28169 [54.24785, 54.31553]**	45.49684 [45.47309, 45.52059]	38.95872 [38.92679, 38.99065]	38.95872 [38.92679, 38.99065]
noise	9.43656 [9.42555, 9.44758]	7.26633 [7.25883, 7.27384]	10.63292 [10.62527, 10.64057]	**5.77814 [5.77379, 5.78250]**	**5.77814 [5.77379, 5.78250]**
ssim_accuracy	0.09372 [0.09364, 0.09381]	0.57667 [0.57648, 0.57686]	**0.58813 [0.58788, 0.58838]**	0.57105 [0.57067, 0.57144]	0.57105 [0.57067, 0.57144]

wavelet_std_cA	56.05784 [55.99991, 56.11577]	**104.58163 [104.51063, 104.65264]**	82.11758 [82.06811, 82.16705]	74.64821 [74.58292, 74.71350]	74.64821 [74.58292, 74.71350]
wavelet_std_cH	26.17410 [26.13622, 26.21197]	21.02859 [21.01131, 21.04588]	**28.18760 [28.17112, 28.20408]**	16.24728 [16.23578, 16.25878]	16.24728 [16.23578, 16.25878]
wavelet_std_cV	19.55588 [19.53515, 19.57661]	17.32409 [17.30833, 17.33986]	**23.00226 [22.98577, 23.01874]**	13.17732 [13.16760, 13.18703]	13.17732 [13.16760, 13.18703]
wavelet_std_cD	12.69779 [12.67614, 12.71945]	10.29657 [10.28403, 10.30910]	**14.58391 [14.56988, 14.59794]**	7.81056 [7.80327, 7.81785]	7.81056 [7.80327, 7.81785]
edge_sum	6303.68356 [6296.05787, 6311.30925]	6166.29038 [6161.88220, 6170.69855]	**7860.94888 [7857.07131, 7864.82646]**	4646.31679 [4642.87726, 4649.75632]	4646.31679 [4642.87726, 4649.75632]

Table 9. Average Performance Metrics with 95% Confidence Intervals for the CAMEL Sequence-2 Video Dataset. This table presents the aggregated performance metrics over all 809 frames of the dynamic, motion-rich CAMEL Sequence-2. Each cell displays the mean value, with the 95% confidence interval shown in brackets below it. The results demonstrate the viability of Vectorized FCDFusion for real-time applications, as it is the only algorithm to significantly exceed the 30 FPS threshold, achieving an average of 159.26 FPS. The best result for each metric is highlighted in bold. Mutual Information is calculated on the source images and is therefore identical for all methods.

Analysis of the CAMEL Dataset Evaluation: For video fusion, computational efficiency is the most critical performance indicator. The algorithms were benchmarked on the CAMEL Sequence-2 images, which were then merged into a side-by-side video sequence, with average metrics reported in this section, and ranged metrics reported in Appendix 8.3. The results unequivocally establish the viability of our proposed method for real-time applications.

Vectorized FCDFusion was the only algorithm to significantly exceed real-time performance thresholds, achieving an average processing speed of 159.26 FPS. This represents a nearly 42-fold speed increase over its non-vectorized counterpart (4.70 FPS) and is more than thrice as fast as the next-best algorithm, Model2024 (44.71 FPS). Importantly, this high speed was not achieved at the expense of a significant drop in quality. Vectorized FCDFusion maintained the third highest average SSIM score (0.571) throughout the sequence, confirming its ability to deliver both high-speed and high-fidelity fusion consistently across time-varying frames.

b) FLIR Dataset: The Average Evaluation Metrics of the five Traditional Python Scripts, for the video processing of the FLIR Dataset, are given in Table 10 below, with the complete table present in Appendix 8.3:

Evaluation Metrics	**Model2015**	**Model2020**	**Model2024**	**FCDFusion (Pixel)**	**Vectorized FCDFusion**
			Processing Speed		
Avg. FPS (95% CI) ↑	0.22 [0.22, 0.22]	2.23 [2.21, 2.25]	3.57 [3.55, 3.59]	0.08 [0.08, 0.08]	**10.05 [9.96, 10.15]**
			Information Content		
Avg. Entropy (95% CI) ↑	4.48 [4.47, 4.50]	**7.33 [7.33, 7.34]**	6.95 [6.94, 6.96]	7.12 [7.11, 7.13]	7.12 [7.11, 7.13]
			Image Quality		

Avg. Std. Dev. (95% CI) ↑	11.94 [11.78, 12.10]	**46.89 [46.67, 47.11]**	34.92 [34.55, 35.29]	43.39 [42.88, 43.89]	43.39 [42.88, 43.89]
Avg. Noise (95% CI) ↓	2.37 [2.35, 2.39]	**1.17 [1.16, 1.18]**	2.21 [2.19, 2.23]	1.59 [1.57, 1.61]	1.59 [1.57, 1.61]
Avg. SSIM (95% CI) ↑	0.14 [0.14, 0.14]	0.70 [0.69, 0.70]	**0.75 [0.75, 0.75]**	0.75 [0.74, 0.75]	0.75 [0.74, 0.75]
Detail & Edge Preservation					
Avg. Edge Sum (x10^3) (95% CI) ↑	20.63 [20.38, 20.88]	20.34 [20.11, 20.57]	**22.39 [22.13, 22.66]**	18.67 [18.46, 18.87]	18.67 [18.46, 18.87]

Table 10. Average Performance Metrics with 95% Confidence Intervals for the FLIR ADAS Video Dataset. This table presents the aggregated performance metrics over all 3,749 frames of the challenging, high-resolution FLIR automotive dataset. Each cell displays the mean value, with the 95% confidence interval shown in brackets. The results highlight the performance degradation on larger images but confirm that Vectorized FCDFusion remains the only algorithm to achieve interactive frame rates (10.05 FPS). Furthermore, Model2024 and Vectorized FCDFusion both achieve the highest structural fidelity (SSIM ≈ 0.75). The best result for each metric is highlighted in bold.

A snapshot of the fused outputs of the models, with the input images in RGB and IR forms, are given in Figure 8 below:

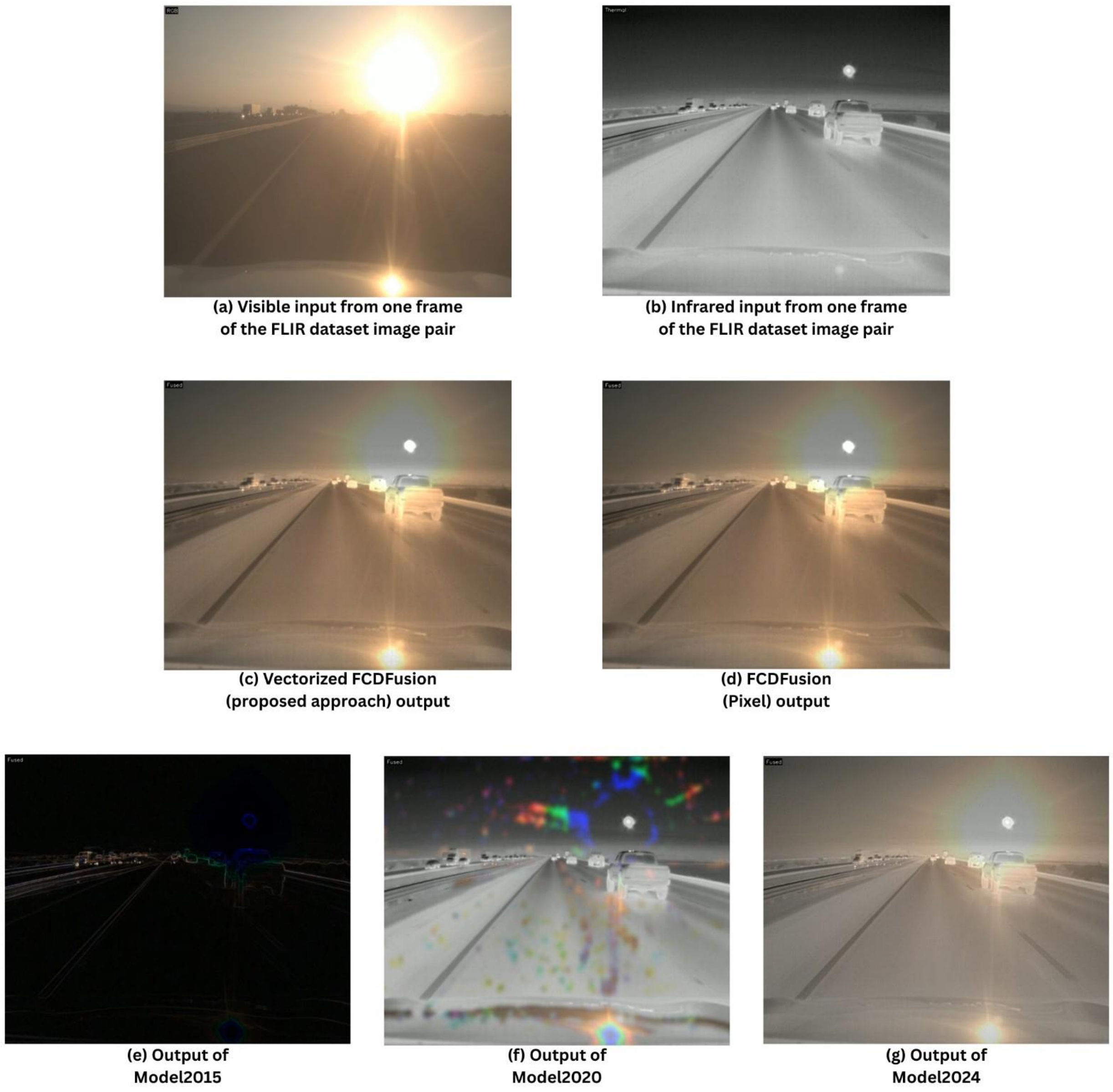

RGB
Thermal
(a) Visible input from one frame of the FLIR dataset image pair
(b) Infrared input from one frame of the FLIR dataset image pair
Fused
Fused
(c) Vectorized FCDFusion (proposed approach) output
(d) FCDFusion (Pixel) output
Fused
Fused
Fused
(e) Output of Model2015
(f) Output of Model2020
(g) Output of Model2024

Figure 8. Qualitative Comparison of Fusion Algorithms on a High Dynamic Range Scene from the FLIR ADAS Dataset. This figure demonstrates the performance of the fusion algorithms in a challenging real-world driving scenario featuring extreme solar glare that compromises the visible-light camera: (a) *Source Visible (RGB) Input:* The image is almost completely washed out by the sun's glare, making the vehicles ahead difficult to discern, (b) *Source Infrared (IR) Input:* The thermal sensor is unaffected by the glare and clearly shows the heat signatures of the vehicles on the road, (c) *Vectorized FCDFusion (Proposed):* The output successfully integrates the thermal signature of the vehicles while preserving the natural color of the scene and mitigating the excessive solar glare. The result is a clear, color-accurate, and structurally sound image crucial for driver awareness, (d) *FCDFusion (Pixel) [4]:* The output is visually and numerically identical to the proposed vectorized version, (e) *Model2015 [7]:* The output fails to produce a photorealistic image and is dominated by edge artifacts, (f) *Model2020 [6]:* This method produces the highest

contrast for the thermal targets, making the vehicles "pop" brightly. However, it introduces significant color artifacts and a less natural-looking scene, (g) *Model2024 [5]:* Effectively fuses the scene but produces a desaturated, hazy output with a prominent halo artifact around the sun.

Note: Full video segments for each fusion algorithm are provided in the GitHub link [41].

Analysis of the FLIR Dataset Video Fusion Evaluation: The FLIR ADAS dataset provides a challenging, real-world scenario for evaluating fusion in the context of automotive applications. As shown by the average metric values, Vectorized FCDFusion once again delivers a dominant performance in terms of speed, being the only algorithm to achieve over 10 FPS, with 10.05 FPS, making it very suitable for real-time in-vehicle systems.

The practical implications are illustrated in the representative frames shown in Figure 8. The output from Vectorized FCDFusion (c) is clear, color-accurate, and structurally sound, which is crucial for maintaining driver awareness. It successfully integrates the thermal signature of the vehicles, and avoids the excessive glare of the sun in the RGB image, thereby resulting in an output where both the sun's glare as well as the vehicles obscured by it can both be seen. While Model2020 (f) enhances the thermal signature to make the vehicles "pop" with slightly higher contrast—a potential advantage for an automated detection system—it does so at a non-real-time frame rate (3.57 FPS) and with a less natural color profile. This final test confirms that our proposed Vectorized FCDFusion effectively balances the critical requirements of high-speed processing and high-fidelity visual output, with minimal sacrificing of the Structural Similarity whilst maintaining exceptional speeds, making it an excellent candidate for practical video fusion tasks.

4.3 Ablation Study of the FCDFusion Algorithm

To validate the key design components within the Vectorized FCDFusion algorithm and understand their interactions, we conducted a comprehensive ablation study on the test pair of images from the RGB-NIR Scene dataset [33]. This analysis systematically simplifies and removes three core components of the gain map equation—the gamma correction, the base gain offset, and the dynamic brightness target—both individually and in combination. The quantitative and qualitative results of this study are presented in Table 11 and Figure 9, respectively.

Algorithm Variant	SSIM (Fidelity) ↓	Entropy (Info) ↓	Std. Dev. (Contrast) ↓	**Visual Effect (See Figure 9)**
Full FCDFusion (Proposed)	0.77571	**7.42700**	**73.36079**	Balanced and natural output (a)
		Single Ablations		
No Gamma (γ=1)	0.76920	6.54860	70.11560	Reduced contrast and flatter image (b)
No Base Gain (+0.5)	0.56590	6.99420	44.26180	Severe loss of detail in dark regions (c)
No Dynamic Target	**0.77770**	7.41800	66.47300	Washed-out highlights in the sky (d)
		Combined Ablations		

No Base Gain & Dynamic Target	0.64016	7.03509	40.60305	Dark image with washed-out sky (e)
No Gamma & Dynamic Target	0.72568	6.38998	54.35337	Very flat image with washed-out sky (f)
No Gamma & Base Gain	0.77417	7.33776	47.37588	Low contrast but avoids crushed blacks (g)
No Gamma, Base Gain & Dynamic Target	0.75786	7.10288	36.36293	Basic scaling, low contrast, washed-out sky (h)

Table 11: Results of the ablation study on the RGB-NIR Scene dataset. While the "No Dynamic Target" variant achieves a marginally higher SSIM, the full proposed algorithm provides the best balance of fidelity, contrast, and information content.

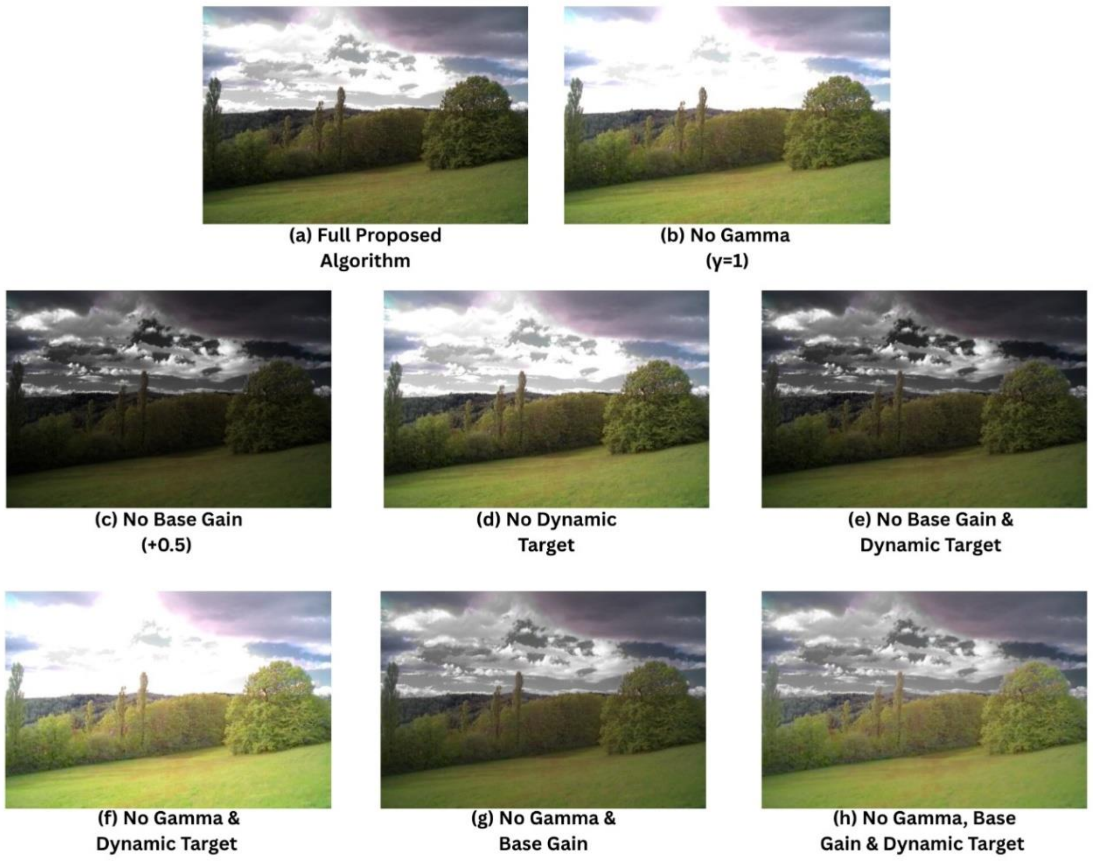

(a) Full Proposed Algorithm
(b) No Gamma (γ=1)
(c) No Base Gain (+0.5)
(d) No Dynamic Target
(e) No Base Gain & Dynamic Target
(f) No Gamma & Dynamic Target
(g) No Gamma & Base Gain
(h) No Gamma, Base Gain & Dynamic Target

Figure 9. Qualitative Results of the Ablation Study on the FCDFusion Algorithm. This figure visually demonstrates the impact of systematically removing key components of the gain map equation, with all results corresponding to the quantitative data in Table 11: *(a) Full Proposed Algorithm:* The baseline output, showing a naturalistic and balanced fusion, *(b) No Gamma:* The image appears flatter and less vibrant, demonstrating the loss of thermal contrast, *(c) No Base Gain:* The foreground and forest are plunged into shadow, showing a severe loss of detail in dark regions, *(d) No Dynamic Target:* The details in the clouds are washed out and the sky is overly bright, *(e) No Base Gain & Dynamic Target:* Combines the flaws of (c) and (d), resulting in a dark image with washed-out highlights, *(f) No Gamma & Dynamic Target:* A very flat and desaturated image with blown-out highlights in the sky, *(g) No Gamma & Base*

Gain: A low-contrast image that avoids the catastrophic "crushed blacks" seen in (c) , *(h) All Components Removed:* The algorithm reverts to a basic scaling, resulting in a low-contrast image with washed-out highlights.

Analysis of Findings: The results of the ablation study, presented in Table 11, demonstrate the critical and often interconnected role each component plays in achieving a high-quality fusion.

- *The Indispensable Role of the Base Gain:* The most significant degradation occurred in the "No Base Gain" variant, where removing the + 0.5 offset caused the SSIM to plummet to 0.5659. This is visually confirmed in Figure 9(c), where the foreground and forest are plunged into an unnatural, dark shadow with a severe loss of texture and detail. This definitively proves the base gain's essential role in preserving information in dark regions and preventing "crushed blacks."
- *The Impact of Gamma on Contrast:* The "No Gamma" variant resulted in the lowest Entropy score (6.5486), producing a visually flatter image that lacks the contrast and perceptual "pop" of the full algorithm, as seen in Figure 9(b). This confirms that the gamma correction (γ=1) is crucial for effectively enhancing the thermal contrast and overall information content of the fused image.
- *A Counter-Intuitive Trade-off between Metrics and Perception:* Interestingly, the "No Dynamic Target" variant produced a marginally higher SSIM score (0.7777) than our full proposed method (0.7757). This result reveals a subtle but important trade-off between raw metric scores and perceptual quality. By using a fixed high brightness target, this variant washes out the detail in the clouds, as seen in Figure 9(d). This brighter, lower-contrast sky happens to be structurally closer to the source near-infrared image (which also has a bright, less-detailed sky), thus slightly increasing the average SSIM. However, our proposed full algorithm, with its dynamic target, intentionally preserves the more detailed highlights from the visible-light source, producing a more naturalistic and visually balanced scene (Figure 9(a)) at the cost of a statistically insignificant drop in the SSIM metric.
- *Interaction Effects:* The combined ablations reveal a complex interplay between components. Notably, the "No Base Gain & Gamma" variant (SSIM = 0.77417) performed significantly better than removing the base gain alone. This suggests that the aggressive gamma correction may exacerbate the information loss in dark regions when the base gain is absent. By removing both, the algorithm reverts to a simpler, more linear scaling that, while less dynamic, avoids the catastrophic failure mode of the "No Base Gain" variant.

Conclusion of Ablation Study: This analysis confirms that all components of the FCDFusion equation are necessary and work in concert. While the base gain has the most dramatic structural impact, all are essential for producing an output that achieves the optimal balance of structural fidelity, information content, and, critically, superior perceptual quality.

5. Application and Deployment

The ultimate measure of a fusion algorithm's success is its real-world viability. Beyond academic benchmarks, this section discusses the practical deployment pathways for Vectorized FCDFusion, contextualizing its performance advantages within specific operational scenarios.

5.1 Autonomous Systems: ADAS and Unmanned Vehicles (UAVs)

- *Advanced Driver-Assistance Systems (ADAS):* The algorithm's real-time processing of the FLIR ADAS dataset (>10 FPS) makes it well-suited for automotive applications. Deployed on an in-vehicle CPU, it can provide a clear, color-accurate fused video feed to enhance the detection of hazards like pedestrians and animals in low-light or adverse weather. Its high fidelity (SSIM > 0.74) ensures a non-distorted scene, which is critical for driver situational awareness.
- *UAVs and Robotics:* On battery-powered platforms like drones and ground robots, where computational efficiency is paramount, Vectorized FCDFusion's ability to run on low-power CPUs is ideal for onboard sensor fusion. This enhances navigation in GPS-denied environments, improves object detection for collision avoidance, and provides real-time intelligence for applications like search and rescue or infrastructure inspection.

5.2 Surveillance and Security: For security applications, our algorithm offers a cost-effective, high-performance solution to upgrade existing surveillance infrastructure.

- *24/7 All-Weather Monitoring:* By fusing IR and visible streams, the system provides a consistent, information-rich video feed in all conditions. The high frame rate and temporal stability reduce the cognitive load on human operators, improving threat detection.
- *Pre-processing for AI Analytics:* Vectorized FCDFusion can serve as a robust pre-processing step for downstream AI analytics. Fusing streams on a CPU before sending data to a GPU-based server can reduce bandwidth and improve input quality for analytical models, potentially enhancing their accuracy and robustness.

5.3. Prospective Scientific and Industrial Applications: Beyond surveillance and autonomous systems, the framework's real-time performance and high fidelity enable new applications in demanding scientific and industrial sectors, particularly where dynamic, multi-modal video is critical for understanding complex physical phenomena.

- *High-Speed Material Stress and Fracture Analysis:* A prime application is in materials science for analyzing material failure under stress. In a typical tensile test, a material sample is subjected to increasing force until it fractures. This process is often recorded with high-speed cameras to study crack propagation. However, a visible-light camera can only see a crack *after* it has formed. An infrared camera, by contrast, can detect the minute thermal changes caused by friction and localized deformation at points of high stress *before* a visible fracture occurs. By fusing a high-speed visible video stream with a synchronized thermal stream, our system would provide an unprecedented diagnostic tool. Engineers could create a single video that visually correlates the buildup of thermal hotspots (from the IR camera) with the exact moment of micro-fracture initiation (from the VIS camera). This application is only viable with a high-frame-rate system. The performance of Vectorized FCDFusion makes it an ideal candidate for processing the data from high-speed cameras in real-time. Furthermore, the high-contrast output of Model2020 could be programmatically selected to specifically enhance and isolate the thermal stress points for automated failure prediction algorithms.

- *Astrophysical and Geophysical Monitoring:* The framework is also well-suited for dynamic event tracking in astrophysics and earth sciences, where events are captured across multiple spectral bands.
  Solar Flare Dynamics: A solar observatory could use our system to fuse visible-light (H-alpha) video of the Sun's chromosphere with an Extreme Ultraviolet (EUV) video, which acts as a thermal map. This would allow solar physicists to observe in a single real-time stream how underlying magnetic field structures (visible) contain and violently release superheated plasma (thermal), improving the understanding of mechanisms that drive space weather.
- *Volcanic Lava Flow Tracking:* A drone with our CPU-based system could monitor an active lava flow. Fusing visible video (showing terrain and cooled crust) with thermal video (hot, fast-moving molten channels) would provide a single, unambiguous data stream, allowing emergency responders to better predict the flow's path and speed, even through smoke and steam.

These prospective applications demonstrate the versatility of our lightweight fusion pipeline. By providing a computationally efficient solution, this work is an enabling technology that makes high-performance, real-time multimodal video analysis accessible to a new range of scientific and industrial fields.

5.4 Live Hardware Prototype and Deployment Demonstration: To provide the definitive validation of our research and demonstrate tangible, real-world viability, we moved beyond simulation and developed a fully operational, real-time fusion prototype on a resource-constrained embedded platform: a Raspberry Pi 5. This hardware demonstration serves as the ultimate proof-of-concept for the deployment pathways discussed above, showcasing the practical application of our findings on a low-power, CPU-only system.

- *Hardware and Software Implementation:* The prototype utilizes two live video streams: a *Raspberry Pi NoIR Camera V2* for the thermal/infrared input and a *Logitech Brio 100 USB webcam* for the visible-light (RGB) input. The entire software stack, including the operating system, camera drivers, and our full Python fusion pipeline, runs directly on the Pi's ARM processor.
- *Real-Time, Multi-Threaded Architecture:* To ensure a responsive and stable live feed, the demonstration script `*raspberry_pi11.py*` is a multi-threaded application. Each camera operates on a dedicated thread, continuously capturing the latest frame and making it available to the main process. The main thread is then free to focus exclusively on fetching the most recent frames, performing the fusion calculation using the currently selected algorithm, and rendering the output. This architecture is critical for preventing the computationally intensive fusion process from blocking frame acquisition, which results in a smooth and high-frame-rate video output.
- *Interactive Control and Parallax Alignment:* A key challenge in any real-world, multi-camera system is parallax: the slight difference in perspective between physically separate sensors. Our prototype directly addresses this with an interactive, on-the-fly manual alignment system. The operator can use keyboard controls to apply fine-grained translation (tx, ty) and scaling adjustments to each video stream independently. This allows for the precise co-registration of the two camera views in real-time, a crucial capability for

calibrating a deployed system in the field. These alignment parameters can be saved and reloaded, demonstrating a complete solution for practical deployment.

This successful deployment on a low-power, single-board computer is the final and most powerful validation of our central thesis. It proves that by leveraging a targeted computational strategy like vectorization, traditional fusion algorithms are not merely a theoretical alternative, but a powerful, efficient, and readily deployable solution for real-time multimodal perception on standard hardware. A video of this live demonstration is available on our project's GitHub repository [41].

6. Discussion

Our approach was to evaluate four traditional Python fusion algorithms: Gradient-Preserving Spectral Mapping [7] (named Model2015 in "Methodology" and subsequent Chapters), Hybrid Image Filtering [6] (Model2020), Weight Induced Contrast Map [5] (Model2024), and FCDFusion [4]. During testing of the FCDFusion's pixel-by-pixel loop algorithm [4], which became our fourth fusion algorithm and Python's *NumPy* library, we developed a highly optimized version termed *"Vectorized FCDFusion"*. This fifth model uses *NumPy*'s array-based operations to significantly improve calculation efficiency without altering the algorithm's nature. Consequently, it produces numerically identical results to the original across all quality metrics, with transformative improvements only in processing time (ms) and FPS. We focused on traditional algorithms to create a real-time video fusion model for common hardware, as they do not require the extensive training data and powerful GPUs necessary for neural network models, which are often unavailable for typical surveying and surveillance platforms.

6.1 The critical role of Implementation: Vectorization of FCDFusion

The most critical finding of this research is the dramatic performance improvement achieved through vectorization. The original pixel-by-pixel implementation of FCDFusion [4] proved computationally prohibitive for video-rate processing. As detailed in Table 12, its average frame per second values ranged from 9.23 FPS (108ms) for a single image pair to below 0.10 FPS (over 14 seconds) on the high-resolution FLIR dataset.

By refactoring the algorithm with optimized *NumPy* array operations, our proposed Vectorized FCDFusion achieves a major performance gain without compromising image quality. This vectorized version produces numerically and visually identical outputs to the original, preserving all quality characteristics (e.g., an average SSIM of 0.747 on the FLIR dataset for both implementations).

The performance gain, however, is transformative. Table 13 quantifies this, showing a speedup of 34x to 125x across datasets compared to the original. This optimization transforms FCDFusion from a theoretical method into a practical tool for high-frame-rate applications, demonstrating that algorithmic implementation is as crucial as algorithmic design for real-world deployment.

Fusion Algorithms	RGB-NIR	CAMEL Image	CAMEL Seq-2	FLIR Dataset

model2015	2.08816	5.33468	2.7576	0.22088
model2020	31.58195	92.5568	26.07	2.22544
model2024	47.23528	88.23798	44.70918	3.56919
FCDFusion Pixel	3.66097	9.23213	4.69507	0.08062
FCDFusion Vectorized	**160.17964**	**434.32785**	**159.26041**	**10.05231**

Table 12: Frames Per Second (FPS) for each Fusion Algorithm in the four experiments.

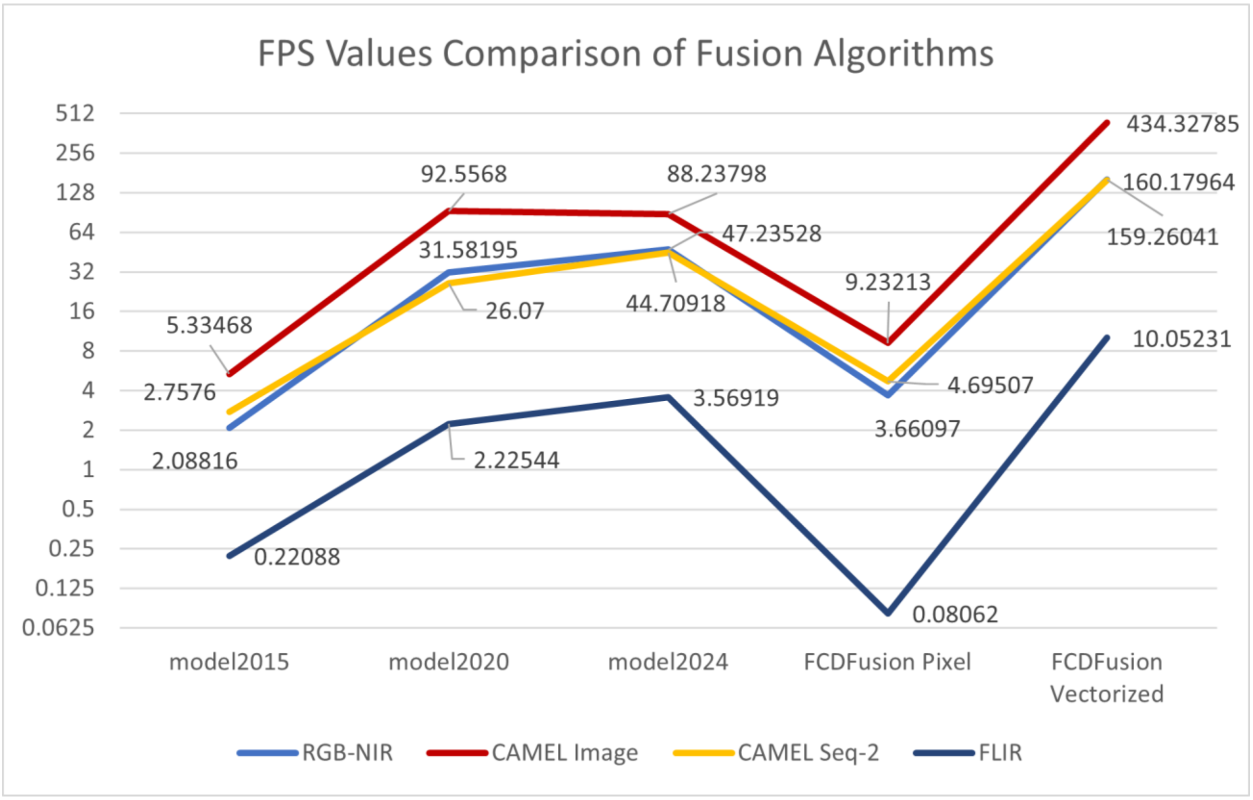


Figure 10. Comparative Analysis of Processing Speed (FPS) Across All Algorithms and Datasets. This line graph plots the average FPS for each of the five fusion algorithms across the four primary experiments. A logarithmic scale is used for the Y-axis to effectively visualize the order-of-magnitude performance differences between the methods. The results unequivocally demonstrate the transformative impact of our vectorization strategy. The proposed Vectorized FCDFusion consistently achieves the highest frame rates across all datasets, often by a significant margin, highlighting its suitability for real-time video processing. The graph also illustrates the performance hierarchy, with Model2024 and Model2020 offering moderate performance, while the non-vectorized FCDFusion (Pixel) and Model2015 are computationally prohibitive for video applications.

Fusion Algorithms	RGB-NIR	CAMEL Image	CAMEL Seq-2	FLIR Dataset
model2015	76.71	81.42	57.75	45.51
model2020	5.07	4.69	6.11	4.52
model2024	3.39	4.92	3.56	2.82
FCDFusion Pixel	43.75	47.05	33.92	124.69
FCDFusion Vectorized	**1.00**	**1.00**	**1.00**	**1.00**

Table 13: Speed Comparison- Comparative Speed-up of Vectorized FCDFusion vs other fusion models.

Fusion Algorithms	RGB-NIR	CAMEL Image	CAMEL Seq-2	FLIR Dataset
model2015	9.507	21.121	9.372	14.153
model2020	71.751	56.694	57.667	69.737
model2024	**77.895**	**58.47**	**58.813**	**75.026**
FCDFusion Pixel	77.571	55.746	57.105	74.703
FCDFusion Vectorized	77.571	55.746	57.105	74.703

Table 14: SSIM Comparison- SSIM Accuracy as measured by each Fusion Algorithm in the four experiments, given in percentage values (%).

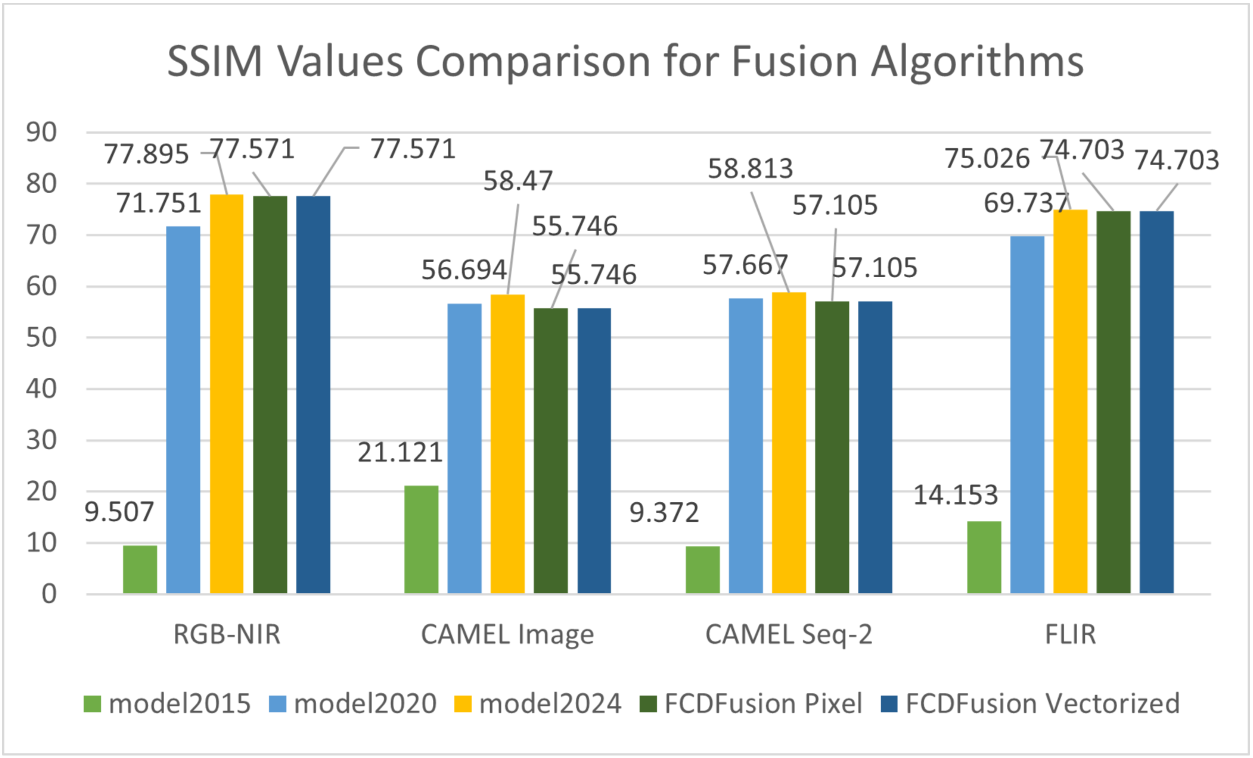


Figure 11. Comparative Analysis of Structural Similarity (SSIM) Across All Algorithms and Datasets. This bar chart plots the average SSIM accuracy for each of the five fusion algorithms across the four primary experiments. The Y-axis represents the SSIM score, where higher values indicate better structural fidelity to the source images. The chart visually demonstrates the "Fidelity vs. Contrast" trade-off; Model2024 consistently achieves the highest SSIM, establishing it as the best method for structural preservation. Our proposed Vectorized FCDFusion performs at a very high, competitive level, often statistically indistinguishable from Model2024. In contrast, the extremely low SSIM of Model2015 confirms its implementation failure, while the moderate scores of Model2020 reflect its design focus on enhancing contrast rather than preserving structural fidelity.

6.2 *Comparative Analysis through the Speed-Accuracy-Complexity Triangle*

To frame our comparative analysis, we extend the classic speed-accuracy trade-off [42] by incorporating a third parameter: complexity or computational power. This forms a Speed-Accuracy-Complexity (SAC) triangle, which is critical for evaluating real-world deployment feasibility. The conventional trade-off describes the inverse relationship between speed (the time taken to display the output, which is classified as “Time Taken (ms)” and “Frames Per Second” in our study) and accuracy (how accurate the fused output will be, with respect to the given inputs, which we have classified as “SSIM Accuracy”). We add computational power as a key constraint, (which includes the processor used to run the models, namely CPU for the Traditional Python scripts, and GPU for the Deep Learning Neural Network models) acknowledging that real-time field deployment often relies on accessible, portable CPUs rather than powerful but less available GPUs. For our ideal version of the SAC triangle, we prioritize speed on CPU hardware, accepting a potential compromise in absolute accuracy compared to slower, more resource-intensive methods. The full results are presented in Appendices 8.2 and 8.3, with key average metrics summarized in Table 11.

- *Speed:* As summarized in Table 12, Vectorized FCDFusion is the undisputed performance leader, running 2.8x to 4.9x faster than the next fastest method, Model2024. The base/detail decomposition of Model2024 and the channel-wise filtering of Model2020 are also efficient, while the per-pixel SVD calculations in Model2015 make it significantly slower.
- *Accuracy (SSIM):* According to Table 13, Model2024 consistently achieved the highest SSIM scores across all datasets (e.g., 0.750 on FLIR, 0.588 on CAMEL Seq-2), indicating its base/detail separation is highly effective at preserving structural information. Both FCDFusion models also performed well, closely matching Model2024 (e.g., 0.747 on FLIR and 0.571 on CAMEL Seq-2). In contrast, Model2015's low SSIM (0.09-0.21) is due to its placeholder reintegration step, which produces an edge-map-like output rather than a photorealistic image.
- *Image Quality & Contrast:* While SSIM measures structure, metrics like Entropy and Standard Deviation reflect information content and contrast. Here, Model2020 often excelled, producing the highest average Entropy and Std. Dev. on several datasets (Table 13). This is likely due to its use of Laplacian saliency maps, which effectively highlight active regions. Visually, this manifests as bright spots on key objects—potentially desirable for target detection but less so for general viewing.

Visually, Model2015's outputs had a significant black tint with white object borders, while Model2020's outputs showed bright, colorful spots on key objects. In contrast, Model2024 and both FCDFusion models produced consistently clear fused images, with objects from both visible and infrared inputs clearly present. As expected from the identical underlying algorithm, the two FCDFusion outputs were visually indistinguishable.

6.3 Insights and Method Selection

Our comparative analysis identifies Model2024 and Vectorized FCDFusion as the primary contenders for practical video fusion applications. Vectorized FCDFusion offers unparalleled speed, at times up to 4.9 times faster than the next-best method, Model2024, making it exceptionally suitable for high-framerate requirements or highly resource-constrained hardware. Conversely, Model2024 consistently demonstrates superior structural fidelity and edge preservation, achieving the highest average SSIM and Edge Sum scores across most datasets. Therefore, the optimal choice is application-dependent. For use cases prioritizing maximum speed with excellent quality, such as high-speed navigation, Vectorized FCDFusion is the superior choice. For applications where maximizing structural detail and achieving the highest possible accuracy is paramount, Model2024 provides the best results among the fast, practical methods. For applications prioritizing target detection, Model2020 remains a viable option due to its saliency-driven approach, which excels at enhancing the contrast of thermal targets. The performance of these models validates the viability of deploying high-quality, traditional fusion algorithms in real-time video contexts.

6.4 Research Objectives

- *RQ1: The Impact of Algorithmic Vectorization: To what extent can algorithmic vectorization improve the computational performance of a traditional fusion algorithm for real-time video, and does this speedup come at the cost of fusion quality?*

Our first research question sought to quantify the performance gains of vectorization and its effect on image quality. The results unequivocally demonstrate that our vectorization strategy provides a transformative speedup with zero loss in fidelity.

- *Zero Loss in Fusion Quality:* The core mathematics of the fusion process remain unchanged between the pixel-loop and vectorized implementations. As expected, this results in a mathematically identical output. This is empirically verified in Table 13, which shows that the SSIM scores for the original FCDFusion (Pixel) and our proposed Vectorized FCDFusion are identical across all four test cases (e.g., 74.7% on the FLIR dataset). The qualitative results in Figures 7, 8, and 9 further confirm this, showing visually indistinguishable fused outputs from both implementations.
- *Transformative and Statistically Significant Speedup:* While the quality is preserved, the computational performance is dramatically different. As detailed in Tables-11 and 12, Vectorized FCDFusion is between 40x and 157x faster than its pixel-loop counterpart. This performance gain is not merely observational but is statistically irrefutable. An independent samples t-test confirms that the mean frame rate of Vectorized FCDFusion is significantly higher than its closest and fastest competitor, Model2024, on both the FLIR dataset (10.05 FPS vs. 3.57 FPS, $p < .001$) and the CAMEL Sequence-2 dataset (159.26 FPS vs. 44.71 FPS, $p < .001$). The improvement over its own pixel-loop version is even more pronounced.

Analysis and Answer to RQ1: This speedup is achieved by replacing Python's inherently slow, sequential for-loop iterations with *NumPy*'s highly optimized, C-backend matrix operations, which can leverage the parallel processing capabilities of modern CPUs. *Therefore, in answer to RQ1, we conclude that algorithmic vectorization provides a transformative (>100x) and statistically significant improvement in computational performance without any measurable trade-off in fusion quality.*

- *RQ2: The Trade-off Between Speed, Fidelity, and Contrast: How do different families of traditional fusion algorithms (direct mapping, multi-scale decomposition, saliency-based) perform relative to one another across the key trade-off axes of processing speed, structural fidelity, and information content?*

Our second research question aimed to understand the performance trade-offs between different fusion philosophies. Our findings, reinforced with statistical analysis, unequivocally demonstrate that no single algorithm is universally superior. Instead, they occupy distinct and predictable positions in a "Speed-Fidelity-Contrast" space, allowing for a clear, data-driven selection process.

- *Fidelity vs. Contrast (Statistically Validated):* The central trade-off between structural fidelity (SSIM) and information content (Entropy) is statistically clear. On the challenging, high-resolution FLIR dataset, Model2024 achieved the highest mean SSIM (0.750). However, an independent samples t-test revealed that its advantage over our proposed Vectorized FCDFusion (0.747) is *not statistically significant (p = 0.058).* This critical finding indicates that for this automotive scenario, Vectorized FCDFusion delivers a statistically indistinguishable level of fidelity at a fraction of the computational cost.
  This is counterbalanced by their performance in information content. The t-tests confirm that Model2020 produced *a statistically significant higher mean Entropy* than both Vectorized FCDFusion and Model2024 on both the FLIR and CAMEL datasets ($p < .001$ in all cases). This validates its role as the "contrast specialist," which is visually confirmed in figures like Figure 8(e) where thermal targets "pop," at the expense of creating a less naturalistic scene compared to the high-fidelity outputs of FCDFusion (Figure 8(c)).
- *Complexity vs. Speed:* The speed hierarchy, shown in Figure 10, directly correlates with algorithmic complexity. The simple, parallelizable matrix operations of Vectorized FCDFusion are demonstrably the fastest. The decomposition-based Model2024 is second, while the more complex saliency maps of Model2020 and the SVD-based Model2015 are the most computationally intensive.

Analysis and Answer to RQ2: This demonstrates a clear, predictable, and now statistically validated trade-off. In answer to RQ2, we conclude that practitioners can and must make a deliberate choice based on their application's specific priorities:

- *For high-speed, high-fidelity applications,* choose *Vectorized FCDFusion,* which offers top-tier fidelity that is often statistically indistinguishable from the best, at a transformative speed.
- For applications where *maximum fidelity* is paramount and a *speed penalty* is acceptable, choose *Model2024*, whose superior structural preservation is statistically significant in certain scenarios.
- For tasks where *maximum target contrast* is the goal (e.g., automated detection), choose *Model2020*, which reliably produces higher-contrast outputs at the expense of both speed and structural fidelity.

- *RQ3: Viability for Real-Time CPU-Based Video Fusion: Are optimized traditional fusion algorithms viable for sustained, real-time (>30 FPS) video processing on standard, CPU-only hardware, and how does their performance generalize across different types of dynamic scenes?*

Our final research question assessed the practical viability of these algorithms for sustained, real-time video processing on standard, CPU-only hardware. The video benchmark results from our desktop-based experiments and our live hardware deployment provide a definitive, two-part answer.

Desktop Benchmark Performance: First, our benchmarks on a desktop AMD Ryzen 5 CPU establish a clear performance hierarchy. The average frame rates achieved on the CAMEL (Seq-2) and FLIR datasets were:

- Vectorized FCDFusion: 159.26 FPS (CAMEL), 10.05 FPS (FLIR)
- Model2024: 44.71 FPS (CAMEL), 3.57 FPS (FLIR)
- Model2020: 26.07 FPS (CAMEL), 2.23 FPS (FLIR)

This data shows both Vectorized FCDFusion and Model2024 algorithms comfortably exceed the >30 FPS "real-time" threshold in the lower-resolution CAMEL scenario. The performance of all algorithms degrades on the much higher-resolution FLIR dataset, but Vectorized FCDFusion remains the only method to achieve a usable interactive frame rate (>10 FPS).

Live Hardware Deployment on Raspberry Pi 5: To provide the ultimate validation of real-world viability, we deployed our full pipeline on a Raspberry Pi 5, a resource-constrained, ARM-based single-board computer. In this live demonstration, processing two simultaneous 640x480 video streams, Vectorized FCDFusion consistently achieved real-time performance, delivering a stable fused video feed with upwards of 60FPS consistently. This successful deployment on low-power, non-x86 hardware is the most powerful confirmation of the algorithm's efficiency and portability.

Analysis and Answer to RQ3: The combination of our benchmark data and live hardware prototype provides a conclusive answer to RQ3. The desktop results demonstrate that Vectorized FCDFusion possesses a significant performance advantage, suggesting good generalization across different scene types. The successful deployment on the Raspberry Pi 5 provides the definitive proof. *Therefore, we conclude that optimized traditional algorithms—specifically Vectorized FCDFusion—are not only viable but are exceptionally well-suited and robust for real-time video fusion on the diverse range of CPU-only hardware found in real-world applications.*

6.5 Limitations

While this study provides a thorough comparative analysis of four traditional fusion algorithms, it has some limitations which relate to the scope of the comparison, the inherent properties of the algorithms, and the experimental conditions.

A primary limitation is the exclusive focus on traditional, non-learning-based fusion algorithms. This choice was made deliberately to investigate methods suitable for real-time deployment on standard CPU hardware without needing extensive training data or specialized GPUs. Consequently, a direct performance benchmark against state-of-the-art deep learning-based image fusion was not conducted. While our proposed Vectorized FCDFusion is highly efficient within its class, its relative performance in pure image quality against leading neural network models (e.g., U2Fusion, SwinFusion) remains an open question for future investigation.

Our analysis also revealed inherent constraints within the evaluated algorithms. The proposed Vectorized FCDFusion algorithm, while fast, exhibits sensitivity to scenes with uniformly high thermal intensity. In such cases, its direct intensity scaling can lead to oversaturation, suggesting a need for an adaptive or localized scaling strategy.

Furthermore, the poor performance of Model2015 is a direct consequence of the placeholder gradient reintegration method used in our implementation. Our results highlight that the theoretical benefits of gradient-domain fusion are critically dependent on a robust solution to this step. Finally, the experimental validation was conducted under specific conditions that may not capture the full range of real-world challenges. The datasets used (RGB-NIR, TNO, Landsat, CAMEL, and FLIR) consist of well-registered image pairs from relatively clear atmospheric conditions. The robustness of these algorithms in adverse environments—such as heavy fog, rain, or with significant sensor misalignment and motion blur—was not evaluated. Thus, while our findings on speed and quality are valid within the tested environments, the generalizability of these results to more challenging operational scenarios is yet to be determined.

7. Conclusion & Future Work

The central contribution of this work is demonstrating that an optimized implementation can transform a traditional image fusion method into a real-time capable solution. By developing a vectorized implementation of the FCDFusion algorithm, we achieved a speedup of up to 125x over its original pixel-loop form, enabling video-rate performance (>30 FPS) on standard CPU hardware without degrading fusion quality. We also presented a comparative analysis of four distinct families of traditional fusion algorithms on a diverse suite of datasets.

Our contributions include a reproducible framework for evaluating these methods across quantitative metrics, illuminating the critical trade-offs between processing speed and fusion quality, and our optimized method: *Vectorized FCDFusion*. Our key findings demonstrate that no single algorithm is universally optimal. The base/detail decomposition method (Model2024) consistently yielded the highest structural similarity (SSIM), making it ideal for high-fidelity applications. Conversely, the hybrid filtering method (Model2020) excelled at producing high-contrast, high-entropy images. The vectorized FCDFusion stands as the fastest method, achieving a speedup of over 120x compared to its original pixel loop form while maintaining competitive quality, making it a viable candidate for real-time applications. Ultimately, this work provides a data-driven analysis to guide practitioners in selecting the most appropriate fusion algorithm for their specific constraints, whether speed for embedded deployment or maximum detail for offline analysis.

Building on the findings of this study, several promising directions for future research emerge. Our immediate priority is to address the key limitation of the gradient-domain method (Model2015) by implementing a full Poisson-based or Fourier-based gradient field reintegration. This upgrade will replace the current placeholder mechanism, enabling a fair and photorealistic evaluation of its fusion performance. Future work will also explore extending image and video fusion to other electromagnetic spectrum bands, such as ultraviolet (UV), microwave, and radio frequencies. This multi-spectral integration could enable new applications in search and rescue (e.g., microwave for through-wall detection), biomedical diagnostics (UV for surface anomalies), and autonomous navigation (radio-based radar). While real-time fusion with X-ray and gamma-ray data is infeasible for portable systems due to hardware and safety constraints, simulation-based or controlled-environment integration could be explored for medical and astronomical applications. Finally, a key enhancement is incorporating temporal consistency. Future iterations will investigate lightweight temporal filtering or recurrent architectures that utilize historical frame information to minimize inter-frame flicker, ensuring smoother visual transitions and improved perceptual quality in continuous video streams.

Conflict of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work in this paper.

Research Involving Human and /or Animals

Not Applicable.

Acknowledgments

Data Availability

Multiple publicly accessible datasets have been used, which are referred to in this paper.

Funding

No open-access funding is available.

Informed Consent

Not Applicable

References

[1] "FREE - FLIR Thermal Dataset for Algorithm Training | OEM.FLIR.com." Flir.com, 2025, oem.flir.com/solutions/automotive/adas-dataset-form.

[2] E. Gebhardt and M. Wolf, "CAMEL Dataset for Visual and Thermal Infrared Multiple Object Detection and Tracking," IEEEInternational Conference on Advanced Video and Signal-based Surveillance (AVSS), 2018.

[3] P. Saha, B. A Mudassar, and S. Mukhopadhyay, "Adaptive Control of Camera Modality with Deep Neural Network-Based Feedback for Efficient Object Tracking," IEEE International Conference on Advanced Video and Signal-based Surveillance (AVSS), 2018.

[4] Li, Hesong, and Ying Fu. "FCDFusion: A Fast, Low Color Deviation Method for Fusing Visible and Infrared Image Pairs." Computational Visual Media, vol. 11, no. 1, 1 Feb. 2025, pp. 195–211, https://doi.org/10.26599/cvm.2025.9450330.

[5] Panda, Manoj Kumar, et al. "A Weight Induced Contrast Map for Infrared and Visible Image Fusion." Computers and Electrical Engineering, vol. 117, 24 Apr. 2024, p. 109256, www.sciencedirect.com/science/article/abs/pii/S0045790624001848, https://doi.org/10.1016/j.compeleceng.2024.109256.

[6] Zhang, Yongxin, et al. "Infrared and Visible Image Fusion with Hybrid Image Filtering." Mathematical Problems in Engineering, vol. 2020, 29 July 2020, pp. 1–17, https://doi.org/10.1155/2020/1757214.

[7] Connah, David, et al. "Spectral Edge: Gradient-Preserving Spectral Mapping for Image Fusion." Journal of the Optical Society of America A, vol. 32, no. 12, 23 Nov. 2015, p. 2384, https://doi.org/10.1364/josaa.32.002384.

[8] Hu, Kun, et al. "SFDFusion: An Efficient Spatial-Frequency Domain Fusion Network for Infrared and Visible Image Fusion." Frontiers in Artificial Intelligence and Applications, 16 Oct. 2024, https://doi.org/10.3233/faia240524.

[9] Zheng, Naishan, et al. "Probing Synergistic High-Order Interaction in Infrared and Visible Image Fusion." 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), vol. 35, 16 June 2024, pp. 26374–26385, https://doi.org/10.1109/cvpr52733.2024.02492.

[10] Chen, Zhaoyu, et al. "FECFusion: Infrared and Visible Image Fusion Network Based on Fast Edge Convolution." Mathematical Biosciences and Engineering, vol. 20, no. 9, 2023, pp. 16060–16082, https://doi.org/10.3934/mbe.2023717.

[11] Li, Hui, and Xiao-Jun Wu. "CrossFuse: A Novel Cross Attention Mechanism Based Infrared and Visible Image Fusion Approach." Information Fusion, vol. 103, 1 Mar. 2024, pp. 102147–102147, https://doi.org/10.1016/j.inffus.2023.102147.

[12] Xie, Xinyu, et al. "FusionMamba: Dynamic Feature Enhancement for Multimodal Image Fusion with Mamba." Visual Intelligence, vol. 2, no. 1, 31 Dec. 2024, https://doi.org/10.1007/s44267-024-00072-9.

[13] Wang, Di, et al. "An Interactively Reinforced Paradigm for Joint Infrared-Visible Image Fusion and Saliency Object Detection." Information Fusion, vol. 98, Oct. 2023, p. 101828, https://doi.org/10.1016/j.inffus.2023.101828.

[14] Liu, Zhu, et al. "PAIF: Perception-Aware Infrared-Visible Image Fusion for Attack-Tolerant Semantic Segmentation." ArXiv (Cornell University), 1 Jan. 2023, https://doi.org/10.48550/arxiv.2308.03979.

[15] Zhao, Wenda, et al. "MetaFusion: Infrared and Visible Image Fusion via Meta-Feature Embedding from Object Detection." 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 1 June 2023, https://doi.org/10.1109/cvpr52729.2023.01341.

[16] Yang, Guang, et al. “A Multi-Scale Information Integration Framework for Infrared and Visible Image Fusion.” Neurocomputing, vol. 600, 28 June 2024, pp. 128116–128116, https://doi.org/10.1016/j.neucom.2024.128116.

[17] Li, Jiawei, et al. “Learning a Graph Neural Network with Cross Modality Interaction for Image Fusion.” ArXiv (Cornell University), 1 Jan. 2023, https://doi.org/10.48550/arxiv.2308.03256.

[18] Liu, Jinyuan, et al. “Infrared and Visible Image Fusion: From Data Compatibility to Task Adaption.” ArXiv (Cornell University), 18 Jan. 2025, https://doi.org/10.48550/arxiv.2501.10761.

[19] Li, Jiawei, et al. “A2RNet: Adversarial Attack Resilient Network for Robust Infrared and Visible Image Fusion.” ArXiv (Cornell University), 13 Dec. 2024, https://doi.org/10.48550/arxiv.2412.09954.

[20] Sun, Yiming, et al. “Dynamic Brightness Adaptation for Robust Multi-Modal Image Fusion.” ArXiv (Cornell University), 26 July 2024, pp. 1317–1325, https://doi.org/10.24963/ijcai.2024/146.

[21] Xu, Jian, and Xin He. “DAF-Net: A Dual-Branch Feature Decomposition Fusion Network with Domain Adaptive for Infrared and Visible Image Fusion.” ArXiv (Cornell University), 17 Sept. 2024, https://doi.org/10.48550/arxiv.2409.11642.

[22] Wang, Yuhao, et al. “Infrared and Visible Image Fusion with Language-Driven Loss in CLIP Embedding Space.” ArXiv (Cornell University), 25 Feb. 2024, https://doi.org/10.48550/arxiv.2402.16267.

[23] Wu, Yuhui, et al. “Breaking Free from Fusion Rule: A Fully Semantic-Driven Infrared and Visible Image Fusion.” IEEE Signal Processing Letters, vol. 30, 1 Jan. 2023, pp. 418–422, https://doi.org/10.1109/lsp.2023.3266980.

[24] Anjali Malviya, and S G Bhirud. “Visual Infrared Video Fusion for Night Vision Using Background Estimation.” ArXiv (Cornell University), 1 Jan. 2010, https://doi.org/10.48550/arxiv.1004.4459.

[25] Wang, Qishun, et al. “High Performance RGB-Thermal Video Object Detection via Hybrid Fusion with Progressive Interaction and Temporal-Modal Difference.” Information Fusion, 1 Sept. 2024, pp. 102665–102665, https://doi.org/10.1016/j.inffus.2024.102665.

[26] Yang, Lizhi, et al. “Drone Object Detection Using RGB/IR Fusion.” Electronic Imaging, vol. 34, no. 14, 16 Jan. 2022, pp. 179–1179–6, https://doi.org/10.2352/ei.2022.34.14.coimg-179.

[27] Spremolla, Ignacio Rocco, et al. RGB-D and Thermal Sensor Fusion. 1 Jan. 2016, pp. 610–617, https://doi.org/10.5220/0005717706100617.

[28] Ben-Shoushan, Ravit, and Anna Brook. “Fused Thermal and RGB Imagery for Robust Detection and Classification of Dynamic Objects in Mixed Datasets via Pre-Trained High-Level CNN.” Remote Sensing, vol. 15, no. 3, 26 Jan. 2023, p. 723, https://doi.org/10.3390/rs15030723.

[29] Lee, Chris H., et al. “Coordinating High-Resolution Hyperspectral and RGB Video Acquisition of Dynamic Natural Water Scenes.” Journal of Applied Remote Sensing, vol. 19, no. 02, 19 May 2025, https://doi.org/10.1117/1.jrs.19.024507.

[30] Toet, Alexander. “TNO Image Fusion Dataset.” Figshare, 15 Oct. 2022, https://doi.org/10.6084/u002Fm9.figshare.1008029.v2.

[31] Toet, Alexander. “The TNO Multiband Image Data Collection.” Data in Brief, vol. 15, 1 Dec. 2017, pp. 249–251, www.sciencedirect.com/science/article/pii/S2352340917304699, https://doi.org/10.1016/j.dib.2017.09.038.

[32] USGS - U.S. Geological Survey. “EarthExplorer.” Usgs.gov, 2025, Landsat Dataset. https://earthexplorer.usgs.gov/

[33] “RGB-NIR Scene Dataset.” EPFL, 2018, RGB-NIR Scene Dataset. https://www.epfl.ch/labs/ivrl/research/downloads/rgb-nir-scene-dataset/

[34] Xingchenzhang. “GitHub - Xingchenzhang/VIFB: Visible and Infrared Image Fusion Benchmark.” GitHub, 2020, VIFB (GitHub). https://github.com/xingchenzhang/VIFB

[35] Shannon, C. E. “A Mathematical Theory of Communication.” Bell System Technical Journal, vol. 27, no. 3, 1948, pp. 379–423, https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

[36] Jagalingam, P., and Arkal Vittal Hegde. “A Review of Quality Metrics for Fused Image.” Aquatic Procedia, vol. 4, 2015, pp. 133–142, https://doi.org/10.1016/j.aqpro.2015.02.019.

[37] Wang, Zhou, et al. “Image Quality Assessment: From Error Visibility to Structural Similarity.” IEEE Transactions on Image Processing, vol. 13, no. 4, Apr. 2004, pp. 600–612, https://doi.org/10.1109/tip.2003.819861.

[38] Olufunke Rebecca Vincent, and Olusegun Folorunso. “A Descriptive Algorithm for Sobel Image Edge Detection.” Informing Science and IT Education Conference, 1 Jan. 2009, https://doi.org/10.28945/3351.

[39] Lee, Gregory, et al. “PyWavelets: A Python Package for Wavelet Analysis.” Journal of Open Source Software, vol. 4, no. 36, 12 Apr. 2019, p. 1237, https://doi.org/10.21105/joss.01237.

[40] Matzner, Shari, et al. “Two-Dimensional Thermal Video Analysis of Offshore Bird and Bat Flight.” Ecological Informatics, vol. 30, Nov. 2015, pp. 20–28, https://doi.org/10.1016/j.ecoinf.2015.09.001.

[41] GitHub repository (https://github.com/rayeedaabir/AIRC-Fusion)

[42] Bogacz, Rafal. “Speed-Accuracy Tradeoff.” Encyclopedia of Computational Neuroscience, 1 Jan. 2022, pp. 3225–3228, https://doi.org/10.1007/978-1-0716-1006-0_319.

[43] Zhang, Yongsheng, Chen Tu, Kun Gao, and Liang Wang, "Multisensor Information Fusion: Future of Environmental Perception in Intelligent Vehicles", Journal of Intelligent and Connected Vehicles, Volume 7, Issue 3, 2024, Pages 163-176, https://doi.org/10.26599/JICV.2023.9210049.

[44] Akilan, Thangarajah, and Hrishikesh Vachhani, "Low-light Pedestrian Detection in Visible and Infrared Image Feeds: Issues and Challenges", arXiv preprint arXiv:2311.08557, 2023, https://arxiv.org/abs/2311.08557.

[45] Khabiri, Roghayeh, Jahangiry, Leila, Birgani, Hosna Rashidi, Sadeghi-bazargani, Homayoun, Interventions for Increasing Pedestrian Visibility to Prevent Injury and Death: A Systematic Review, Health & Social Care in the Community, 2025, 2958743, 13 pages, 2025. https://doi.org/10.1155/hsc/2958743

[46] Tabitha S. Combs, Laura S. Sandt, Michael P. Clamann, Noreen C. McDonald, Automated Vehicles and Pedestrian Safety: Exploring the Promise and Limits of Pedestrian Detection, American Journal of Preventive Medicine, Volume 56, Issue 1, 2019, Pages 1-7, ISSN 0749-3797, https://doi.org/10.1016/j.amepre.2018.06.024

8. Appendix

8.1. Pseudocodes:

The fusion pipeline generated for the comparative analysis involved nine Python (.py) files being run on VSCode. The pseudocodes, as well as the Python files generated for the fusion pipeline, are given below:

1. Pseudocode for main.py:

Algorithm 1: Main Fusion Pipeline Orchestration

Objective: To automate the fusion of registered visible and thermal image sequences and generate fused video outputs for evaluation.

Inputs:

- DatasetType: The type of dataset being processed (e.g., "FLIR", "CAMEL"), which dictates the image pairing strategy.
- VisibleImageSet: A directory of visible spectrum images.
- ThermalImageSet: A directory of thermal spectrum images.
- Config: A set of configuration parameters including input/output paths, fusion method, and hyperparameters.
- CLI_Args: Optional command-line arguments to override Config.

Outputs:

- FusedImageSet: A directory of fused images.
- FusedVideo: A video compiled from FusedImageSet.
- ComparisonVideo: A side-by-side video of visible, thermal, and fused frames.

```
Procedure:
 1. BEGIN
 2. // Phase 1: Initialization and Configuration
 3. params ← InitializeParameters(Config, CLI_Args)
Comment: Load base configuration for the specified DatasetType. Override with any provided command-line
arguments for fusion method, hyperparameters, processing limits, etc.
 4. fusion_processor ← Instantiate ImageFusionProcessor with params.fusion_method and its specific
    hyperparameters.
 5. dataset_processor ← Instantiate DatasetProcessor with fusion_processor and params.num_workers.
 6. // Phase 2: Parallel Image Fusion
 7. image_pairs ← GetPairedImageList(VisibleImageSet, ThermalImageSet, DatasetType)
Comment: Identify corresponding visible-thermal image pairs. Use JSON mapping for FLIR-type datasets or
synchronized text files for CAMEL-type datasets.
 8. results ← dataset_processor.ProcessImagePairsInParallel(image_pairs, params.limit)
Comment: Each pair is processed by the fusion_processor on a separate worker thread. The function returns
metrics and an ordered list of output file paths.
 9. IF results.processed_count = 0 THEN: **Log** "No images were processed." and **TERMINATE**.
10. END IF
11. // Phase 3: Video Synthesis
12. ordered_fused_files ← results.ordered_output_paths
13. AssembleFusedVideo(ordered_fused_files, params.output_video_path, params.fps)
Comment: Creates a single video stream from the fused images.
14. AssembleSideBySideVideo(image_pairs, ordered_fused_files, params.comparison_video_path, params.
    fps)
Comment: Creates a three-panel video (Visible | Thermal | Fused) for qualitative comparison, ensuring frame
synchronization.
15. Log "Pipeline processing complete."
16. END
```

2. *Pseudocode and Tables for config.py:*

The pipeline is designed to be highly configurable, supporting multiple datasets and fusion algorithms. Key parameters are managed through a centralized configuration module. Table 1 outlines the structural differences between the supported dataset schemas (FLIR and CAMEL). Table 2 lists the default hyperparameters for the fusion models used in our experiments. The system dynamically loads the appropriate configuration based on the selected dataset, as described in Algorithm 2 in Table 3.

Table 1: Dataset Configuration Schemas

Parameter	FLIR Setting	CAMEL Setting
Pairing Mechanism	JSON mapping file	Synchronized text files (.txt)
Visible Data Subdirectory	video_rgb_test/data	vis
Thermal Data Subdirectory	video_thermal_test/data	IR
Image Naming	Generic pattern (*.jpg)	Formatted sequence ({:06d}.jpg)
Metadata File	N/A	Environmental info (-info.txt)
Output Directory	..._output_flir	..._output_camel

Table 2: Default Fusion Model Hyperparameters

Model	Parameter	Symbol	Default Value
Model2024	Gaussian Kernel Size	k_{gauss}	15
	Base Fusion Weight	ω_{base}	0.5
	Detail Fusion Method	-	max_abs

Model2020	Decomposition Kernel Size	k_{decomp}	11
	Decomposition Sigma	Σ_{decomp}	5
	Base Laplacian Kernel Size	L_{size}	3
	Base Gaussian Kernel Size	G_{size}	11
	Base Gaussian Sigma	G_{sigma}	5
	Guided Filter Radius	r_{guided}	7
	Guided Filter Epsilon	ϵ_{guided}	0.01
	Detail Median Kernel Size	k_{median}	3
	Detail Laplacian Kernel Size	L_{median}	3
	Detail Weight Kernel Size	k_{weight}	11
	Detail Weight Sigma	Σ_{weight}	5
Model2015	Gradient Kernel Size	k_{grad}	3
FCDFusion	No tunable parameters	-	-

Algorithm 2: Dynamic Configuration Loading

Objective: To initialize the system's global parameters based on the selected dataset type.

Inputs:

- DatasetType: A string identifying the dataset (e.g., "FLIR", "CAMEL").
- BasePathOverride (Optional): A custom path to the root of the dataset.

Output:

- A globally accessible SystemConfiguration containing all necessary paths and hyperparameters.

Procedure:

```
1.  BEGIN
2.  base_path ← BasePathOverride OR default path for DatasetType.
3.  SWITCH DatasetType DO
4.  **CASE** "FLIR":
5.  `SystemConfiguration` ← **Load** path structures and file names based on the FLIR schema (from
    Table 1) using `base_path`.
6.  `SystemConfiguration.pairing_strategy` ← `JSON_MAP`.
7.  **CASE** "CAMEL":
8.  `SystemConfiguration` ← **Load** path structures and file names based on the CAMEL schema
    (from Table 1) using `base_path`.
9.  `SystemConfiguration.pairing_strategy` ← `SYNCED_TXT_FILES`.
10. **DEFAULT**:
11. **Log** "Unsupported dataset type, defaulting to FLIR."
12. **CALL** `SetActiveConfiguration`("FLIR", `BasePathOverride`).
13. **RETURN**.
14. END SWITCH
15. SystemConfiguration ← Load default hyperparameters for all fusion models (from Table 2).
16. Log "Configuration loaded for " + DatasetType.
17. END
```

3. *Pseudocode for image_fusion_processor.py:*

The core of our pipeline is the ImageFusionProcessor module, which encapsulates four distinct fusion algorithms. Upon initialization, it receives a pair of visible (I_{vis}) and thermal (I_{th}) images. A common preprocessing step first ensures spatial alignment and consistent color channels. Then, based on the selected configuration, one of the fusion algorithms (described in Algorithms 3 to 5 in Tables 4 to 6) is executed to produce the final fused image, I_{fused}.

Algorithm 3: Preprocessing and Fusion Dispatch

Objective: To prepare input images and delegate to the selected fusion algorithm.

Inputs:

- Visible image, I_{vis}
- Thermal image, I_{th}
- FusionMethod: A string identifying the algorithm (e.g., "Model2024", "FCD").
- TargetResolution (Optional): A target (width, height) tuple.

Output:

- A fused image, I_{fused}.

Procedure:

```
1.  BEGIN
2.  // Spatial and Color Alignment
3.  IF dimensions of $I_{vis}$ ≠ dimensions of $I_{th}$ THEN:
4.  $I_{th}$ ← **Resize**($I_{th}$, dimensions of $I_{vis}$).
5.  END IF
6.  $I_{vis}$ ← ConvertTo3Channel ($I_{vis}$).
7.  $I_{th}$ ← ConvertTo3Channel ($I_{th}$).
8.  IF TargetResolution is specified THEN
9.  $I_{vis}$ ← **Resize**($I_{vis}$, `TargetResolution`).
10. $I_{th}$ ← **Resize**($I_{th}$, `TargetResolution`).
11. END IF
12. // Fusion Method Dispatch
13. SWITCH FusionMethod DO:
14. **CASE** "Model2024": $I_{fused}$ ← **Model2024_Fusion**($I_{vis}$, $I_{th}$).
15. **CASE** "FCD": $I_{fused}$ ← **FCD_Fusion**($I_{vis}$, $I_{th}$).
16. **CASE** "Model2020": $I_{fused}$ ← **Model2020_Fusion**($I_{vis}$, $I_{th}$).
17. **CASE** "Model2015": $I_{fused}$ ← **Model2015_Fusion**($I_{vis}$, $I_{th}$).
18. END SWITCH
19. RETURN $I_{fused}$
20. END
```

Algorithm 4: Model2024 Fusion

Objective: To fuse images via a two-scale decomposition (base and detail).

Inputs:

- Visible image, I_{vis}
- Thermal image, I_{th}
- Hyperparameters: Gaussian kernel size k_{gauss}, base weight ω_{base}, detail method DetailMethod.

Output:

- A fused image, I_{fused}

Procedure:

```
1.  BEGIN
2.  // 1. Two-Scale Decomposition
3.  $B_{vis}$ ← GaussianBlur ($I_{vis}$, $k_{gauss}$) // Base layer of visible
4.  $B_{th}$ ← GaussianBlur ($I_{th}$, $k_{gauss}$) // Base layer of thermal
5.  $D_{vis} \leftarrow I_{vis} - B_{vis}$ // Detail layer of visible
6.  $D_{th} \leftarrow I_{th} - B_{th}$ // Detail layer of thermal
7.  // 2. Base Layer Fusion
8.  $B_{fused} \leftarrow \omega_{base} \cdot B_{vis} + (1 - \omega_{base}) \cdot B_{th}$
9.  // 3. Detail Layer Fusion
10. SWITCH DetailMethod DO:
11. **CASE** "max_abs":
12. $D_{fused}$ ← **ChoosePixelwise**($D_{vis}$, $D_{th}$) where $|D_{vis}| \ge |D_{th}|$.
```

```
13. **CASE** "weighted":
14. $W_{vis}$ ← $|D_{vis}| / (|D_{vis}| + |D_{th}| + \epsilon)$
15. $D_{fused}$ ← $W_{vis} \cdot D_{vis} + (1-W_{vis}) \cdot D_{th}$
16. **CASE** "adaptive":
17. $\sigma^2_{vis}$ ← **LocalVariance**($D_{vis}$)
18. $\sigma^2_{th}$ ← **LocalVariance**($D_{th}$)
19. $W_{vis}$ ← $\sigma^2_{vis} / (\sigma^2_{vis} + \sigma^2_{th} + \epsilon)$
20. $D_{fused}$ ← $W_{vis} \cdot D_{vis} + (1-W_{vis}) \cdot D_{th}$
21. END SWITCH
22. // 4. Image Reconstruction
23. $I_{fused} \leftarrow B_{fused} + D_{fused}$
24. RETURN ClipAndScale($I_{fused}$) // Scale to [0, 255]
25. END
```

Algorithm 5: FCD Fusion (Fast Low-Color Deviation)

Objective: To enhance a color image with thermal intensity information.

Inputs:

- Visible image, I_{vis} (3-channel BGR)
- Thermal image, I_{th} (single-channel intensity)

Output:

- A fused image, I_{fused}.

Procedure:

```
1.  BEGIN
2.  // 1. Intensity Component Extraction
3.  $V_{max} \leftarrow$ PixelwiseMax ($I_{vis,B}$ , $I_{vis,G}$ , $I_{vis,R}$)
4.  $I_{intensity} \leftarrow I_{th}$ // Assumes single-channel thermal input
5.  // 2. Calculate Adaptive Scaling Factor
6.  $\alpha \leftarrow (I_{intensity} / 255)^2$
7.  $k \leftarrow \alpha \cdot (V_{max} + 255) / (2 \cdot V_{max} + \epsilon) + 0.5$
Comment: k is a per-pixel gain map.
8.  // 3. Fusion by Scaling
9.  $I_{fused} \leftarrow I_{vis} \odot k$ // Element-wise multiplication
10. RETURN ClipAndScale($I_{fused}$)
11. END
```

4. Pseudocode for dataset_processor.py: To facilitate large-scale experiments, a DatasetProcessor module was developed to automate the processing of entire image and video datasets. This module is responsible for identifying corresponding visible-thermal image pairs, distributing the fusion tasks across multiple processing threads, and aggregating the results. The core workflow is detailed in Algorithms 6 and 7 in Tables 7 and 8. They support various dataset structures by using different pairing strategies and can process both static image directories and video files.

Algorithm 6: Batch Fusion and Evaluation

Objective: To process a dataset of visible-thermal pairs, execute the fusion, and aggregate performance metrics.

Inputs:

- DataSource: A pair of image directories or video files.
- PairingStrategy: Method to identify corresponding frames ("JSON", "SYNCED_TXT", "GLOB").
- FusionMethod: The selected fusion algorithm (e.g., "Model2024").

- OutputConfig: Path for saving fused results and metrics.
- Params: Processing parameters (e.g., limit, num_workers).

Outputs:

- AggregatedMetrics: A summary of performance across the dataset.
- FusedDataSet: A collection of fused images or a fused video.

Procedure:

```
1. BEGIN
2. // 1. Image Pair Discovery
3. image_pairs ← IdentifyImagePairs (DataSource, PairingStrategy)
```

Comment: This function encapsulates the logic for JSON, TXT, or glob-based pairing to return an ordered list of (VisiblePath, ThermalPath) tuples.

```
4. Apply Params.limit to image_pairs if specified.
5. IF image_pairs is empty THEN
6. **Log** "No image pairs found." and **TERMINATE**.
7. END IF
8. // 2. Parallel Processing
9. all_results ← an empty list.
10. Initialize a thread pool with Params.num_workers workers.
11. FOR EACH pair in image_pairs IN PARALLEL DO
12. `individual_result` ← **ProcessAndEvaluatePair**(`pair`, `FusionMethod`, `OutputConfig`)
13. **Atomically** add `individual_result` to `all_results`.
14. END FOR
15. Wait for all threads to complete.
16. // 3. Metrics Aggregation and Reporting
17. IF all_results is empty THEN
18. **Log** "Processing failed for all pairs." and **TERMINATE**.
19. END IF
20. AggregatedMetrics ← AggregateMetrics (all_results)
```

Comment: Calculate mean, min, and max for metrics like processing time, Entropy, SSIM, etc., across the dataset.

```
21. Save AggregatedMetrics to a file in OutputConfig.path.
22. RETURN AggregatedMetrics.
23. END
```

Algorithm 7: Sub-Procedure: ProcessAndEvaluatePair

Objective: To fuse a single image pair, save the result, and calculate its quality metrics.

Inputs:

- image_pair: A tuple containing (VisiblePath, ThermalPath).
- FusionMethod: The selected fusion algorithm.
- OutputConfig: Path for saving the fused image.

Output:

- A dictionary, result_metrics, containing performance and quality scores for the pair.

Procedure:

```
1. BEGIN
2. $I_{vis}$ ← LoadImage (image_pair.VisiblePath)
3. $I_{th}$ ← LoadImage (image_pair.ThermalPath)
4. start_time ← CurrentTimestamp()
5. $I_{fused}$ ← DispatchToFusionAlgorithm ($I_{vis}$, $I_{th}$, FusionMethod) (See Algorithm 3)
6. proc_time ← CurrentTimestamp() - start_time
7. SaveImage ($I_{fused}$, OutputConfig.path)
8. quality_metrics ← CalculateAllMetrics ($I_{fused}$, $I_{vis}$, $I_{th}$)
```

```
Comment: This calls functions to compute Entropy, SSIM, Mutual Information, etc.
    9.  result_metrics ← quality_metrics
    10. result_metrics.Add ('ProcessingTime', proc_time)
    11. RETURN result_metrics.
    12. END
```

5. *Pseudocode for benchmark_utils.py:* We initially started with the model2024 fusion algorithm, but as we opted for a more comprehensive evaluation with the other fusion models (model2015, model2020, FCDFusion (Pixel), and FCDFusion (Vectorized), we kept the evaluation metrics consistent throughout all models. To determine the optimal hyperparameters for the Model2024 fusion algorithm, we implemented a systematic grid search benchmark. This process iterates through a predefined space of parameter combinations, applying each to a sample dataset. For each combination, a suite of quantitative performance metrics is computed. The results are then aggregated and ranked to identify the most effective parameter set. The methodology for this benchmark is detailed in Algorithm 8 in Table 9.

Algorithm 8: Hyperparameter Grid Search and Evaluation

Objective: To systematically evaluate a range of hyperparameter combinations for a given fusion model and identify the optimal set based on quantitative metrics.

Inputs:

- DataSource: A pair of image directories or a single image pair.
- FusionModel: The fusion model to be benchmarked (e.g., "Model2024").
- HyperparameterSpace: A definition of parameter ranges to test (e.g., kernel sizes, weights).
- OutputConfig: A base path for storing benchmark results and sample fused images.

Output:

- A ranked table (BenchmarkResults) of hyperparameter combinations and their corresponding performance scores.

Procedure:

```
    1.  BEGIN
    2.  results_collection ← an empty list.
    3.  parameter_combinations ← GenerateCartesianProduct (HyperparameterSpace).
Example: For kernel_sizes = [7,11], weights = [0.3,0.5], this generates [(7, 0.3), (7, 0.5), (11, 0.3), (11, 0.5)].
    4.  FOR EACH current_params in parameter_combinations DO
    5.  // a. Execute Fusion
    6.  `metrics` ← **ExecuteBatchFusion**(`DataSource`, `FusionModel`, `current_params`).
Comment: This internally calls a function similar to **Algorithm 7** (Batch Fusion and Evaluation) using the `current_params`. For directory sources, it processes a limited subset (e.g., N=10) for efficiency and returns averaged metrics. For a single image, it returns direct metrics.
    7.  // b. Record Results
    8.  **IF** `metrics` were successfully generated **THEN**
    9.  `result_entry` ← `current_params`.
    10. `result_entry`.**AddAll**(`metrics`).
    11. **Add** `result_entry` to `results_collection`.
    12. **OptionallySaveSampleImages**(`DataSource`, `FusionModel`, `current_params`, `OutputConfig`).
    13. **END IF**
    14. END FOR
    15. // 3. Rank and Report
    16. IF results_collection is empty THEN
    17. **Log** "No benchmark results were generated." and **TERMINATE**.
    18. END IF
    19. BenchmarkResults ← ConvertToTable (results_collection).
    20. primary_metric ← SelectPrimaryMetric (e.g., SSIM, Mutual Information).
```

```
21. Sort BenchmarkResults in descending order based on primary_metric.
22. Save BenchmarkResults to a CSV file in OutputConfig.path.
23. RETURN BenchmarkResults.
24. END
```

6. *Representation for evaluation_metrics.py:* To quantitatively evaluate the performance of our fusion algorithms, we employed a comprehensive set of seven established image quality metrics. Each metric assesses a different aspect of the fused image, from information content and contrast to structural fidelity. The metrics are calculated for the fused image (I_F) and its corresponding visible (I_V) and thermal (I_T) source images. The following metrics were used:

a) *Shannon Entropy (EN):* Measures the richness of information in the fused image. A higher entropy value indicates more information and detail. It is defined as:

$$EN(I) = -\sum_{i=0}^{L-1} p(l)log_2 p(l) \tag{1}$$

where L is the number of gray levels (256 for an 8-bit image) and $p(l)$ is the probability of the gray level l occurring in the image I.

b) *Standard Deviation (SD):* Reflects the contrast of the fused image. A higher value signifies better contrast and a wider distribution of gray levels. It is calculated as:

$$SD(I) = \sqrt{\frac{1}{M \times N}\sum_{i=1}^{M} \sum_{j=1}^{N} (I(i,j) - \mu)^2} \tag{2}$$

where $I(i, j)$ is the pixel intensity at position (i, j), μ is the mean intensity of the image, and $(M x N)$ is the image size.

c) *Mutual Information (MI):* Quantifies the amount of information transferred from the source images (I_V and I_T) to the fused image (I_F). It is calculated as the sum of mutual information between the fused image and each source image:

$$MI(I_F,I_V,I_T) = MI(I_F,I_V) + MI(I_F,I_T) \tag{3}$$

where:

$$MI(X,Y) = \sum_{x \in X}^{X} \sum_{y \in Y}^{Y} p(x,y)log\left(\frac{p(x,y)}{p(x)p(y)}\right) \tag{4}$$

d) ***Edge Intensity (▽G):*** Measures the total amount of edge information preserved in the fused image using the Sobel operator. A higher value indicates sharper edges and better preservation of structural details.

$$\nabla G(I) = \sum_{i=1}^{M} \sum_{j=1}^{N} \sqrt{S_x(i,j)^2 + S_y(i,j)^2} \tag{5}$$

where S_x and S_y are the horizontal and vertical Sobel gradient maps of the image I.

e) *Structural Similarity Index (SSIM):* Measures the structural similarity between the fused image and the source images. The final metric is the average of the SSIM scores computed against the visible and thermal images:

$$SSIM(I_F, I_V, I_T) = \frac{SSIM(I_F, I_V) + SSIM(I_F, I_T)}{2} \tag{6}$$

where $SSIM(X, Y)$ is the standard SSIM function, which considers luminance, contrast, and structure.

f) *Noise Estimation (NE):* Estimates the level of noise in the fused image by calculating the mean absolute difference between the image and a smoothed version of itself.

$$NE(I) = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} |I(i,j) - I_{blur}(i,j)| \tag{7}$$

where I_{blur} is the image I convolved with a 3x3 Gaussian kernel.

g) *Wavelet Standard Deviation (WSD):* Analyzes the image in the frequency domain. We performed a single-level Haar wavelet decomposition of the fused image to obtain Approximation (cA), Horizontal (cH), Vertical (cV), and Diagonal (cD) sub-bands. The standard deviation of each sub-band is computed as a metric. Higher standard deviations in detail sub-bands (cH, cV, cD) typically correspond to richer texture and detail.

7. *Pseudocode for video_utils.py:* To facilitate qualitative analysis and visual demonstration of the fusion results, two types of video outputs are generated. The first is a standard video composed of the sequence of fused frames. The second is a side-by-side comparison video that synchronizes the original visible, thermal, and resulting fused frames into a single view. The procedures for generating these videos are detailed in Algorithms 9 and 10 in Tables 10 and 11.

Algorithm 9: Fused Video Assembly

Objective: To compile an ordered sequence of fused image frames into a single video file.

Inputs:
- FusedFrames: An ordered list of file paths to the fused images.
- OutputPath: The file path for the output video.
- FPS: The desired frames per second for the output video.

Output:
- A video file at OutputPath.

Procedure:

```
1.  BEGIN
2.  IF FusedFrames is empty THEN
3.  **Log** "No frames to assemble." and **TERMINATE**.
4.  END IF
5.  first_frame ← LoadImage (FusedFrames[0]).
6.  video_dims ← (width of first_frame, height of first_frame).
7.  video_writer ← InitializeVideoWriter (OutputPath, FPS, video_dims).
8.  FOR EACH frame_path in FusedFrames DO
9.  `frame` ← **LoadImage**(`frame_path`).
10. **IF** dimensions of `frame` ≠ `video_dims` **THEN**
11. `frame` ← **Resize**(`frame`, `video_dims`).
12. **END IF**
13. `video_writer`.**WriteFrame**(`frame`).
14. END FOR
15. video_writer.Finalize().
16. Log "Fused video saved to " + OutputPath.
17. END
```

Algorithm 10: Side-by-Side Comparison Video Assembly

Objective: To create a composite video displaying the visible, thermal, and fused frames simultaneously for qualitative comparison.

Inputs:

- FrameTriplets: An ordered list of corresponding (VisiblePath, ThermalPath, FusedPath) tuples.
- OutputPath: The file path for the output comparison video.
- FPS: The desired frames per second for the output video.

Output:

- A side-by-side comparison video file at OutputPath.

Procedure:

```
1. BEGIN
2. IF FrameTriplets is empty THEN
3. **Log** "No frame triplets to assemble." and **TERMINATE**.
4. END IF
5. // Determine panel dimensions from the first fused frame.
6. first_fused ← LoadImage(FrameTriplets[0].FusedPath).
7. panel_dims ← (width of first_fused, height of first_fused).
8. composite_dims ← (panel_dims.width * 3, panel_dims.height).
9. video_writer ← InitializeVideoWriter(OutputPath, FPS, composite_dims).
10. FOR EACH triplet in FrameTriplets DO
11. $I_V$ ← **LoadImageAndResize**(`triplet.VisiblePath`, `panel_dims`).
12. $I_T$ ← **LoadImageAndResize**(`triplet.ThermalPath`, `panel_dims`).
13. $I_F$ ← **LoadImageAndResize**(`triplet.FusedPath`, `panel_dims`).
14. // Add identifying labels to each panel.
15. **AddLabel**($I_V$, "Visible").
16. **AddLabel**($I_T$, "Thermal").
17. **AddLabel**($I_F$, "Fused").
18. // Create the composite frame.
19. $I_{composite}$ ← **HorizontallyConcatenate**($I_V$, $I_T$, $I_F$).
20. `video_writer`.**WriteFrame**($I_{composite}$).
21. END FOR
22. video_writer.Finalize().
23. Log "Comparison video saved to " + OutputPath.
24. END
```

8. Pseudocode for model_2015.py: For one of our baseline comparisons, we implemented a placeholder gradient-domain fusion method based on the structural-transfer approach proposed by Connah et al. [7]. This technique, which we refer to as Model2015, operates by transferring the structural saliency from a combined visible-thermal gradient field to the gradient field of the color visible image. The core of the method involves a per-pixel Singular Value Decomposition (SVD) to find an optimal rotation matrix for this transfer. The process is detailed in Algorithm 11 in Table 12.

Algorithm 11: Gradient-Domain Fusion via SVD (Model2015)

Objective: To fuse a visible color image (I_V) and a thermal intensity image (I_T) by transforming the gradient field of I_V based on the structural information from both.

Inputs:

- Visible image, I_V (3-channel BGR)
- Thermal image, I_T (single-channel intensity)
- Sobel kernel size, k_{grad}

Output:

- A fused image, I_F.

```
Procedure:
    1. BEGIN
    2. // 1. Construct Source and Reference Gradient Fields
    3. $I_H \leftarrow$ Concatenate ($I_V$, $I_T$) to form a 4-channel image.
    4. $\nabla H \leftarrow$ ComputeGradients ($I_H$ , $k_{grad}$).
Comment: For each pixel, $\nabla H$ is a 4x2 matrix where each row is the [$\nabla x$ , $\nabla y$] gradient of a channel.
    5. $\nabla R \leftarrow$ ComputeGradients ($I_V$, $k_{grad}$).
Comment: For each pixel, $\nabla R$ is a 3x2 matrix for the B, G, R channels.
    6. // 2. Compute Optimal Rotation via SVD (per-pixel)
    7. FOR EACH pixel p from 1 to ($M$ x $N$) DO
    8. // Decompose the source and reference gradient matrices
    9. $U_H(p)$, $S_H(p)$, $V_H^T(p)$ ← **SVD**($\nabla H(p)$).
    10. $U_R(p)$, $S_R(p)$, $V_R^T(p)$ ← **SVD**($\nabla R(p)$).
    11. // Form the target gradient by combining structural components
    12. $A_H(p)$ ← **DiagonalMatrix**($S_H(p)$).
    13. $\nabla F(p)$ ← $U_R(p) \cdot A_H(p) \cdot V_H^T(p)$.
Comment: This transfers the structure/orientation ($V_H^T$) and magnitude ($A_H$) from the combined field,
while preserving the color direction ($U_R$) from the visible image.
    14. END FOR
    15. // 3. Reconstruct Fused Image from Target Gradient Field
    16. Decompose $\nabla F$ into its channel components: $\nabla F_B$ , $\nabla F_G$ , $\nabla F_R$.
    17. $I_{F,B} \leftarrow$ ReintegrateGradientField ($\nabla F_B$).
    18. $I_{F,G} \leftarrow$ ReintegrateGradientField ($\nabla F_G$).
    19. $I_{F,R} \leftarrow$ ReintegrateGradientField ($\nabla F_R$).
Comment: This step solves the Poisson equation $\nabla^2 I=div(\nabla F)$. A common
approximation is using the inverse Fourier transform.
    20. $I_F \leftarrow$ CombineChannels ($I_{F,B}$ , $I_{F,G}$ , $I_{F,R}$).
    21. RETURN ClipAndScale ($I_F$).
    22. END
```

9. *Pseudocode for model_2020.py:* We also implement a multi-scale fusion algorithm, designated Model2020, which is inspired by the techniques presented by Zhanget al. [6]. This method operates on a per-channel basis, first decomposing each source image into a low-frequency base layer and a high-frequency detail layer. It then employs distinct saliency-driven strategies to fuse each layer before reconstructing the final fused image. This two-pronged approach allows for the preservation of large-scale energy from the thermal image while retaining fine-grained textures from the visible image. The complete procedure is detailed in Algorithm 12 in Table 13.

```
Algorithm 12: Multi-Scale Decomposition Fusion (Model2020)
Objective: To fuse images by separately processing their base (low-frequency) and detail (high-frequency)
components.
Input:
    ● Visible image, $I_V$ (3-channel BGR)
    ● Thermal image, $I_T$ (3-channel BGR)
    ● Hyperparameters: A set of kernel sizes and sigmas for decomposition and saliency.
Output:
    ● A fused image, $I_F$.
Procedure:
    1. BEGIN
    2. Initialize $I_F$ as a zero matrix with the same dimensions as $I_V$.
    3. FOR EACH color channel c in {B, G, R} DO
```

```
4.  // **1. Two-Scale Decomposition**
5.  $B_V, D_V$ ← **Decompose**($I_{V,c}$) using Gaussian blur.
6.  $B_T, D_T$ ← **Decompose**($I_{T,c}$) using Gaussian blur.
7.  // **2. Base Layer Fusion (Saliency-driven)**
8.  $S_V$ ← **SaliencyMap**($B_V$) via absolute Laplacian.
9.  $S_T$ ← **SaliencyMap**($B_T$) via absolute Laplacian.
10. $P_{base}$ ← **PixelwiseMax**($S_V, S_T$). // Creates a binary decision map.
11. $W_{base}$ ← **RefineWeights**($P_{base}$, guide_image=$B_V$) using a Guided Filter.
 Comment: The guided filter ensures edge-preserving smoothing of the weight map.
12. $B_F$ ← $W_{base} \odot B_V + (1 - W_{base}) \odot B_T$.
13. // **3. Detail Layer Fusion (Activity-driven)**
14. $A_V$ ← **ActivityMap**($D_V$) via median filter and absolute Laplacian.
15. $A_T$ ← **ActivityMap**($D_T$) via median filter and absolute Laplacian.
16. $P_{detail}$ ← **PixelwiseMax**($A_V, A_T$). *// Creates a binary decision map.*
17. $W_{detail}$ ← **GaussianBlur**($P_{detail}$).
18. $D_F$ ← $W_{detail} \odot D_V + (1 - W_{detail}) \odot D_T$.
19. // **4. Channel Reconstruction**
20. $I_{F,c}$ ← $B_F + D_F$.
21. END FOR
22. RETURN ClipAndScale ($I_F$).
23. END
```

8.2. *Experimentation for Image Fusion Evaluation and Specifications:*

For the purposes of this study, we did comprehensive evaluations of the five fusion algorithms over three Image Fusion datasets and two Video Fusion datasets, with the details of Image Fusion analysis given below:

Results: The Evaluation Metrics of the four Traditional Python Scripts, for TNO inputs, are given in Table 14 below:

Evaluation Metrics	Model2015	Model2020	Model2024	FCDFusion (Pixel)	Vectorized FCDFusion
Proc Time	1.02407	0.08176	0.04930	0.55993	**0.01339**
Fps	0.97650	12.23093	20.28331	1.78594	**74.70618**
Entropy	4.86302	**7.19838**	6.57182	6.79328	6.79328
Mutual Info Src	**0.37664**	**0.37664**	**0.37664**	**0.37664**	**0.37664**
Std Dev	15.37691	**42.22995**	25.77782	29.67213	29.67213
Noise	2.60156	**1.26796**	2.45345	1.52549	1.52549
Ssim Accuracy	0.06733	0.68329	0.72144	**0.75052**	**0.75052**
Wavelet Std Ca	25.63098	**84.02899**	50.24942	58.67422	58.67422
Wavelet Std Ch	**11.21688**	5.63849	7.97520	5.65705	5.65705
Wavelet Std Cv	**12.44379**	6.21105	7.71256	6.63317	6.63317
Wavelet Std Cd	2.84782	1.48546	**3.13290**	1.74220	1.74220
Edge Sum	11116.89946	11615.77586	**13626.53868**	11044.74359	11044.74359

Note: Best results for each metric are in bold. Mutual Information is calculated on the source images and is therefore identical for all methods.

The fused outputs of the models, with the input images in RGB and NIR forms, are given in Figure 1 below:

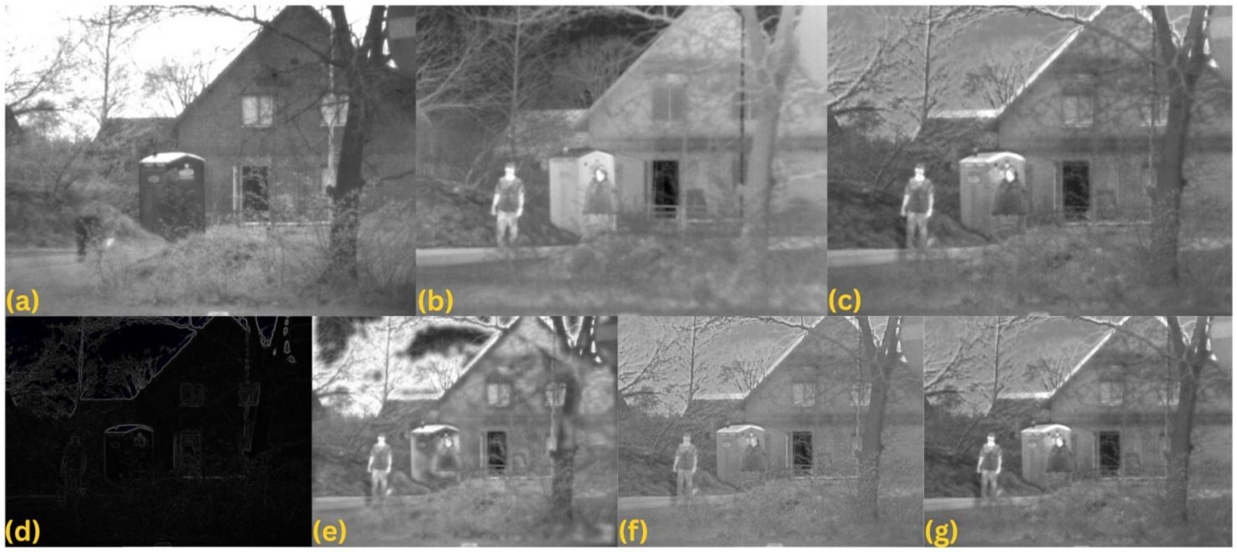


Figure 1. Qualitative Comparison of Fusion Algorithms on a Grayscale Night Scene from the TNO Dataset. This figure demonstrates the performance of the algorithms on a standard military-relevant benchmark scenario, where the inputs and outputs are grayscale: *(a) Source Intensified Visual (RGB) Input:* Shows the structure and texture of the building and foliage, *(b) Source Infrared (IR) Input:* Clearly reveals the heat signature of a person standing in front of the house, who is otherwise camouflaged in the visible image, *(c) Vectorized FCDFusion (Proposed):* Delivers a high-fidelity output that effectively integrates the thermal signature of the person while preserving the textural details of the house and trees, *(d) Model2015 [7]:* Fails to produce a photorealistic image, instead generating an output that resembles a high-contrast edge map, *(e) Model2020 [6]:* Produces the highest contrast for the thermal target, making the person appear brightest against the background, but slightly sacrifices the texture of the building, *(f) Model2024 [5]:* Creates a well-balanced image but with slightly lower overall contrast compared to the other top-performing methods, *(g) FCDFusion (Pixel) [4]:* The output is visually and numerically identical to the proposed vectorized version.

Results: On our initial tests of the fusion models, we initially used all four bands of the Landsat Bands-2-5 to create a full image of (8811 x 8851) pixels. To ensure consistency, the earlier Python codes were then reformatted to fit the current pipeline for the Landsat Dataset, which resulted in memory errors when implementing a vectorized version of Connah et al [7] as our model2015. However, for high-resolution imagery such as the full Landsat scenes (which would be around 367 MB in size for all four images), this approach was found to be computationally infeasible due to memory constraints exceeding 16GB of RAM. Therefore, for the current comparative analysis, the fusion algorithms were re-evaluated on cropped sections of the dataset of (1024 x 1024) pixels. The Evaluation Metrics of the four Traditional Scripts, for the cropped Landsat inputs, are given in Table 15 below:

Evaluation Metrics	Model2015	Model2020	Model2024	FCDFusion (Pixel)	Vectorized FCDFusion
			Processing Speed		
Time Taken (s) ↓	2.51850	0.23590	0.12230	1.32450	**0.04530**
FPS (Frames Per Second) ↑	0.39710	4.23950	8.17590	0.75500	**22.07600**

Information Content					
Entropy ↑	6.20230	7.63637	7.61523	**7.82431**	**7.82431**
Mutual Info (Src)	2.18480	2.18480	2.18480	2.18480	2.18480
Image Quality					
Std. Dev. (Contrast) ↑	21.84400	49.53679	48.85781	**64.35662**	**64.35662**
Noise (Estimated) ↓	5.55947	**2.04521**	2.30647	2.78091	2.78091
SSIM Accuracy ↑	0.05072	**0.99240**	0.99096	0.88472	0.88472
Detail & Edge Preservation					
Wavelet Std (cA) ↑	38.43914	98.48273	97.02740	**127.86609**	**127.86609**
Wavelet Std (cH) ↑	**14.76928**	7.94292	8.48642	10.90886	10.90886
Wavelet Std (cV) ↑	**12.69612**	6.88394	7.35766	9.28038	9.28038
Wavelet Std (cD) ↑	**7.19381**	2.49882	2.80556	3.49731	3.49731
Edge Sum (x10^3) ↑	**61601.83548**	43358.13739	45736.68012	57024.60071	57024.60071

Note: Best results for each metric are in bold. Mutual Information is calculated on the source images and is therefore identical for all methods.

The fused outputs of the models, with input images from the Landsat dataset, are given in Figure 2 below:

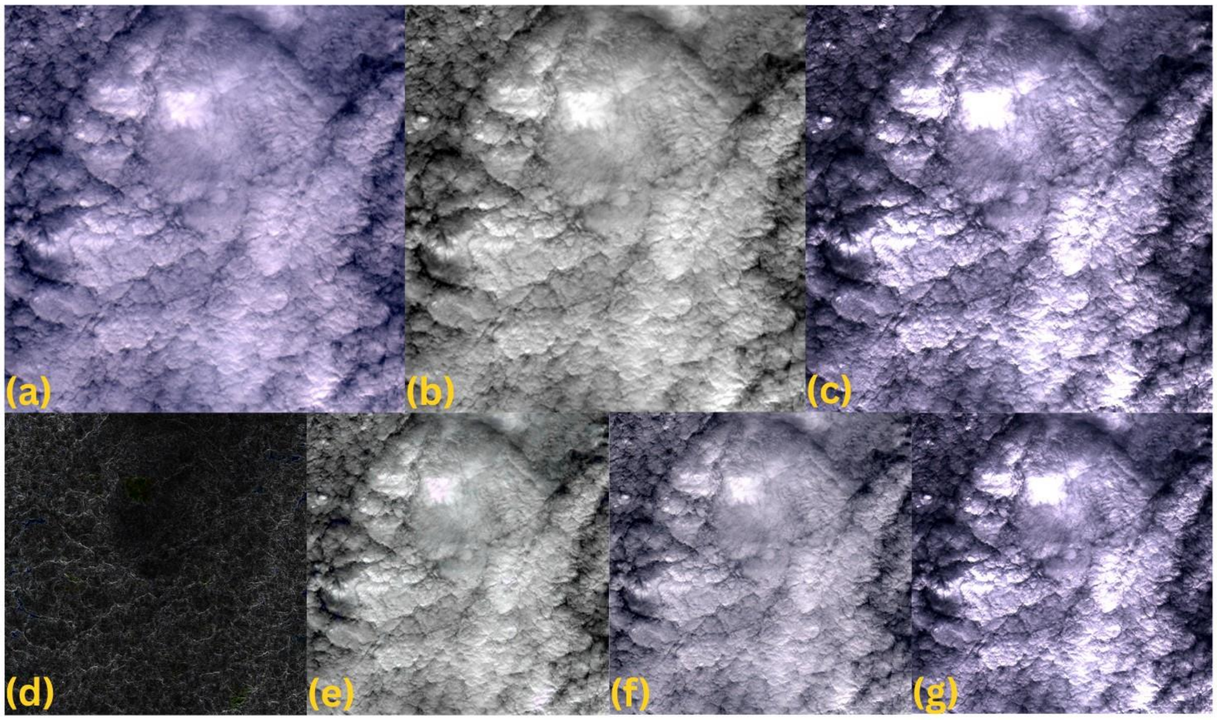


Figure 2. Qualitative Comparison of Fusion Algorithms on a Cropped Landsat Scene. This figure showcases algorithm performance on high-resolution satellite imagery with complex natural textures. The source images are of a cloud formation, providing a challenging test for detail and contrast preservation: *(a) Source Visible (RGB) Input:* Shows the natural color of the clouds, with subtle variations in brightness, *(b) Source Near-Infrared (IR) Input:* Reveals a high degree of texture and structural detail within the cloud formation that is less apparent in the visible image *(c) Vectorized FCDFusion (Proposed):* Produces a high-contrast output that effectively enhances the textural details from the IR input. However, its direct intensity scaling introduces a significant color shift, resulting in a dramatic, purple-tinted appearance, *(d) Model2015 [7]:* Fails to produce a coherent image, resulting in an output that

primarily captures high-frequency edge information, *(e) Model2020 [6]:* Achieves the highest structural fidelity in this test case (SSIM of 0.992), creating a photorealistic fusion that cleanly integrates the IR texture without significant color distortion, *(f) Model2024 [5]:* Produces a high-quality fusion that is visually very similar to the Model2020 output, confirming its strong performance on this dataset, *(g) FCDFusion (Pixel) [4]:* The output is visually and numerically identical to the proposed vectorized version.

1. *FLIR Dataset [1] (specifications):*

Full Name: FLIR ADAS Dataset version 2.0.0.

Data Acquisition and Sensors: The dataset was collected from a vehicle-mounted camera system and features footage from various locations, including the United States, England, and France.

- *Visible Spectrum Camera:* A Teledyne FLIR BlackFly S (model BFS-U3-51S5C) with a 52.8° horizontal field of view (HFOV) lens was used to capture 8-bit RGB images.
- *Thermal Camera:* A Teledyne FLIR Tau 2 (with a 13 mm f/1.0 lens) providing a 45° HFOV captured thermal images. The camera operated in T-linear mode, providing 16-bit raw thermal data, though the publicly provided .jpeg files are post-processed 8-bit images.
- *Synchronization:* The camera pair was time-synchronized using Teledyne FLIR's Guardian software, ensuring a direct temporal correspondence between visible and thermal frames.

Dataset Usage Notes: The dataset is organized into Training, Validation, and Video segments. Our study utilizes the Validation Video Set. While the dataset provides extensive object annotations (e.g., person, car, bike) in COCO format, our study focuses on full-frame pixel-level image fusion and therefore does not make use of these annotations.

2. *CAMEL Dataset [2, 3] (specifications):*

Full Name: Context-Aware Multi-spectral Light-field Dataset

Data Acquisition and Sensors:

- *Visible Spectrum Camera:* Sony Xperia smartphone, capturing RGB video at 30 FPS with resolutions up to 3840x2160 (4K).
- *Thermal Camera:* FLIR Vue Pro, capturing thermal video at 30 FPS with a fixed resolution of 336x256.

Synchronization Method: Frame synchronization is derived from the <Frame Number> column in the SeqX-Vis.txt and SeqX-IR.txt annotation files. Our pipeline processes the intersection of these frame numbers to ensure temporal alignment.

Selected Sequences: The four chosen sequences represent a diverse range of conditions, as summarized in Table 5.

Table 16: Characteristics of Selected CAMEL Dataset Sequences

Characteristic	Sequence 1	Sequence 2	Sequence 8	Sequence 13
Time & Conditions	Day (1:18 PM), Clear	Day (1:37 PM), Clear	Night (7:23 PM), Clear, Low Light	Night (9:08 PM), Partly Cloudy, Low Light
Temp. / Windchill (°F)	66.9 / N/A	71.1 / N/A	34.0 / 31.0	39.9 / 35.1
Visual Resolution	1280x720	1280x720	1280x720	3840x2160 (4K)
IR Resolution	336x256	336x256	336x256	336x256

Camera Setup	Stationary (Tripod)	Stationary (Tripod)	Stationary (Tripod)	Handheld (Platform & Scene Motion)
Scene & Content	Outdoor, campus field	Outdoor, campus, trees	Outdoor, campus path	Elevated, campus view at night
Number of Frames Used	1488	809*	450	469*

Note: The number of frames used (e.g., 809 for Seq-2) is based on the common frame numbers found and successfully processed in our pipeline, which may differ slightly from the total frames listed in the info file (1220 for Seq-2).

8.3. Evaluation of the Video Fusion Approach:

8.3.1. CAMEL Dataset Sequence-1:

a) *Table of Average Values with 95% Confidence Intervals (Table 17):*

Evaluation Metrics	**Model2015**	**Model2020**	**Model2024**	**FCDFusion (Pixel)**	**Vectorized FCDFusion**
proc_time	0.38077 [0.37881, 0.38273]	0.03689 [0.03658, 0.03720]	0.02249 [0.02233, 0.02264]	0.85016 [0.83349, 0.86682]	**0.00699 [0.00689, 0.00709]**
fps	2.65125 [2.63857, 2.66393]	27.84180 [27.61085, 28.07275]	45.20803 [44.92594, 45.49013]	1.54469 [1.48364, 1.60573]	**151.84551 [149.99339, 153.69763]**
entropy	5.72876 [5.72701, 5.73051]	**7.40476 [7.40350, 7.40602]**	7.03408 [7.03162, 7.03654]	7.12554 [7.12201, 7.12906]	7.12554 [7.12201, 7.12906]
mutual_info_src	**0.61465 [0.61372, 0.61558]**	**0.61465 [0.61372, 0.61558]**	**0.61465 [0.61372, 0.61558]**	**0.61465 [0.61372, 0.61558]**	**0.61465 [0.61372, 0.61558]**
std_dev	27.91540 [27.87615, 27.95465]	47.31721 [47.28135, 47.35307]	41.85437 [41.80213, 41.90662]	**47.86228 [47.74638, 47.97818]**	**47.86228 [47.74638, 47.97818]**
noise	5.73656 [5.72894, 5.74417]	**3.31000 [3.30224, 3.31777]**	5.67348 [5.66326, 5.68371]	4.13313 [4.11995, 4.14632]	4.13313 [4.11995, 4.14632]
ssim_accuracy	0.07427 [0.07416, 0.07438]	0.60131 [0.60095, 0.60167]	0.62410 [0.62377, 0.62444]	**0.64763 [0.64726, 0.64799]**	**0.64763 [0.64726, 0.64799]**
wavelet_std_cA	49.33465 [49.26710, 49.40220]	93.07668 [93.00868, 93.14468]	79.78656 [79.68985, 79.88328]	**93.59022 [93.36612, 93.81431]**	**93.59022 [93.36612, 93.81431]**
wavelet_std_cH	18.77459 [18.74242, 18.80676]	12.73643 [12.69773, 12.77513]	**18.77521 [18.73988, 18.81055]**	15.55572 [15.50772, 15.60372]	15.55572 [15.50772, 15.60372]
wavelet_std_cV	**16.51802 [16.49064, 16.54540]**	10.71139 [10.68682, 10.73596]	15.38719 [15.35472, 15.41966]	11.77171 [11.72635, 11.81707]	11.77171 [11.72635, 11.81707]
wavelet_std_cD	**7.59748 [7.58788, 7.60708]**	3.84466 [3.83616, 3.85315]	7.18019 [7.17075, 7.18963]	4.79473 [4.77960, 4.80985]	4.79473 [4.77960, 4.80985]
edge_sum	3888.42803 [3881.84967, 3895.00638]	3654.59635 [3645.83715, 3663.35554]	**4709.84182 [4699.52519, 4720.15844]**	3845.76263 [3832.05443, 3859.47082]	3845.76263 [3832.05443 , 3859.47082]

b) *Table of Ranged Values (Table 18):*

Metric (Ranged) W=12 Dataset: CAMEL Seq1 (N=1488)	**model2015**	**model2020**	**model2024**	**FCDFusion (Pixel)**	**FCDFusion (Vectorized)**
Processing Speed					
Proc. Time (ms) Range	269.0 - 466.0	21.0 - 62.0	15.0 - 47.0	113.0 - 2289.0	**2.0 - 14.0**
FPS Range	2.14 - 3.71	16.10 - 46.64	21.48 - 64.58	0.44 - 8.82	**73.63 - 474.25**
Information Content					
Entropy Range	5.625 - 5.870	**7.351 - 7.517**	6.949 - 7.207	6.922 - 7.300	6.922 - 7.300
Mutual Info (Src) Range	0.537 - 0.635	0.537 - 0.635	0.537 - 0.635	0.537 - 0.635	0.537 - 0.635
Image Quality					
Std. Dev. Range	25.905 - 31.371	45.672 - 51.119	39.085 - 44.034	**41.035 - 51.524**	**41.035 - 51.524**
Noise Metric Range	5.355 - 6.397	**3.068 - 3.911**	5.307 - 6.385	3.514 - 4.931	3.514 - 4.931
SSIM Accuracy Range	0.069 - 0.081	0.565 - 0.609	0.592 - 0.631	**0.616 - 0.658**	**0.616 - 0.658**
Detail/Edge Preservation					
Wavelet Std (cA) Range	45.747 - 55.238	89.869 - 100.242	74.411 - 83.453	**80.264 - 100.275**	**80.264 - 100.275**
Wavelet Std (cH) Range	**17.313 - 21.713**	11.895 - 15.968	**17.745 - 21.174**	13.235 - 18.528	13.235 - 18.528
Wavelet Std (cV) Range	**15.426 - 19.232**	9.903 - 13.240	14.491 - 18.445	9.987 - 15.376	9.987 - 15.376
Wavelet Std (cD) Range	**7.060 - 8.480**	3.594 - 4.644	6.635 - 7.935	4.167 - 5.943	4.167 - 5.943
Edge Sum (x10³) Range	3.610 - 4.539	3.379 - 4.282	**4.380 - 5.407**	3.217 - 4.631	3.217 - 4.631

Note that, W- Number of Active Workers, N- Number of Images processed. Best values from each metric are highlighted in bold.

8.3.2. CAMEL Dataset (Sequence-2):

a) Table of Average Values with 95% Confidence Intervals (Table 19):

Evaluation Metrics	**Model2015**	**Model2020**	**Model2024**	**FCDFusion (Pixel)**	**Vectorized FCDFusion**
proc_time	0.36435 [0.36257, 0.36613]	0.03928 [0.03884, 0.03972]	0.02276 [0.02253, 0.02299]	0.28048 [0.26914, 0.29183]	**0.00663 [0.00652, 0.00675]**
fps	2.75760 [2.74486, 2.77034]	26.07000 [25.80147, 26.33854]	44.70918 [44.33432, 45.08404]	4.69507 [4.52993, 4.86022]	**159.26041 [156.62526, 161.89556]**
entropy	6.52083 [6.51939, 6.52227]	**7.59131 [7.59089, 7.59172]**	7.37546 [7.37474, 7.37619]	7.07726 [7.07607, 7.07845]	7.07726 [7.07607, 7.07845]
mutual_info_src	**0.16535 [0.16473, 0.16597]**	**0.16535 [0.16473, 0.16597]**	**0.16535 [0.16473, 0.16597]**	**0.16535 [0.16473, 0.16597]**	**0.16535 [0.16473, 0.16597]**
std_dev	33.05826 [33.02281, 33.09372]	**54.28169 [54.24785, 54.31553]**	45.49684 [45.47309, 45.52059]	38.95872 [38.92679, 38.99065]	38.95872 [38.92679, 38.99065]
noise	9.43656 [9.42555, 9.44758]	7.26633 [7.25883, 7.27384]	10.63292 [10.62527, 10.64057]	**5.77814 [5.77379, 5.78250]**	**5.77814 [5.77379, 5.78250]**
ssim_accuracy	0.09372 [0.09364, 0.09381]	0.57667 [0.57648, 0.57686]	**0.58813 [0.58788, 0.58838]**	0.57105 [0.57067, 0.57144]	0.57105 [0.57067, 0.57144]
wavelet_std_cA	56.05784 [55.99991, 56.11577]	**104.58163 [104.51063, 104.65264]**	82.11758 [82.06811, 82.16705]	74.64821 [74.58292, 74.71350]	74.64821 [74.58292, 74.71350]

wavelet_std_cH	26.17410 [26.13622, 26.21197]	21.02859 [21.01131, 21.04588]	**28.18760 [28.17112, 28.20408]**	16.24728 [16.23578, 16.25878]	16.24728 [16.23578, 16.25878]
wavelet_std_cV	19.55588 [19.53515, 19.57661]	17.32409 [17.30833, 17.33986]	**23.00226 [22.98577, 23.01874]**	13.17732 [13.16760, 13.18703]	13.17732 [13.16760, 13.18703]
wavelet_std_cD	12.69779 [12.67614, 12.71945]	10.29657 [10.28403, 10.30910]	**14.58391 [14.56988, 14.59794]**	7.81056 [7.80327, 7.81785]	7.81056 [7.80327, 7.81785]
edge_sum	6303.68356 [6296.05787, 6311.30925]	6166.29038 [6161.88220, 6170.69855]	**7860.94888 [7857.07131, 7864.82646]**	4646.31679 [4642.87726, 4649.75632]	4646.31679 [4642.87726, 4649.75632]

b) Table of Ranged Values (Table 20):

Metric (Ranged) W=12 Dataset: CAMEL Seq2 (N=809)	**model2015**	**model2020**	**model2024**	**FCDFusion (Pixel)**	**FCDFusion (Vectorized)**
Processing Speed					
Proc. Time (ms) Range	332.0 - 821.0	19.0 - 51.0	17.0 - 68.0	136.0 - 1804.0	**3.0 - 11.0**
FPS Range	1.22 - 3.02	19.43 - 52.43	14.68 - 60.14	0.55 - 7.36	**90.14 - 386.57**
Information Content					
Entropy Range	6.446 - 6.583	**7.574 - 7.610**	7.351 - 7.393	7.047 - 7.117	7.047 - 7.117
Mutual Info (Src) Range	0.145 - 0.186	0.145 - 0.186	0.145 - 0.186	0.145 - 0.186	0.145 - 0.186
Image Quality					
Std. Dev. Range	31.466 - 34.587	**52.956 - 55.363**	44.724 - 46.014	38.097 - 40.047	38.097 - 40.047
Noise Metric Range	8.913 - 9.929	6.955 - 7.570	10.361 - 10.884	**5.617 - 5.959**	**5.617 - 5.959**
SSIM Accuracy Range	0.091 - 0.097	0.568 - 0.583	**0.581 - 0.596**	0.560 - 0.583	0.560 - 0.583
Detail/Edge Preservation					
Wavelet Std (cA) Range	53.422 - 58.577	**101.758 - 106.891**	80.546 - 83.210	72.817 - 76.811	72.817 - 76.811
Wavelet Std (cH) Range	24.708 - 27.684	20.288 - 21.707	**27.538 - 28.834**	15.836 - 16.788	15.836 - 16.788
Wavelet Std (cV) Range	18.529 - 20.497	16.571 - 17.902	**22.366 - 23.726**	12.798 - 13.541	12.798 - 13.541
Wavelet Std (cD) Range	11.767 - 13.581	9.823 - 10.760	**14.017 - 15.083**	7.513 - 8.122	7.513 - 8.122
Edge Sum ($x10^3$) Range	5.966 - 6.659	5.975 - 6.331	**7.725 - 7.979**	4.513 - 4.779	4.513 - 4.779

8.3.3. CAMEL Dataset (Sequence-8):

a) Table of Average Values with 95% Confidence Intervals (Table 21):

Evaluation Metrics	**Model2015**	**Model2020**	**Model2024**	**FCDFusion (Pixel)**	**Vectorized FCDFusion**
proc_time	0.37529 [0.37224, 0.37833]	0.03844 [0.03795, 0.03894]	0.02283 [0.02255, 0.02311]	0.36900 [0.35171, 0.38630]	**0.00723 [0.00711, 0.00736]**
fps	2.68412 [2.66324, 2.70500]	26.51894 [26.17246, 26.86541]	44.45350 [43.97515, 44.93186]	3.57540 [3.38634, 3.76446]	**143.23956 [140.59536, 145.88376]**
entropy	5.35775 [5.35464, 5.36086]	**7.21134 [7.21014, 7.21254]**	6.91056 [6.91007, 6.91105]	6.92170 [6.91967, 6.92374]	6.92170 [6.91967, 6.92374]

mutual_info_src	**0.19736 [0.19690, 0.19782]**	**0.19736 [0.19690, 0.19782]**	**0.19736 [0.19690, 0.19782]**	**0.19736 [0.19690, 0.19782]**	**0.19736 [0.19690, 0.19782]**
std_dev	23.57773 [23.53455, 23.62091]	**38.54885 [38.51329, 38.58441]**	32.52874 [32.51531, 32.54217]	35.08179 [35.04173, 35.12186]	35.08179 [35.04173, 35.12186]
noise	3.71045 [3.70418, 3.71672]	2.89934 [2.89551, 2.90316]	4.15562 [4.14999, 4.16124]	**2.66500 [2.66104, 2.66897]**	**2.66500 [2.66104, 2.66897]**
ssim_accuracy	0.11865 [0.11851, 0.11879]	0.62037 [0.62008, 0.62066]	**0.66206 [0.66188, 0.66224]**	0.66086 [0.66072, 0.66100]	0.66086 [0.66072, 0.66100]
wavelet_std_cA	43.29818 [43.21602, 43.38034]	**75.07743 [75.00551, 75.14935]**	61.15224 [61.12559, 61.17889]	68.18700 [68.10697, 68.26702]	68.18700 [68.10697, 68.26702]
wavelet_std_cH	13.71286 [13.68993, 13.73578]	12.63248 [12.62199, 12.64297]	**15.66471 [15.65144, 15.67797]**	11.68114 [11.66584, 11.69644]	11.68114 [11.66584, 11.69644]
wavelet_std_cV	11.54036 [11.51625, 11.56448]	11.36387 [11.34447, 11.38328]	**14.34993 [14.32657, 14.37330]**	10.84098 [10.82086, 10.86109]	10.84098 [10.82086, 10.86109]
wavelet_std_cD	5.25143 [5.24319, 5.25968]	4.31575 [4.30904, 4.32246]	**6.43927 [6.42910, 6.44945]**	4.40322 [4.39558, 4.41086]	4.40322 [4.39558, 4.41086]
edge_sum	3059.81753 [3054.33937, 3065.29568]	3657.62866 [3655.28207, 3659.97525]	**4116.79803 [4113.24238, 4120.35368]**	3027.67591 [3024.77164, 3030.58019]	3027.67591 [3024.77164 , 3030.58019]

b) Table of Ranged Values (Table 22):

Metric (Ranged) W=12 Dataset: CAMEL Seq8 (N=450)	**model2015**	**model2020**	**model2024**	**FCDFusion (Pixel)**	**FCDFusion (Vectorized)**
Processing Speed					
Proc. Time (ms) Range	301.0 - 555.0	20.0 - 57.0	16.0 - 38.0	111.0 - 1773.0	**4.0 - 16.0**
FPS Range	1.80 - 3.33	17.46 - 49.76	26.02 - 62.24	0.56 - 9.04	**63.55 - 281.18**
Information Content					
Entropy Range	5.271 - 5.509	**7.154 - 7.246**	6.897 - 6.934	6.852 - 6.966	6.852 - 6.966
Mutual Info (Src) Range	0.177 - 0.206	0.177 - 0.206	0.177 - 0.206	0.177 - 0.206	0.177 - 0.206
Image Quality					
Std. Dev. Range	22.480 - 25.174	**36.944 - 39.428**	32.202 - 33.090	33.507 - 35.996	33.507 - 35.996
Noise Metric Range	3.522 - 3.969	2.774 - 3.145	3.978 - 4.511	**2.456 - 2.853**	**2.456 - 2.853**
SSIM Accuracy Range	0.116 - 0.124	0.613 - 0.634	**0.654 - 0.670**	0.655 - 0.668	0.655 - 0.668
Detail/Edge Preservation					
Wavelet Std (cA) Range	41.306 - 46.377	**71.893 - 76.852**	60.494 - 62.101	65.161 - 70.067	65.161 - 70.067
Wavelet Std (cH) Range	12.983 - 14.469	12.401 - 13.098	**15.187 - 16.432**	10.976 - 12.145	10.976 - 12.145
Wavelet Std (cV) Range	10.984 - 12.721	10.501 - 11.911	**13.289 - 15.035**	9.707 - 11.238	9.707 - 11.238
Wavelet Std (cD) Range	5.013 - 5.528	4.027 - 4.604	**6.092 - 6.900**	4.054 - 4.577	4.054 - 4.577
Edge Sum (x10^3) Range	2.900 - 3.311	3.538 - 3.790	**3.998 - 4.347**	2.855 - 3.148	2.855- 3.148

8.3.4. CAMEL Dataset (Sequence-13):

a) Table of Average Values (Table 23):

Evaluation Metrics	Model2015	Model2020	Model2024	FCDFusion (Pixel)	Vectorized FCDFusion
proc_time	0.36684 [0.36386, 0.36981]	0.03897 [0.03849, 0.03945]	0.02336 [0.02308, 0.02363]	0.79394 [0.76564, 0.82223]	**0.00823 [0.00808, 0.00838]**
fps	2.74674 [2.72546, 2.76802]	26.13236 [25.80824, 26.45649]	43.43326 [42.98766, 43.87886]	1.62562 [1.51672, 1.73453]	**126.69738 [124.26775, 129.12702]**
entropy	4.71702 [4.68577, 4.74827]	**7.20156 [7.18018, 7.22294]**	6.72864 [6.71179, 6.74550]	6.16741 [6.13965, 6.19516]	6.16741 [6.13965, 6.19516]
mutual_info _src	**0.22362 [0.21918, 0.22807]**	**0.22362 [0.21918, 0.22807]**	**0.22362 [0.21918, 0.22807]**	**0.22362 [0.21918, 0.22807]**	**0.22362 [0.21918, 0.22807]**
std_dev	20.03708 [19.73964, 20.33452]	**43.31849 [42.83534, 43.80164]**	34.52177 [34.26852, 34.77503]	30.56618 [30.19549, 30.93686]	30.56618 [30.19549, 30.93686]
noise	2.47010 [2.41724, 2.52297]	1.90049 [1.86409, 1.93689]	2.70586 [2.65629, 2.75544]	**1.59122 [1.54846, 1.63398]**	**1.59122 [1.54846, 1.63398]**
ssim_accura cy	0.20677 [0.20515, 0.20840]	0.53497 [0.53177, 0.53818]	**0.56484 [0.56221, 0.56748]**	0.53259 [0.52925, 0.53593]	0.53259 [0.52925, 0.53593]
wavelet_std_ cA	37.05171 [36.50972, 37.59370]	**85.61900 [84.65021, 86.58779]**	67.04040 [66.55716, 67.52365]	60.05851 [59.34064, 60.77638]	60.05851 [59.34064, 60.77638]
wavelet_std_ cH	11.62082 [11.40614, 11.83550]	10.15019 [9.97926, 10.32111]	**12.41159 [12.21561, 12.60756]**	8.43074 [8.24856, 8.61293]	8.43074 [8.24856, 8.61293]
wavelet_std_ cV	8.76010 [8.55702, 8.96319]	7.68175 [7.54408, 7.81941]	**9.64035 [9.43214, 9.84857]**	6.87285 [6.68526, 7.06044]	6.87285 [6.68526, 7.06044]
wavelet_std_ cD	4.17230 [4.10124, 4.24336]	2.56578 [2.52169, 2.60987]	**4.20821 [4.12988, 4.28654]**	2.64796 [2.57499, 2.72094]	2.64796 [2.57499, 2.72094]
edge_sum	2200.41917 [2156.22759, 2244.61074]	2889.07095 [2840.60256, 2937.53933]	**3070.08140 [3018.80554, 3121.35726]**	2046.34371 [2000.49248, 2092.19493]	2046.34371 [2000.49248, 2092.19493]

b) Table of Ranged Values (Table 24):

Metric (Ranged) W=12 Dataset: CAMEL Seq13 (N=469)	model2015	model2020	model2024	FCDFusion (Pixel)	FCDFusion (Vectorized)
Processing Speed					
Proc. Time (ms) Range	264.0 - 529.0	21.0 - 50.0	17.0 - 47.0	117.0 - 1996.0	**3.0 - 12.0**
FPS Range	1.89 - 3.79	19.84 - 46.94	21.13 - 58.01	0.50 - 8.55	**86.94 - 319.08**
Information Content					
Entropy Range	3.900 - 5.300	**6.640 - 7.542**	6.067 - 6.939	5.140 - 6.603	5.140 - 6.603
Mutual Info (Src) Range	0.152 - 0.382	0.152 - 0.382	0.152 - 0.382	0.152 - 0.382	0.152 - 0.382
Image Quality					
Std. Dev. Range	12.747 - 25.745	**31.326 - 51.474**	26.234 - 39.135	18.944 - 37.304	18.944 - 37.304

Noise Metric Range	1.313 - 3.583	1.080 - 2.744	1.591 - 3.856	**0.783 - 2.409**	**0.783 - 2.409**
SSIM Accuracy Range	0.166 - 0.240	0.452 - 0.585	**0.497 - 0.608**	0.433 - 0.582	0.433 - 0.582
Detail/Edge Preservation					
Wavelet Std (cA) Range	23.767 - 47.365	**61.664 - 101.988**	50.796 - 76.031	37.194 - 73.083	37.194 - 73.083
Wavelet Std (cH) Range	4.801 - 16.018	4.898 - 14.022	**6.084 - 17.246**	3.598 - 12.129	3.598 - 12.129
Wavelet Std (cV) Range	4.553 - 13.700	4.758 - 11.372	**5.370 - 14.990**	3.148 - 10.642	3.148 - 10.642
Wavelet Std (cD) Range	2.296 - 5.466	1.378 - 3.593	**2.292 - 5.990**	1.140 - 4.122	1.140 - 4.122
Edge Sum ($\times 10^3$) Range	1.251 - 3.156	1.830 - 3.847	**1.917 - 4.214**	1.145 - 2.903	1.145- 2.903

8.3.5. FLIR Dataset:

a) Table of Average Values with 95% Confidence Intervals (Table 25):

Evaluation Metrics	**Model2015**	**Model2020**	**Model2024**	**FCDFusion (Pixel)**	**Vectorized FCDFusion**
proc_time	4.79834 [4.76297, 4.83372]	0.48702 [0.48267, 0.49136]	0.29128 [0.28951, 0.29305]	14.22562 [14.07706, 14.37418]	**0.10803 [0.10704, 0.10901]**
fps	0.22088 [0.21914, 0.22263]	2.22544 [2.20506, 2.24582]	3.56919 [3.54591, 3.59248]	0.08062 [0.07953, 0.08170]	**10.05231 [9.95887, 10.14574]**
entropy	4.48388 [4.47095, 4.49681]	**7.33099 [7.32592, 7.33605]**	6.94998 [6.93773, 6.96223]	7.11714 [7.10654, 7.12773]	7.11714 [7.10654, 7.12773]
mutual_info_src	**0.36562 [0.36001, 0.37123]**	**0.36562 [0.36001, 0.37123]**	**0.36562 [0.36001, 0.37123]**	**0.36562 [0.36001, 0.37123]**	**0.36562 [0.36001, 0.37123]**
std_dev	11.94296 [11.78131, 12.10460]	**46.88819 [46.66759, 47.10878]**	34.91821 [34.54998, 35.28645]	43.38750 [42.88232, 43.89268]	43.38750 [42.88232, 43.89268]
noise	2.36584 [2.34621, 2.38547]	**1.16612 [1.15603, 1.17621]**	2.21030 [2.18861, 2.23198]	1.59265 [1.57433, 1.61097]	1.59265 [1.57433, 1.61097]
ssim_accuracy	0.14153 [0.13907, 0.14398]	0.69737 [0.69467, 0.70008]	**0.75026 [0.74815, 0.75237]**	0.74703 [0.74445, 0.74961]	0.74703 [0.74445, 0.74961]
wavelet_std_cA	21.38947 [21.07653, 21.70242]	**93.53674 [93.09201, 93.98147]**	69.26984 [68.53076, 70.00892]	86.49506 [85.48295, 87.50718]	86.49506 [85.48295, 87.50718]
wavelet_std_cH	**7.31220 [7.22799, 7.39641]**	4.42284 [4.37934, 4.46633]	5.60666 [5.56087, 5.65245]	4.44833 [4.41931, 4.47735]	4.44833 [4.41931, 4.47735]
wavelet_std_cV	**6.42721 [6.35360, 6.50083]**	4.02001 [3.98152, 4.05849]	5.35710 [5.31078, 5.40342]	4.20825 [4.17524, 4.24127]	4.20825 [4.17524, 4.24127]
wavelet_std_cD	**3.41805 [3.38965, 3.44645]**	1.58653 [1.56933, 1.60373]	3.11679 [3.08361, 3.14998]	2.19019 [2.16056, 2.21981]	2.19019 [2.16056, 2.21981]
edge_sum	20629.02539 [20376.38460, 20881.66618]	20341.18631 [20113.50960, 20568.86303]	**22393.40578 [22127.04408, 22659.76748]**	18665.29249 [18456.91319, 18873.67179]	18665.29249 [18456.91319 , 18873.67179]

b) Table of Ranged Values (Table 26):

Metric (Ranged) W=12 Dataset: FLIR (N=3749)	model2015	model2020	model2024	FCDFusion (Pixel)	FCDFusion (Vectorized)
Processing Speed					
Proc. Time (ms) Range	3310.0 - 16009.0	246.0 - 810.0	179.0 - 452.0	2619.0 - 23290.0	**46.0 - 218.0**
FPS Range	0.06 - 0.30	1.23 - 4.06	2.21 - 5.59	0.04 - 0.38	**4.58 - 21.58**
Information Content					
Entropy Range	3.306 - 5.280	**6.837 - 7.657**	6.433 - 7.712	6.607 - 7.717	6.607 - 7.717
Mutual Info (Src) Range	0.063 - 0.722	0.063 - 0.722	0.063 - 0.722	0.063 - 0.722	0.063 - 0.722
Image Quality					
Std. Dev. Range	5.150 - 20.974	**29.956 - 66.694**	22.509 - 63.969	26.159 - 82.153	26.159 - 82.153
Noise Metric Range	0.809 - 3.864	**0.478 - 1.762**	0.816 - 3.071	0.543 - 2.504	0.543 - 2.504
SSIM Accuracy Range	0.062 - 0.299	0.494 - 0.795	**0.600 - 0.839**	0.560 - 0.843	0.560 - 0.843
Detail/Edge Preservation					
Wavelet Std (cA) Range	7.722 - 38.877	**59.478 - 133.248**	44.861 - 127.712	52.226 - 164.212	52.226 - 164.212
Wavelet Std (cH) Range	**3.022 - 13.630**	2.130 - 7.473	2.833 - 8.563	2.411 - 5.889	2.411 - 5.889
Wavelet Std (cV) Range	**2.051 - 11.624**	1.755 - 6.161	2.146 - 7.821	1.784 - 6.586	1.784 - 6.586
Wavelet Std (cD) Range	**1.223 - 5.673**	0.598 - 2.577	0.975 - 4.368	0.658 - 3.676	0.658 - 3.676
Edge Sum (x10³) Range	5.722 - 43.624	7.954 - 36.122	**8.165 - 40.448**	6.703 - 34.073	6.703 - 34.073

Note: Best results for each metric are in bold. Mutual Information is calculated on the source images and is therefore identical for all methods.

8.4. Ablation Study Results (Table 27):

Evaluation Metrics	No Gamma	No Base Gain	No Dynamic Target	No Base + Dynamic	No Gamma + Dynamic	No Base + Gamma	No Base + Gamma + Dynamic	Vectorized FCDFusion
Proc Time	0.00590	0.00600	0.00590	0.00590	0.00581	0.00615	0.00575	**0.00540**
Fps	168.56780	166.11760	170.73610	169.42576	172.20117	162.69604	173.89320	**184.06570**
Entropy	6.54860	6.99420	7.41800	7.03509	6.38998	7.33776	7.10288	**7.42700**
Mutual Info Src	0.78500	0.78500	0.78500	**0.78505**	**0.78505**	**0.78505**	**0.78505**	0.78500
Std Dev	70.11560	44.26180	66.47300	40.60305	54.35337	47.37588	36.36293	**73.36080**
Noise	1.54270	1.49850	1.89360	1.78743	1.63843	**1.48139**	1.71847	1.72970
Ssim Accuracy	0.76920	0.56590	**0.77770**	0.64016	0.72568	0.77417	0.75786	0.77570
Wavelet Std Ca	139.70070	87.78070	132.31160	80.34431	108.13486	94.19605	72.01788	**146.13740**
Wavelet Std Ch	7.28620	8.35910	8.36150	**8.79400**	7.10801	7.04441	7.45866	8.16940
Wavelet Std Cv	6.16160	5.56930	**6.30450**	6.08054	5.94188	4.95421	5.42743	6.15840
Wavelet Std Cd	2.34430	1.99380	**2.46030**	2.24182	2.31675	1.79195	2.05174	2.37080
Edge Sum	5055.81390	5615.48770	**6418.16140**	6353.77112	5234.08334	5412.97218	5938.49495	5974.74680