Neural-Assisted Feature Matching

Internship Report

Author: EL OUARRAT Haytam
Internship Period: Mars – August 2025
Location: SteelSeries, Lille, France

Advisors: Pierre Biret, Damien Granger, Raphaël Greff
University Supervisor: Phillipe Joly

Engineering Degree in Robotics and Interactive Systems
UPSSITECH
University of Toulouse

August 19, 2025

FOR GLORY

Abstract

Real-time analysis of gaming footage requires feature matching algorithms that are both robust and computationally efficient. SteelSeries' Moments software currently relies on the Scale-Invariant Feature Transform (SIFT) to detect and match in-game icons, offering strong invariance to scale, rotation, and illumination changes. However, SIFT's high computational cost and latency make it unsuitable for high frame rate, CPU-bound environments. This work proposes a Neural-Assisted Feature Matching approach that accelerates the most computationally intensive stages of SIFT—keypoint detection and descriptor computation—through the integration of lightweight convolutional neural networks. We investigate recent neural methods, including ALIKE, LightGlue, and XFeat, identifying XFeat as the primary candidate for its favorable speed–accuracy trade-off. A custom gaming-focused dataset was created using real gameplay footage and synthetic transformations to evaluate matching quality, latency, and resource usage. The proposed hybrid pipeline achieves substantial reductions in processing time compared to the existing SIFT-based implementation, while preserving comparable matching accuracy and robustness under varied gaming conditions. These findings demonstrate the potential of neural-assisted methods for real-time, resource-constrained computer vision tasks, with direct applicability to gaming analytics and broader real-time visual correspondence applications.

Contents

FOR GLORY

List of Figures

Chapter 1

Acknowledgements

Chapter 2

Introduction

2.1 Host Organism

2.1.1 SteelSeries

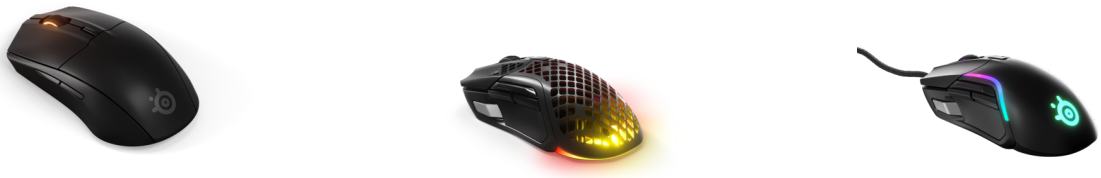
Company history and Background

Steel**Series**, a Danish manufacturer of gaming peripherals, was founded in 2001 by Jacob Wolff-Petersen. The company originally launched under the name Soft Trading, and made its mark with innovative gaming mousepads in the early 2000s. In 2007, Soft Trading rebranded to SteelSeries, reflecting its broadened focus beyond mousepads and into a full range of PC gaming accessories. Key milestones in SteelSeries' evolution include the 2008 acquisition of Ideazon, which brought the Zboard and World of Warcraft gaming keyboard into its portfolio and furthered its presence in the North American market. The company grew rapidly in the 2010s, fueled by its involvement in the esports scene and partnerships with professional gamers. SteelSeries has since expanded its product line to include high-performance gaming mice, keyboards, headsets, and mousepads, becoming a leading brand in the gaming industry.

Key Products and Technologies

SteelSeries, renowned for its gaming peripherals and accessories, spanning several product categories. Its product portfolio includes:

- **Gaming Mice:** High-precision mice with customizable buttons and sensors.



(a) Rival 3 Wireless Gen 2 (b) Aerox 5 (c) Rival 5

Figure 2.1: SteelSeries Gaming Mice

- **Keyboards:** Mechanical and membrane keyboards designed for gaming performance.
- **Headsets:** Wired and wireless headsets with advanced audio features.
- **Mousepads:** Various sizes and materials optimized for different play styles.
- **Software:**
  - **SteelSeries GG:** is an all-in-one software platform that brings together the various tools and services SteelSeries offers to enhance the gaming experience. It serves as the central hub for managing SteelSeries peripherals and includes multiple sub-applications.

(a) Apex Pro Gen 3

(b) Apex Pro Mini Gen 3

(c) Apex Pro TKL Gen 3

Figure 2.2: SteelSeries Gaming Keyboards

(a) Arctis Nova 3 Wireless

(b) Arctis Pro Wireless

(c) Arctis GameBuds™ Glo-range

Figure 2.3: SteelSeries Gaming Keyboards

- **SteelSeries Engine:** the part of GG that handles the core device configuration. It's used to customize settings for SteelSeries mice, keyboards, headsets, and other gear. Through Engine, users can adjust RGB lighting effects, set up macros, and fine-tune mouse sensitivity (DPI).
- SteelSeries' advanced audio suite built specifically for gamers who want precise control over their sound experience. It offers a powerful parametric equalizer that lets users independently customize audio for game sounds, voice chat, and microphone input.
- **SteelSeries Moments:** a gameplay capture tool within GG that automatically records key moments during gaming sessions. It can detect in-game events like kills, wins, or goals and save short clips around those events.

2.1.2 Mission

SteelSeries' mission is to create the best gaming gear in the world, empowering gamers to perform at their best, whether it is for professionals who seek perfection, or casuals who seek a sense of competition and completion. Its implication over the years in the esports scene has made it a trusted brand among professional gamers, and its commitment to innovation continues to drive the development of new products that enhance the gaming experience. Most notably, SKT Gaming, a professional esports organization, has been using SteelSeries products since 2012, and has won multiple championships in games like CS:GO and League of Legends etc.

2.2 Context & Motivation

2.2.1 Feature Matching in Computer Vision

Definition

Feature matching refers to the process of finding corresponding keypoints between two images of the same scene or object. These keypoints are distinctive image features, such as corners, edges, blobs, or regions

Figure 2.4: 5x League of Legends World Champions SKT T1

with significant intensity changes that can be reliably identified in different images, even if those images have undergone transformations.

2.2.2 How Feature Matching Works

Feature matching typically involves several steps:

- **Feature Extraction:** the initial step in feature matching, identifies distinctive, stable, and informative points of high-intensity variation—such as corners or edges—that are invariant to changes in rotation and scale.
- **Feature Description:** the process of computing a descriptor for each keypoint, which is a compact representation of the local image patch around the keypoint. This descriptor should be robust to changes in lighting, viewpoint, and other image transformations.
- **Feature Matching:** the final step involves finding correspondences between the keypoints in the two images based on their descriptors. This is typically done using techniques like nearest neighbor search, where the descriptor of each keypoint in one image is compared to all descriptors in the other image to find the best match.

2.2.3 Scale Invariant Feature Transform

In David G. Lowe's 2004 paper, **SIFT** stands for *Scale Invariant Feature Transform*, a method designed to detect and describe local features in images that remain stable across various transformations. The term encapsulates the core strengths of the algorithm:

- **Scale Invariance:** SIFT demonstrates robustness against changes in image size by detecting extrema in scale-space using a Difference-of-Gaussian approach, making it effective across different zoom levels and image resolutions.
- **Transformation Resilience:** The algorithm maintains stability under various image transformations including rotation, scaling, partial changes in illumination, and viewpoint variations, ensuring reliable feature detection across different viewing conditions.
- **Distinctive Feature Representation:** SIFT identifies keypoints characterized by location, scale, orientation, and descriptor vectors based on local gradients, converting raw image data into a highly distinctive representation suitable for matching and recognition tasks.

Thus, SIFT provides a powerful framework for extracting and matching image features under challenging real-world conditions [1].

2.2.4 Challenges in Gaming Applications

The primary challenge for image processing in real-time gaming is the extreme demand for **speed** and **low latency**, as any analysis must occur within a tight millisecond timeframe to avoid introducing lag. This becomes particularly difficult because image processing algorithms must *compete for the same computational resources* that the game itself is using. This is not a minor issue, as modern AAA titles are specifically designed to **monopolize the GPU**. Games like *Alan Wake 2* or *Cyberpunk 2077*, with demanding features such as *path tracing*, already push high-end graphics cards to their absolute limits merely to render the game world. Adding an additional workload like real-time image processing on top of this creates a severe **performance bottleneck**, forcing developers to make significant trade-offs between **accuracy** and **efficiency**, and to rely on *incredibly lightweight algorithms* that won't cripple the game's frame rate.

2.2.5 Limitations of Traditional Feature Matching Techniques

Real-time applications, such as visual odometry (VO), SLAM, and gaming, require both high accuracy and extremely low latency; thus, feature matching algorithms must satisfy very stringent constraints. Traditional feature-based systems using SIFT or ORB work well for reliable and accurate matching under varying conditions, but they are slower because of the explicit data association steps involved. Such latency, however, is unacceptable when one has to make decisions within milliseconds. Zhao and Vela (2020) characterize this trade-off by stating, "feature-based systems exhibit good performance, yet have higher latency due to explicit data association; direct & semidirect systems have lower latency, but are inapplicable in some target scenarios or exhibit lower accuracy than feature-based ones" [4]. This observation thus attests the traditional pipelines' readymade opposing factors-speed versus precision-while promoting lightweight or neural-assisted alternatives to ensure on-time delivery without siding with robustness.

2.3 Project Objectives

2.3.1 Reproducing Feature Matching Techniques with Neural Networks

Recent years have witnessed a surge of interest in leveraging neural networks to enhance image processing tasks, applications in this field range from image classification and object detection to semantic segmentation and vision language models. The idea is to use neural networks to learn the feature extraction and matching process, potentially leading to faster and more efficient algorithms. This project aims to explore the feasibility of using neural networks to reproduce traditional feature matching techniques, such as SIFT or ORB, while maintaining or improving their performance.

2.3.2 Problem Statement

This research investigates whether lightweight neural networks can effectively accelerate the most computationally intensive stages of traditional feature matching—specifically, keypoint detection and description, in SteelSeries Moments software. By integrating neural-assisted processing into a SIFT-based pipeline, the project aims to achieve substantial reductions in processing time without sacrificing matching accuracy or robustness. The findings will inform not only gaming software optimization but also the broader application of neural-accelerated methods in real-time computer vision, particularly on hardware-limited platforms.

2.4 Industrial Relevance

2.4.1 Integration with SteelSeries Moments Software

For a company like SteelSeries, which leads in the gaming and esports market, user experience is paramount. The primary goal of this project is to achieve a reduced processing time compared to the traditional SIFT method, currently in-use in the SteelSeries Moments software. This will allow users to quickly and efficiently analyze their gaming footage, enhancing their overall experience with the product without relying on heavy computational resources.

Chapter 3

Literature Review: Neural Feature Matching

3.1 Review of Recent Methods

In the last decade, neural networks have profoundly reshaped the landscape of feature matching in all vision tasks, progressively replacing or complementing traditional methods such as SIFT and ORB. Driven by the relentless pursuit of computational efficiency, accuracy and adaptability, researches have turned to deep learning as the logical solution to this challenge, leveraging its powerful representation-learning capabilities to address longstanding issues in classicals methods. particularly in fields demanding fast accurate processing, such as robotics and augmented reality.

3.2 ALIKE: Accurate and Lightweight Keypoint Detection and Descriptor Extraction

To address the issue that existing keypoint detectors are non-differentiable and cannot be optimized via back-propagation, this paper presents a novel, partially differentiable keypoint detection module that outputs accurate sub-pixel keypoints. The method uses a reprojection loss to directly optimize these keypoints and a dispersity peak loss for regularization, ensuring accuracy. Descriptors are also extracted at a sub-pixel level, trained with a stable neural reprojection error loss. This entire process is handled by a lightweight network capable of running at 95 frames per second on 640x480 images. The proposed method achieves performance equivalent to state-of-the-art approaches in homography estimation, camera pose estimation, and visual localization, while significantly reducing inference time.

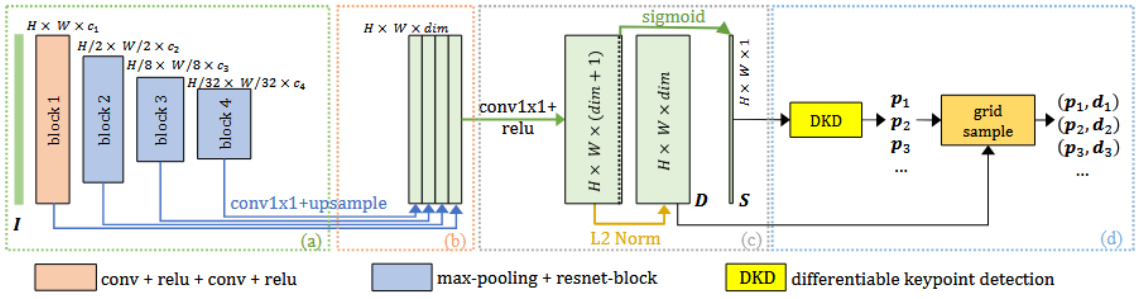


Figure 3.1: architecture overview of ALIKE

3.2.1 Shared Feature Encoder

At its base, ALIKE uses a lightweight Convolutional Neural Network (CNN) as its backbone. This encoder takes an input image and processes it through several convolutional layers to produce a set of shared feature maps. This design is efficient because the computationally intensive process of feature

extraction is done only once, and the results are used for both the detection and description tasks that follow. The focus is on creating a compact yet powerful network that can run quickly on standard hardware.

3.2.2 Keypoint Detection Head

This is where ALIKE's main innovation lies. Instead of just identifying pixels on a grid, the detection head is designed to find keypoints with sub-pixel accuracy.

- **Score Map Generation:** The detection head first produces a reliability or score map, where high values indicate a higher probability of a keypoint being present.
- **Partially Differentiable Detection:** To achieve sub-pixel accuracy, ALIKE performs a soft-argmax operation on local 2x2 patches of the score map. This process is "partially differentiable," meaning it allows the network to learn the precise location of a keypoint within a pixel's boundaries through back-propagation during training.
- **Loss Functions:** The detection process is optimized using two specific loss functions mentioned in the research:
  - **Reprojection Loss:** This loss directly optimizes the sub-pixel coordinates of the keypoints, pushing them to be more geometrically consistent between different views of the same scene.

$$\mathcal{L}_{\text{rep}} = \frac{1}{N} \sum_{i=1}^{N} \|p_i - \hat{p}_i\|_2$$

$\mathcal{L}_{\textbf{rep}}$ The Reprojection Loss value.
N The total number of matching keypoint pairs.
p_i The coordinates of a keypoint in the first image.
$\hat{p}_i$ The coordinates of the corresponding keypoint from the second image, projected onto the first.
$\|\cdot\|_2$ The L2 norm, which calculates the Euclidean distance between the two points.

  - **Dispersity Peak Loss:** This acts as a regularizer, encouraging the score map to have sharp, distinct peaks. This prevents keypoints from clumping together and ensures they are well-defined.

$$\mathcal{L}_{\text{peak}} = \frac{1}{W \times H} \sum_{x,y} \left(\max(0, S(x,y) - \max_{\substack{dx,dy \in \{-1,0,1\}^2 \\ (dx,dy) \neq (0,0)}} S(x+dx, y+dy)) \right)$$

$\mathcal{L}_{\textbf{peak}}$ The Dispersity Peak Loss value.
W, H The width and height of the score map.
$S(x, y)$ The score of the pixel at coordinate (x, y).
$\max_{...}$ The maximum score found within the 8-pixel neighborhood around the pixel (x, y).
$\max(0, \cdot)$ A rectifier (ReLU) ensuring the loss is non-negative. It penalizes pixels that are not a strict local maximum.

3.2.3 Descriptor Extraction Head

The descriptor head uses the shared feature maps from the encoder to generate a distinctive descriptor for each detected keypoint.

- **Sub-Pixel Descriptor Extraction:** To match the precision of the detection head, descriptors are also extracted in a sub-pixel manner. The network uses the precise sub-pixel coordinates generated by the detection head to sample from the descriptor feature map using bilinear interpolation. This ensures that the descriptor accurately represents the feature at its exact location, rather than just the nearest pixel center.
- **Training loss:** The descriptors are trained using a stable neural reprojection error loss. This loss function helps in learning descriptors that are robust to viewpoint and illumination changes, leading to more reliable matching.

By combining an efficient backbone with a novel, differentiable method for sub-pixel keypoint detection and description, ALIKE achieves state-of-the-art performance on various matching tasks while maintaining a very high processing speed (e.g., 95 FPS on 640x480 images).

FOR GLORY

3.3 LightGlue: Local Feature Matching at Light Speed

LightGlue is a deep neural network designed to efficiently and accurately match local features between images. It builds upon the SuperGlue architecture by introducing simpler, more efficient design choices that make it faster, more memory-efficient, easier to train, and adaptive to the complexity of the image pair. This adaptability allows LightGlue to process easier image pairs more quickly, making it ideal for time-sensitive tasks like 3D reconstruction. The model and its training code are publicly available. The architecture of LightGlue is designed to match sparse local features between two images efficiently

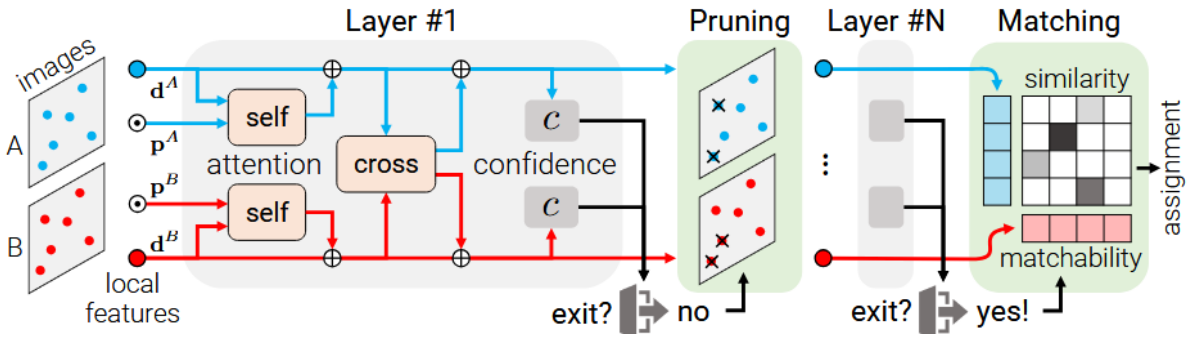


Figure 3.2: architecture overview of LightGlue

while maintaining high accuracy. It builds upon the SuperGlue framework but introduces several key improvements that make it faster, easier to train, and more adaptive. Here's a breakdown of its core architectural components:

3.3.1 Input Representation:

LightGlue takes as input two sets of local features from both reference and target image, each feature consists of normalized 2D coordinates and a 256-dimensional descriptor.

3.3.2 Transformer-based Backbone:

LightGlue uses a stack of Transformer layers to jointly process the features from both images. Each layer includes:

- **Self-Attention Mechanism:** This allows the model to capture long-range dependencies and relationships between features within each image.
- **Cross-Attention Mechanism:** This enables the model to learn correspondences between features from the reference and target images, enhancing the matching process.

3.3.3 Lightweight Matching Head:

After each layer, the network can predict similarity scores for all feature pairs, as well as predict the matchability scores for each point (i.e., whether a point is likely to have a corresponding point in the other image). It combines similarity and matchability into a soft partial assignment matrix.

3.3.4 Adaptive Inference:

This is a unique feature of LightGlue, teh ability to dynamically adjust the computations by using:

- **Adaptive Depth:** after each layer, a classifier predicts if a sufficient number of features
- **Adaptive Depth:** the points predicted as confidently unmatchable are pruned from further processing, reducing computational load.

3.4 XFeat: Accelerated Features for Lightweight Image Matching

This work presents a new lightweight CNN structure that redefines the trade-off between efficiency and accuracy for local feature extraction. In contrast to existing deep models that need GPUs and/or hardware-specific tuning, XFeat offers real-time performance on CPU-only systems by combining three components: a featherweight backbone, a very light keypoint detection branch, and a match refinement module that was new in its design. XFeat also uniquely has both sparse and semi-dense matching modes, the first efficient model to do so, and is suitable for visual localization and pose estimation. XFeat is typically 5–9× faster than other state-of-the-art methods, but offers similar or better accuracy. The three design choices: growth of channel number tripled, early resolution preservation, and course-to-fine refinement, contribute to high quality matching for practical device constraints. It is a scalable, lightweight and practical deep learning architecture for the real world.

3.4.1 Featherweight Network Backbone

The backbone of XFeat is designed to balance out computational efficiency and representational accuracy:

- **Channel Distribution Strategy**: The network begins with a minimal number of channels in the initial convolutional layers, where the spatial resolution is highest, to reduce computational load. As the spatial resolution decreases through subsequent layers, the number of channels is increased, following a tripling strategy (e.g starting with 4 channels and increasing up to 128 channels). This approach ensures that the network remains lightweight without compromising on feature extraction quality.
- **Convolutional Blocks**: The architecture comprises six convolutional blocks, each consisting of basic layers that include 2D convolutions with kernel sizes ranging from 1 to 3, followed by ReLU activations and Batch Normalization. Strides of 2 are used for downsampling, **progressively reducing the spatial dimensions while increasing the depth of feature maps**

3.4.2 Local Feature extraction

XFeat employs specialized heads for different aspects of feature extraction:

- **Descriptors Head**: This component extracts a dense feature map by merging multi-scale features from the encoder using a feature pyramid strategy. Intermediate representations at different scales (1/8, 1/16, and 1/32 of the original resolution) are upsampled and combined to enhance the receptive field without significantly increasing computational demands. A fusion block integrates these representations into the final feature map and an additional convolutional block generates a reliability map that indicates the confidence of each local feature.
- **Keypoint Head**: To efficiently detect keypoints, XFeat introduces a dedicated parallel branch focused on low-level image structures. The input image is represented as a 2D grid of 8x8 pixel cells, each reshaped into 64-dimensional features. This representation preserves spatial granularity within individual cells while allowing rapid 1x1 convolutions for keypoint regression. After four convolutional layers, a keypoint embedding is obtained, encoding the distribution of keypoints within each cell.

3.4.3 Dense Matching and Match Refinement

XFeat includes a lightweight module for dense feature matching:

- Match Refinement Module: This module refines coarse matches to achieve pixel-level accuracy without requiring high-resolution feature maps. It employs a simple MLP (Multi-Layer Perceptron), that predicts offsets between matching features, enabling efficient semi-dense matching suitable for resource-constrained settings.

What is Mutual Nearest Neighbor?

MNN, is a popular matching strategy used in image correspondence tasks to robustly establish matches between two sets of local feature descriptors. It helps ensure that matches are reciprocal, which greatly reduces false matches compared to simple nearest neighbor methods.

Mutual Nearest Neighbors (MNN) Algorithm

- **Compute Nearest Neighbors**: Given two feature sets:
  - Set $A = \{a_1, a_2, \ldots, a_n\}$
  - Set $B = \{b_1, b_2, \ldots, b_m\}$

  For each feature $a_i \in A$, find its nearest neighbor in B using:

$$\text{NN}(a_i, B) = \arg\min_{b_j \in B} \|a_i - b_j\|$$

  where $\|\cdot\|$ is the Euclidean distance metric.

- **Reciprocal Check** For each candidate pair (a_i, b_j) from Step 1:
  1. Verify mutual nearest neighbor relationship:

$$\begin{aligned} &\text{NN}(a_i, B) = b_j \quad \text{(Original condition from Step 1)} \\ &\text{NN}(b_j, A) = a_i \quad \text{(Reciprocal condition)} \end{aligned}$$

  2. The pair (a_i, b_j) is considered valid MNN iff both conditions hold [2][5]

- **Formal Definition** A pair (a_i, b_j) is mutual nearest neighbors if:

$$\forall b_k \in B, \|a_i - b_j\| \leq \|a_i - b_k\| \quad \wedge \quad \forall a_l \in A, \|b_j - a_i\| \leq \|b_j - a_l\|$$

  This bidirectional verification ensures correspondence stability in feature matching applications.

- **Discard Non-mutual Matches** If a pair does not fulfill mutual nearest neighbor criteria, it is discarded. After discarding non-mutual pairs, the final matched pairs are most robust against ambiguous matches.

3.4.4 Sparse & Semi-dense Matching

The paper introduces two different matching modes based on previously explained MNN, *XFeat* and *XFeat**, they differ in the way they carry image matching:

Aspect	XFeat	XFeat* (Star)
Matching strategy	Sparse matching (keypoint-based)	Semi-dense matching (coarse feature maps)
Keypoints used	Small set (up to 4,096 typically)	Larger set (10,000 typically)
Refinement step	No refinement	Match refinement (coarse-to-fine)
Accuracy	Good	Higher (due to dense matching & refinement)
Computational Cost	Lower (faster, simpler)	Slightly higher (but still lightweight)
Application scenarios	Visual localization, fast real-time matching	Pose estimation, dense scene

1. Sparse (XFeat) vs. Semi-dense Matching (XFeat*):
   - *XFeat* extracts a small set of keypoints (defaulting to 4096), uses Mutual Nearest Neighbor to match sparse features. It is fast and suitable for applications that prioritize speed and simplicity, such as visual localization or robotics.
   - *XFeat** extracts semi-dense features, typically around 10,000 descriptors, and retains descriptors densely at different scales. It also performs matching in a coarse-to-fine manner, capturing more detailed correspondences across the image.
2. Match Refinement in *XFeat**:
   XFeat* introduces a lightweight match refinement module:
   Matches descriptors at a coarse level first. Then predicts pixel-level offsets to refine matches to exact pixel coordinates. This refinement significantly improves accuracy, especially in scenarios demanding precise correspondences (e.g., detailed scene reconstruction). *XFeat* on the other hand has no such refinement; matches are at the original, coarse level.
3. To resume:

Metric	XFeat (Basic)	XFeat* (Star)
Speed	Very Fast	Fast (slightly slower due to refinement)
Accuracy	Good	Higher accuracy
Density of matches	Sparse	Semi-dense
Robustness	Robust	More robust

- *XFeat* is a fast, sparse matching approach focusing on extreme computational efficiency.
- *XFeat** is a semi-dense, refined matching approach, offering better accuracy at the slight expense of computational resources.

3.5 Gaps and Limitations

While there have been substantial advancements in neural-supported feature matching, frameworks such as ALIKE, XFeat, and LightGlue each encounter trade-offs limiting their suitability for purpose-built 2D image contexts. ALIKE does a great job at efficient keypoint detection and binary descriptors which makes it suitable for mobile and embedded contexts, but it is limited as it provides no built-in matching pipeline, and poor detection performance when the stimulus contains low-texture or synthetic scenarios, such as gaming or icon-based imagery. LightGlue employs an attention-based transformer model that makes it robust to perspective and lighting changes, but this comes at a high computational cost that also limits its adaptability to low-cost or low-latency environments for rapid feature matching. This state of play lays bare a glaring gap in the research alignment: there is no single feature matching framework that combines the benefit of speed (e.g., ALIKE and XFeat) with the ability to adaptively and robustly handle image distortion to the extent of (or more) than LightGlue, and is specifically optimized for "2D" applications, such as icon localization, synthetic-to-real matching, and dynamic in-game overlays—where conventional 3D priors and richly textured cues are often absent or unreliable. XFeat improves upon some its limitations by adopting a more integrated process to detection, description, and matching in a lightweight end-to-end model that demonstrates impressive efficiency gains when doing computationally expensive matching comparisons, however, its performance is limited when considering view and lighting changes or changing imagery states when the model is representative of highly structured and realistic 2D datasets.

Chapter 4

Methodology

4.1 Problem Formulation

4.1.1 Feature Matching as Correspondence Prediction

The current solution implemented at SteelSeries leverages SIFT (Scale-Invariant Feature Transform) for detecting specific icons and matching them across frames in a live video feed. This method was selected due to its well-documented robustness to scale, rotation, and illumination changes, which is critical in dynamic gaming environments where UI elements may vary slightly between frames or scenes. Unlike deep learning-based object detectors such as YOLO (You Only Look Once) or SSD (Single Shot MultiBox Detector), which require large annotated datasets and frequent re-training to adapt to new games, graphical updates, or icon variations—SIFT operates in a fully handcrafted and deterministic manner, making it model-free and highly generalizable. Its independence from external data or pre-trained models ensures long-term maintainability and consistency, especially in production systems that must support a wide range of games without frequent intervention. Moreover, feature matching methods like SIFT allow us to establish correspondences between keypoints in a pair of images without prior training, enabling quick deployment and adaptability. This is particularly advantageous in scenarios where the visual elements to be detected (such as achievement badges, in-game items, or menu icons) are not part of a predefined object class and may appear in various sizes or partially occluded contexts. The matching process, relying purely on local feature descriptors, operates seamlessly and with minimal setup, reducing engineering overhead while maintaining high matching accuracy. However, while SIFT is reliable, its computational cost—especially on CPU-bound systems without GPU acceleration—poses challenges for real-time performance at high frame rates, motivating the exploration of neural-assisted or hybrid approaches that can retain robustness while improving throughput.

4.1.2 Objectives in Terms of Speed, Accuracy, and Robustness

Performance Requirements

To establish a comprehensive evaluation framework, we define specific performance targets that the enhanced system must achieve across three key dimensions:

Image Size	1280×720	960×540	640×360	320×180	128×72
Scale	100%	75%	50%	25%	10%
KPs detected	636	534	403	155	46
Time	0.0837s	0.0470s	0.0226s	0.0082s	0.0025s

Table 4.1: Performance Requirements and Current Baseline Comparison

4.1.3 Choice of Models

Although each Neural Network solution excel in different aspects, we have to select a model that balances speed, accuracy, and robustness. The chosen model should be lightweight enough to run efficiently on CPU

while still providing high matching precision and recall across various game icons and UI elements. Among the available options, we considered **XFeat** as our primary candidate due to its proven performance.

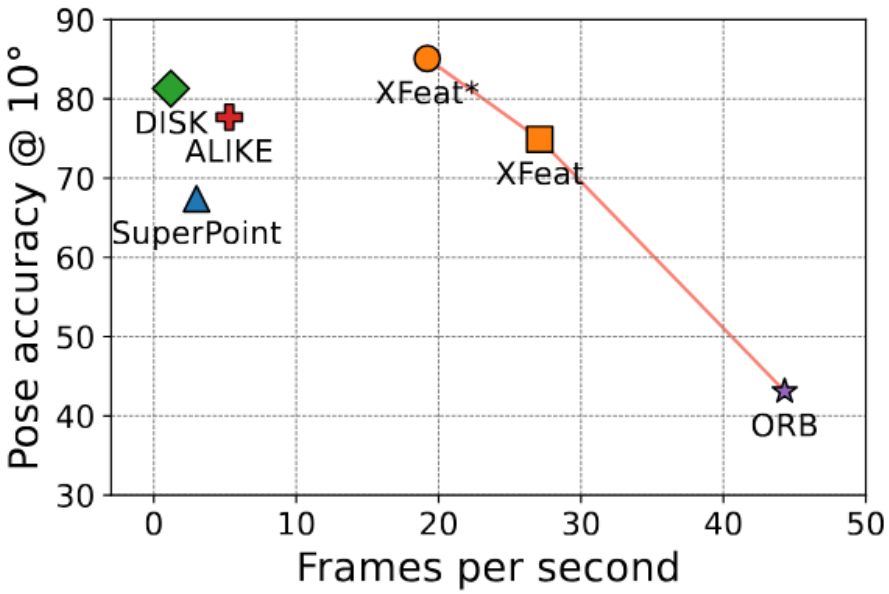


Figure 4.1: Comparison of XFeat with other feature matching methods

4.1.4 Datasets

XFeat utilizes two primary datasets for training and evaluation: the MegaDepth dataset and a COCO_20k subset of the COCO2017 dataset. MegaDepth is widely used in the computer vision community for structure-from-motion and depth estimation tasks, and XFeat's handling of this dataset is adapted from the LoFTR official code. The COCO_20k subset consists of 20,000 images selected from the full COCO2017 train set, with images chosen based on resolution; this subset is provided for convenience, but users must adhere to the COCO terms of use. XFeat's training setup, as described in the accompanying CVPR 2024 paper, leverages these datasets to achieve robust and efficient keypoint detection and description, enabling both sparse and semi-dense image matching suitable for real-world, resource-constrained applications. The datasets provide diverse scenes and challenges, ensuring that XFeat generalizes well to various environments, and supporting its goal of real-time, accurate local feature extraction on standard hardware.

4.2 Experimental Design

4.2.1 Data Collection and Preprocessing

Icons and Templates

The data collection process involves handpicking a wide range of game icons and UI templates that would represent the diversity of visual elements encountered in gaming environments, while avoiding any flat or smooth icon; as these would not provide sufficient texture for feature matching.

Figure 4.2: Example of gaming elements used in dataset creation

Figure 4.3: Example of gaming elements avoided in dataset creation

Footage Collection

To complete a realistic gaming footage scenario, we have picked random footage from various TwitchTV streams, focusing on gameplay that prominently features the UI elements and random backgrounds that could be encountered in real-world gaming sessions. The footage is then processed to extract frames at a certain rate, ensuring that the dataset captures dynamic interactions with the UI elements. Each frame is annotated with the corresponding icons and UI elements present, allowing for precise evaluation of feature matching performance.

Figure 4.4: Example of backgrounds used in dataset creation

This will ensure that the dataset is representative of the actual gaming environments and provides a solid foundation for training and evaluating the feature matching.

4.2.2 Design of Teacher-Student Framework

The teacher-student framework is designed to leverage the strengths of both handcrafted feature matching methods and neural network-based approaches. For this, we have selected two strong candidates: ALIKE,

originally picked by the authors, and a sophisticated SIFT variant, currently in use at SteelSeries. Both methods have been used to extract keypoints and match them across our data samples.

4.2.3 SIFT Implementation

The current solution implemented at SteelSeries leverages a sophisticated SIFT implementation that enhances the traditional SIFT algorithm for our specific use case. The implementation goes as follows:

We employ a carefully tuned Scale-Invariant Feature Transform (SIFT) configuration to maximise the number of stable, repeatable keypoints in varied imaging conditions. The detector is initialised with the following parameters:

- $nOctaveLayers = 4$: controls the number of scale-space intervals per octave. Increasing this value produces more keypoints across finer scale increments, improving detection in textured or low-detail regions at the cost of additional computation.
- $contrastThreshold = 0.02$: sets the minimum contrast required for a candidate keypoint to be retained. Lowering this threshold allows the detector to preserve low-contrast features that may still be distinctive, particularly in low-texture or dim areas, but also increases sensitivity to noise.
- $edgeThreshold = 5$: filters out keypoints lying on edges by evaluating the ratio of principal curvatures in the Difference-of-Gaussians response. Lower values are stricter, rejecting more elongated edge responses; our choice balances the removal of unstable edge points with the retention of useful linear features.
- $\sigma = 1.8$: specifies the Gaussian smoothing applied at the initial scale of each octave. Larger values increase robustness to noise and small perturbations but can blur fine details; $\sigma = 1.8$ offers a compromise between stability and spatial precision.

After detecting keypoints in both the reference and target images, descriptors are computed and normalised by dividing each vector by the sum of its elements. This amplitude normalisation mitigates the effect of overall descriptor magnitude differences, focusing the comparison on relative component values.

Candidate correspondences are generated using a FLANN-based approximate nearest-neighbour search with a KDTree index. For each reference descriptor, the two nearest neighbours in the target descriptor set are retrieved. Match filtering follows a two-pronged strategy:

1. **Relative criterion**: Lowe's ratio test is applied, retaining matches where the closest neighbour's distance is less than 0.7 times that of the second-closest neighbour. This rejects ambiguous matches where multiple candidates are similarly close in descriptor space.
2. **Absolute criterion**: Any match with a nearest neighbour distance below 0.09 is also accepted, even if the ratio test is not satisfied, ensuring that exceptionally strong correspondences are preserved.

The filtered matches are recorded as index pairs into the respective keypoint arrays. This combination of tuned detector parameters, descriptor normalisation, and dual match filtering yields a set of correspondences that is both distinctive and resilient to background clutter and illumination changes.

4.3 Evaluation metrics

Chapter 5

Implementation, Result, and Analysis

5.1 Overview of the approach

Before training, we have reviewed what the chosen network requires as input. Since we are training using synthetic dataset that would imitate our target scenarios, we focused on generating diverse and representative samples.

5.2 Dataset Preparation & Preprocessing

The network expects:

- **Two images:** the reference image, and the target image. Stored as numpy arrays.
- **Two sets of keypoints:**
  - **Reference keypoints:** An array of keypoint coordinates found in the reference image. It has a shape of $(N, 2)$, where N is the number of keypoints, and the coordinates are (x, y).
  - **Target keypoints:** An array of keypoint coordinates found in the target image that correspond to the reference keypoints. It has the same shape $(N, 2)$.
- **Matches:** An array of shape $(N, 2)$ that stores the indices of the matching keypoints. For example, $[i, j]$ means that the i-th keypoint in the reference keypoints matches the j-th keypoint in the target keypoints.
- **Homography Matrix:** A 3 by 3 matrix that relates the reference icon to the target one. It is used to transform points from the reference to the target image, and vice versa.
- **Two binary masks:** Binary masks of the target image that indicates the region where the pixel exists. This is useful to separate the foreground from the background.

These elements would be stored as a zipped numpy file (.npz) to be later accessed during training and evaluation, and stored in a folder structure that mirrors the dataset organization.

5.2.1 Data Synthesis

General Approach

To create a diverse and representative dataset, we employed data synthesis techniques that will allow us to experiment with various scenarios. Our general idea consists of cropping a region of interest from the selected gaming frames, this will help us simulate different viewpoints where the targeted 2D template may appear.

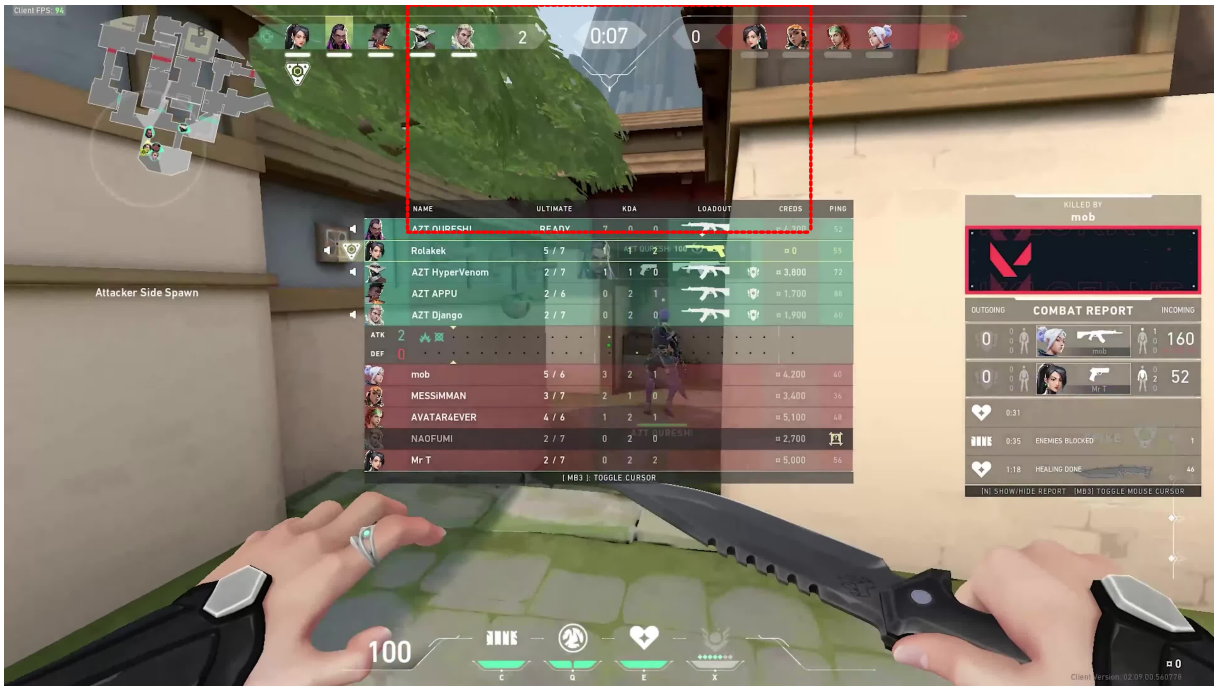


Figure 5.1: Example of a cropped region of interest from a gaming frame.

This newly selected area will serve as our background for the rest of the synthesis process. Then, we select an icon from our set of templates, which will be our target during training. We overlay our icon onto the background at the center, and we ensure that the icon is fully contained within the background region by downscaling it slightly and continuously till it fits perfectly.

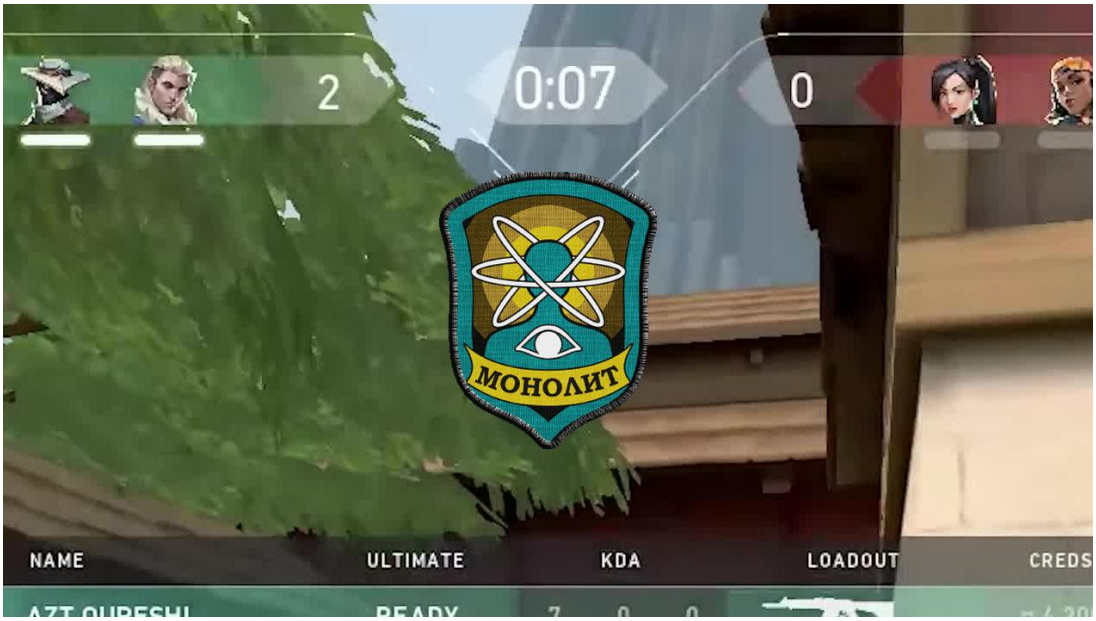


Figure 5.2: Example of the icon overlaid on the background.

While this describes our general design principle for synthetic data generation, the process was refined over several iterations to address specific challenges such as background complexity, icon transformations, and the robustness of extracted keypoints.

FOR GLORY

5.3 Training Procedure & Experimental Setup

Our chosen approach consists of training the network on synthetic data using a teacher-student framework. In the early stages, we kept the same teacher model used by XFeat[3]. ALIKE, was used as a third party teacher tool to extract ground truth keypoints from our training images.

Figure 5.3: Overview of extraction process.

We then generate a binary mask showing the presence of the icon within the background.

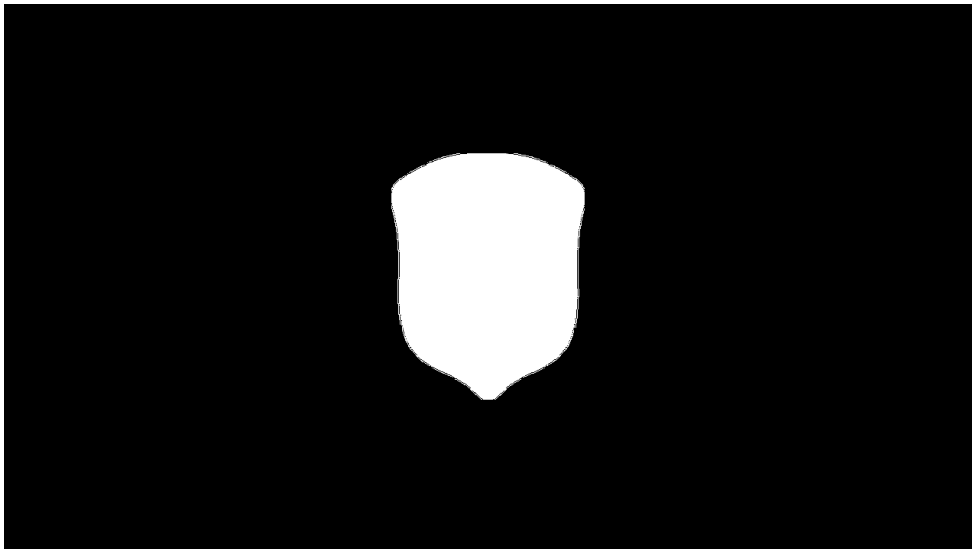


Figure 5.4: Example of the binary mask generated for the icon.

This mask is used to filter out keypoints that do not belong to the icon, ensuring that only relevant features are considered during training. This would teach the model to focus only the keypoints and data existing within the icon region , which is what we want to achieve.

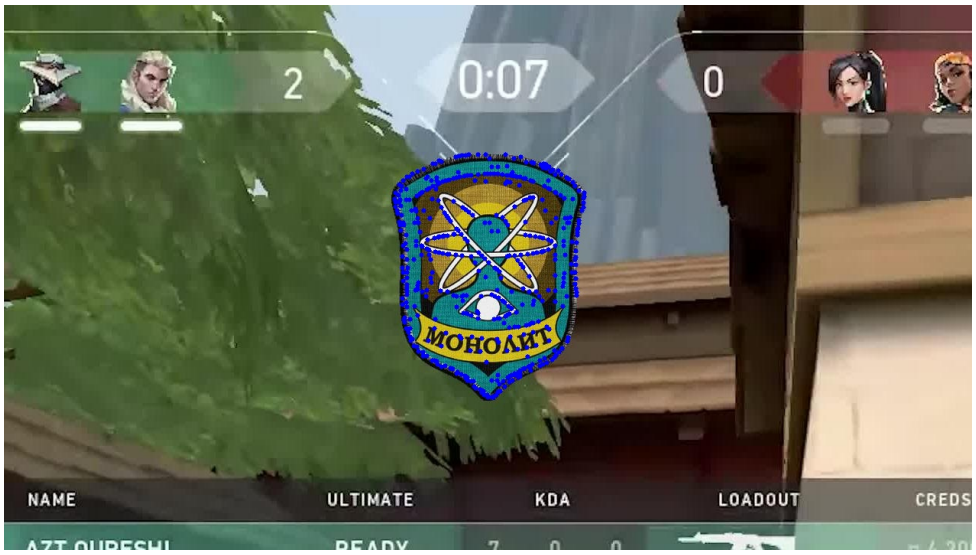


Figure 5.5: Overview of filtering process.

By repeating this process on both **reference** and **target** images, we get a pair of keypoint arrays ready to match using ALIKE's matching algorithm to finally get our training data samples.

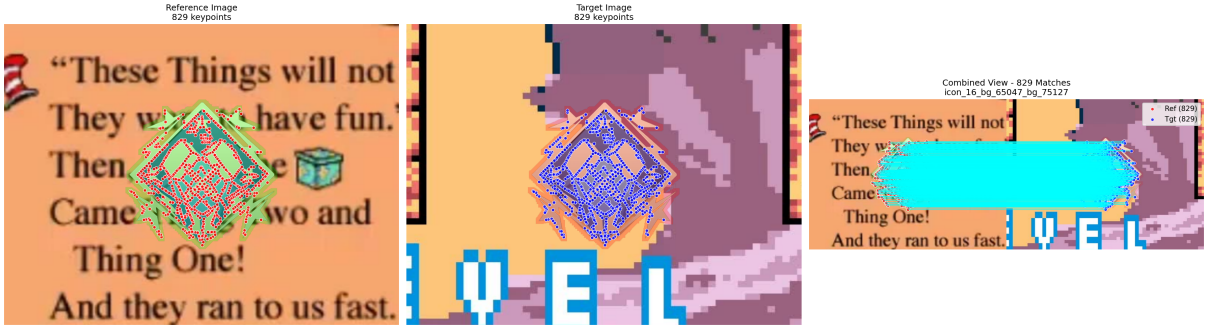


Figure 5.6: Example of the synthesis process, showing the background and the overlaid icon as well as the detected keypoints and masks.

At a first time we trained the model following the default parameters:

- Learning rate: 0.001
- Batch size: 10
- Number of iterations: 160k

This has led us to under average results, prompting further investigation into hyperparameter tuning and data formulation.

5.3.1 Results

The metrics obtained from our initial training runs indicate that while the model is capable of learning the basic features of the icons, it struggles with more complex backgrounds and variations in icon appearance.

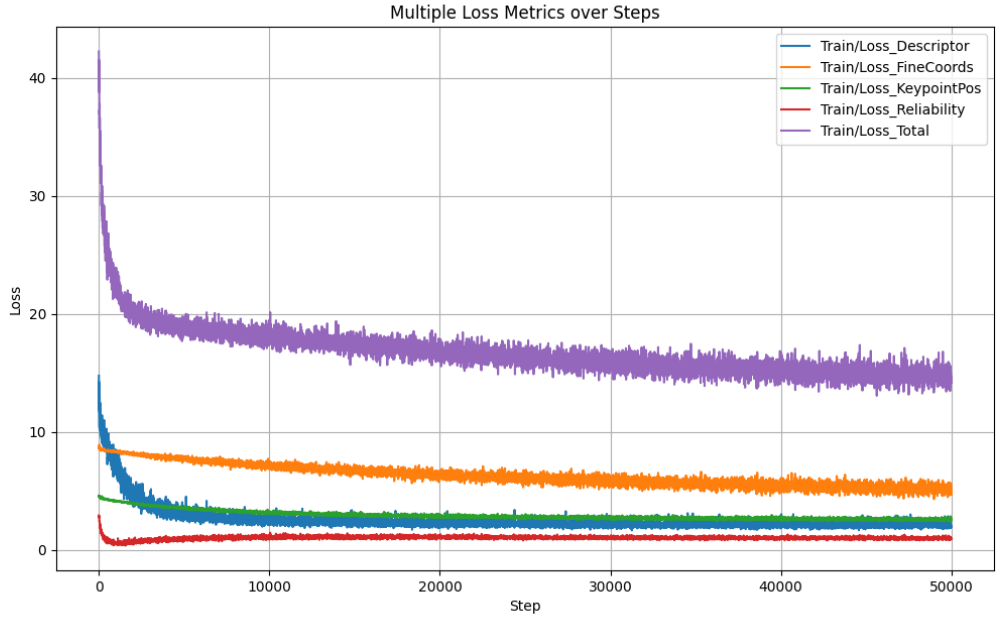


Figure 5.7: Loss curves for initial training runs.

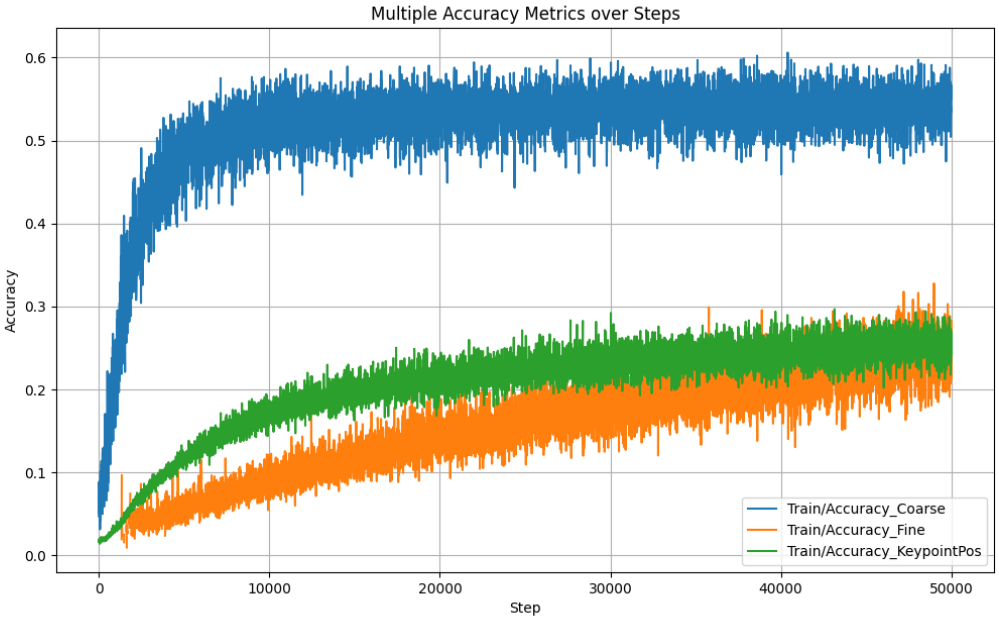


Figure 5.8: Accuracy curves for initial training runs.

Accuracy Metrics

- **Coarse Accuracy:** Rapid increase in the first 5k steps, reaching 55%-60%. Then it plateaus with small oscillations.
- **Keypoint Position Accuracy:** Steady growth until 20k steps, then saturates around 25%-27%.
- **Fine Matching Accuracy:** Slow but consistent improvement, reaching 20%-22% by step 50k, with more variance in later steps.

FOR GLORY

We can interpret this as follows:

- The model learns coarse correspondences very quickly and maintains that performance, suggesting early layers or coarse matching blocks converge fast.
- Fine matching is the slowest to improve, likely because it depends on the coarse stage being strong first, and needs more subtle feature learning.
- Keypoint position accuracy plateaus mid-training, meaning spatial localization stops improving after 20k steps.

Loss Metrics

- **Descriptor Loss:** Sharp drop in the first ∼5k steps, then remains low and stable (around 1–2).
- **Fine Coordinates Loss:** Slow but steady decrease across the 50k steps, still trending down by the end of training.
- **Keypoint Position Loss:** Drops quickly in early training and stabilizes, mirroring the accuracy plateau.
- **Reliability Loss:** Low from the start with minimal change, indicating stable confidence estimation.
- **Total Loss:** Large drop in the first ∼5k steps, followed by a slower but continuous decline until the end.

We can interpret this as follows:

- Most of the learning happens in the first 5k–10k steps, especially for descriptors and keypoint positioning.
- Fine coordinate refinement remains the main bottleneck, as its loss is still higher and accuracy is still improving, suggesting more training could help.
- Reliability loss stability shows that confidence estimation remains consistent throughout training.

Confusion matrix

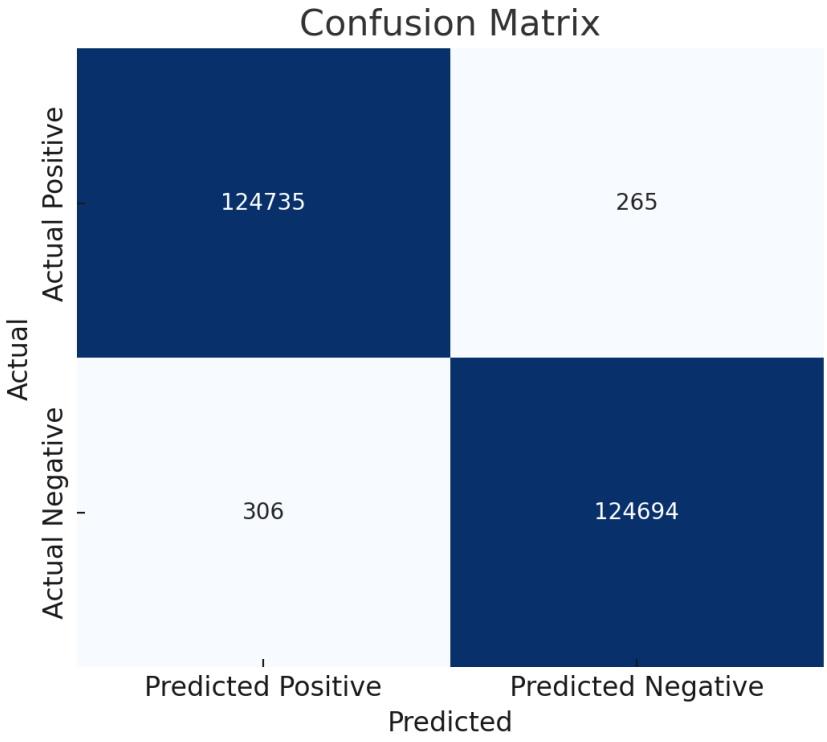


Figure 5.9: Confusion matrix for initial training runs.

FOR GLORY

The confusion matrix shows the following values:

- **True Positives (TP):** 124,735 cases where positive samples were correctly classified.
- **False Negatives (FN):** 265 cases where positive samples were incorrectly classified as negative.
- **False Positives (FP):** 306 cases where negative samples were incorrectly classified as positive.
- **True Negatives (TN):** 124,694 cases where negative samples were correctly classified.

From these values, we can compute:

- **Accuracy:** $\frac{TP+TN}{TP+TN+FP+FN} \approx 99.77\%$
- **Precision:** $\frac{TP}{TP+FP} \approx 99.76\%$
- **Recall:** $\frac{TP}{TP+FN} \approx 99.79\%$
- **F1-Score:** $\approx 99.78\%$

These results indicate that the model performs extremely well, with both precision and recall above 99.7%, suggesting minimal misclassification in both classes, but not enough, as False positives are extremely important in our case, as they would lead to false matches, and ultimately impact the user experience negatively.

Overall Assessment

The training shows strong performance in terms of fast convergence on coarse matching and descriptor learning, with stable behavior and no signs of divergence. However, fine matching accuracy remains low compared to coarse accuracy, and keypoint accuracy saturates early, indicating possible underfitting in fine localization. **To address these issues, several strategies can be considered: extending the training or applying fine-tuning with a smaller learning rate to further improve fine accuracy, reweighting the losses to give more emphasis to the FineCoords component once the coarse stage has converged, and applying data augmentation techniques that highlight sub-pixel differences to strengthen fine matching capabilities. Additionally, a curriculum learning approach could be adopted, starting with simpler examples before progressively increasing difficulty for the fine stage.**

5.4 Iterative Refinement

During experimentation, we implemented and evaluated three alternative synthesis pipelines derived from our general concept:

1. **NAFM A – Static Background-Icon, Transformed Keypoints:** The background and icons remained unchanged, while the keypoints were transformed using the homography matrix generated during the synthesis process. This matrix was applied to map the keypoints from the reference image to the target image, rather than re-extracting them from both images using the teacher model.
2. **NAFM B – Cropped Background/Dynamic Icon-Transformed Keypoints:** The reference image was cropped down to focus on the area of interest, and ignore the background. The target icon was transformed using translation, placing it in a random position within the background.
3. **NAFM C – Unfiltered SIFT Features** The reference and target images were not changed from the variant B, but the keypoints were extracted and matched using SIFT, without filtering them using the binary mask.

Each variant maintained the core pipeline (ROI selection, overlay, keypoint extraction, matching), but emphasized different aspects of visual variability.

5.4.1 Variant A

In variant A, we focused on maintaining a static background and icon, while transforming the keypoints obtained from the reference image to the target image using a homography matrix. This would allow us to generate high quality matches, as our objective is to find absolute correspondences between the reference and the query templates. So we thought that using the transformed keypoints would yield better results than re-extracting them from the target image.

Homography-based keypoint transfer

Let the reference keypoints be $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ with $\mathbf{x}_i = (u_i, v_i)^\top$ in pixel coordinates. Write them in homogeneous form $\tilde{\mathbf{x}}_i = (u_i, v_i, 1)^\top$. Let the homography be $H \in \mathbb{R}^{3\times 3}$.

The mapped keypoints on the target image are

$$\tilde{\mathbf{x}}'_i \sim H\,\tilde{\mathbf{x}}_i, \qquad \mathbf{x}'_i = \left(\frac{\tilde{u}'_i}{\tilde{w}'_i}, \frac{\tilde{v}'_i}{\tilde{w}'_i}\right)^\top,$$

where $\tilde{\mathbf{x}}'_i = (\tilde{u}'_i, \tilde{v}'_i, \tilde{w}'_i)^\top$. In matrix form for all points:

$$\tilde{X}' \sim H\tilde{X}, \quad X' = \begin{bmatrix} \tilde{X}'_{1,:}/\tilde{X}'_{3,:} \\ \tilde{X}'_{2,:}/\tilde{X}'_{3,:} \end{bmatrix}^\top.$$

Notes. If $\tilde{w}'_i = 0$ the point maps to infinity and is discarded.

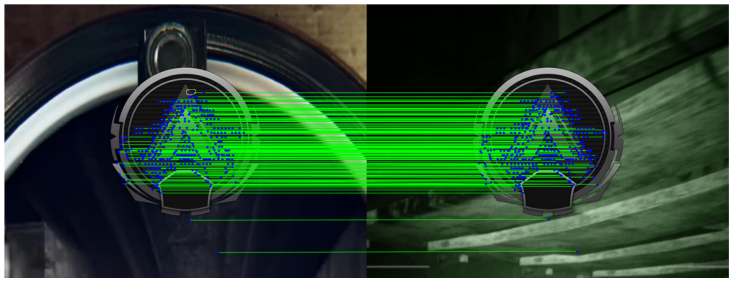


Figure 5.10: Example of the synthesis process for variant A

Although this approach seems promising, it relies heavily on the quality of our data synthesis process, and the homography matrix generation. In practice, we found that the performance was sensitive to these factors.

Accuracy metrics

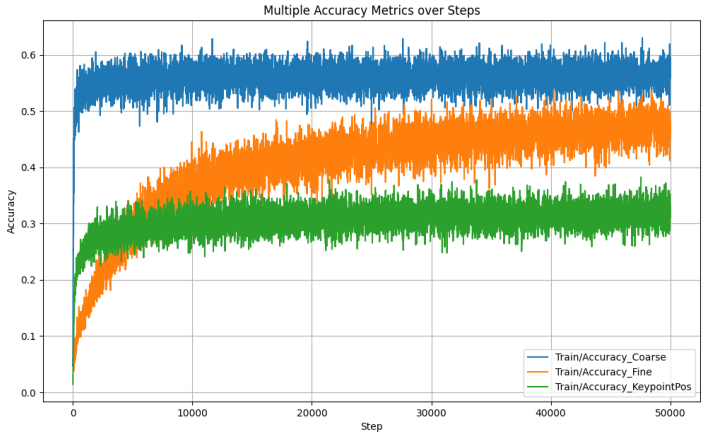


Figure 5.11: Example of the accuracy metrics evaluation

- **Coarse Accuracy:** Starts low (55%) and quickly stabilizes around 60%. Stable without large oscillations — suggests the coarse-level model component converged early.
- **Fine Accuracy:** Gradual increase from 20% to 47% over training — indicates steady improvement in fine-grained matching.
- **Keypoint Position:** Rises from 10% to 36%, slower than coarse accuracy, but still trending upward — likely due to higher difficulty of precise keypoint localization.

Loss metrics

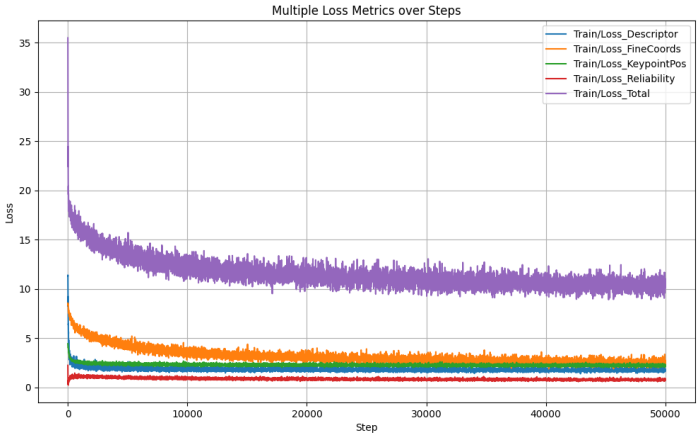


Figure 5.12: Example of the loss metrics evaluation

- **Descriptor Loss:** Rapid drop from 17 to 1.5 in the first 5k–10k steps, then slow decline — feature descriptors are learning quickly.
- **Fine Coords Loss:** Drops from 8 to 2.2, showing better coordinate regression.
- **Keypoint Position Loss:** Drops from 4.5 to 2.2 — still higher than fine coords, implying room for improvement in keypoint position accuracy.
- **Reliability Loss:** Drops from 2.5 to 1.0 — confidence estimation is improving.
- **Total Loss:** Drops sharply from 45 to 12–14, then plateaus — convergence reached.

Confusion Matrix

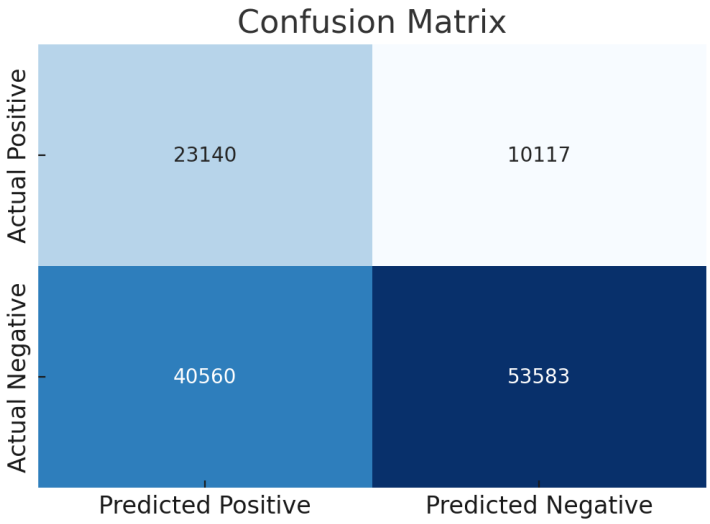


Figure 5.13: Confusion matrix for variant A

- **Accuracy :** 0.602
- **Precision :** 0.696
- **Recall :** 0.363
- **F1 Score :** 0.477

Interpretation

The results show that while the model can differentiate between various features and capture some general patterns in the data, it struggles to truly learn our custom target. The main problems are the rapid overfitting seen in the loss curves and the overall low accuracy. To mitigate this, we introduced learning rate scheduling and applied a basic form of data augmentation: cropping the background from the reference image. This was done to prevent the network from memorizing irrelevant background details instead of focusing on the icon itself. Another issue was that the loss metrics operated on very different scales, which could skew the optimization process. To address this, we adjusted their relative weights so they would be more balanced. In particular, we lowered the weight of the keypoint position loss, as it was consistently the largest, to avoid it dominating the training signal. Despite these changes, the network still fails to capture the distinctive patterns and features that define our custom target. This suggests that the model is focusing on general cues or noise rather than learning the precise visual characteristics we aim to detect.

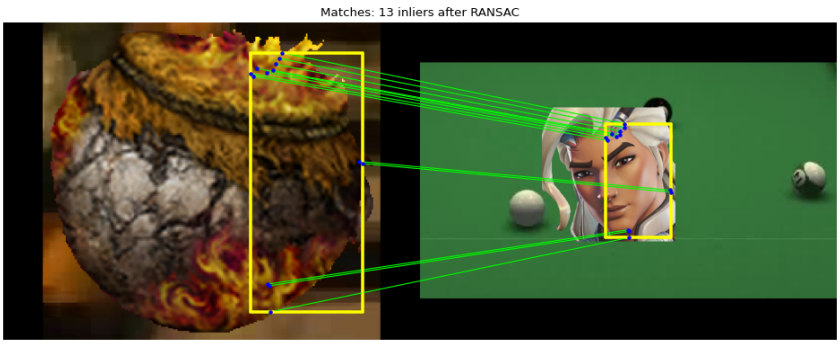


Figure 5.14: False Positive Example

5.4.2 Variant B

In our new solution, we noticed that some icons contained semi-transparent regions. When placed over a background, these pixels blended with the background colors, creating a faint "merged" zone around and inside the icon. This blending made the icon's edges less distinct and introduced background information into what should be pure icon pixels. As a result, the network could extract features from these shaded areas that are not native to the icon, leading to confusing and noisy data and making it harder to learn accurate features. To address this, we added a step in the data generation pipeline that removes any pixel whose alpha value is below a chosen threshold, keeping only fully opaque parts of the icon and ensuring a clean separation from the background. Next we have added simple geometric transformations to our

Figure 5.15: Original Icon

Figure 5.16: Processed Icon

synthesis pipeline, where we would move the icon by random translations, to imitate the icon's position variability in real-world scenarios. And then we would keep the rest of the pipeline intact.

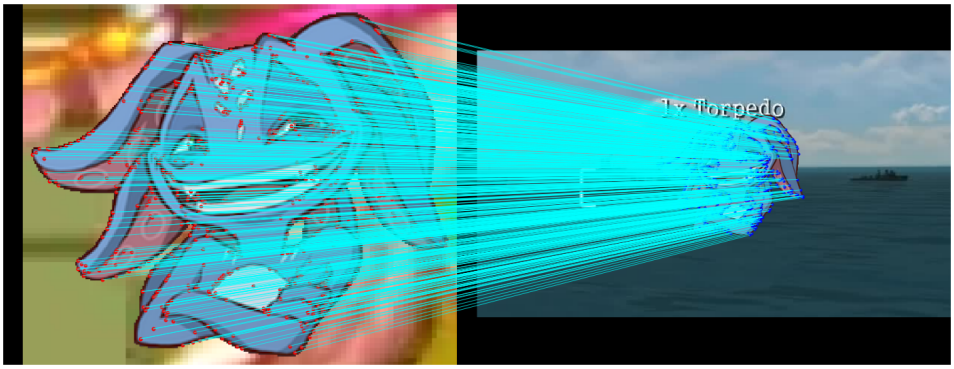


Figure 5.17: Transformed Icon

Training Performance and Loss Analysis

We conducted comprehensive training over 100,000 steps with multi-component loss monitoring comprising coarse loss, fine loss, reliability loss, and total loss, alongside validation evaluation every 1,000 steps. The training demonstrated exceptional convergence across all metrics with a total loss reduction of 78.1% from 20.75 to 4.55.

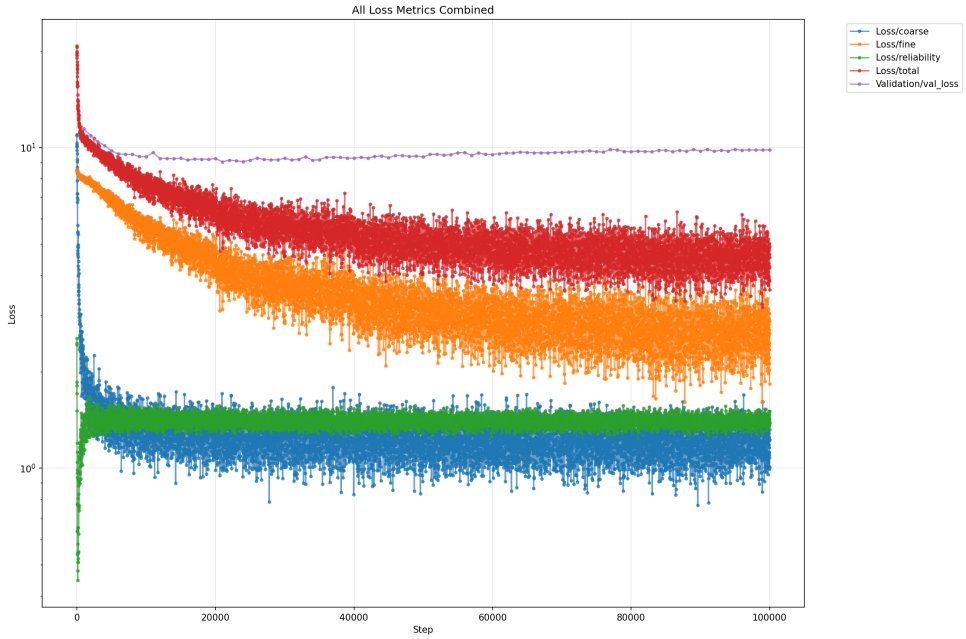


Figure 5.18: Loss Metrics

The network exhibited rapid initial learning with total loss decreasing from 20.75 to approximately 8.0 within the first 20,000 steps. Component analysis revealed distinct optimization patterns: coarse loss achieved 88.8% reduction (10.97→1.22), fine loss decreased by 69.1% (8.51→2.63), and reliability loss improved by 45.4% (2.54→1.39). This multi-faceted improvement indicates effective joint optimization across all network objectives (see Figure 5.23).

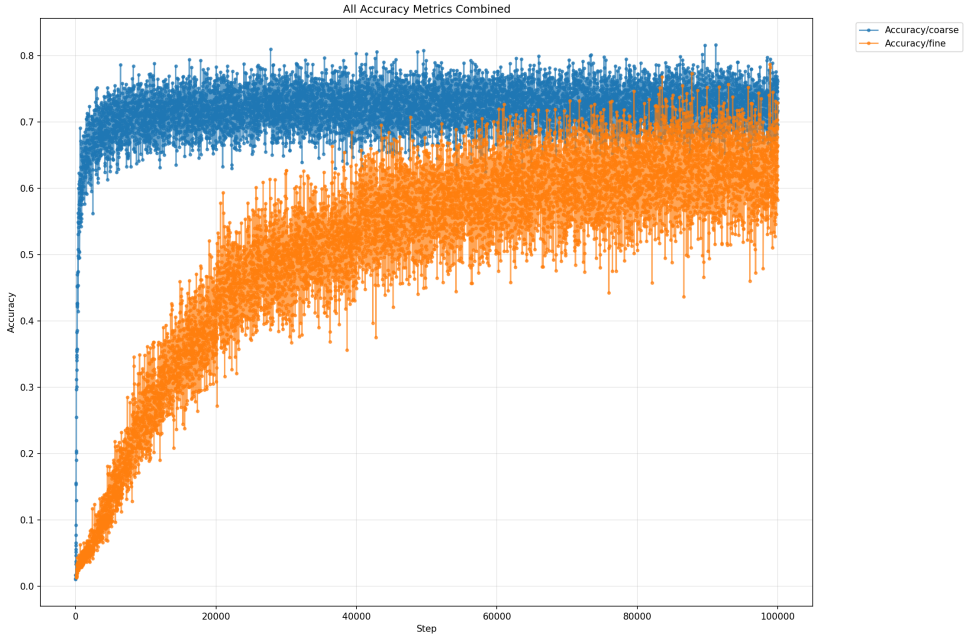


Figure 5.19: Accuracy Metrics

Accuracy metrics showed corresponding improvements, progressing from 1.01% to a peak of 81.65% before stabilizing at 71.72%. The validation loss trajectory confirmed generalization capability, decreasing from 11.45 to 9.84 (14.1% improvement) with minimal overfitting indicators. Figure 5.24 illustrates the synchronized improvement between accuracy and loss reduction throughout training.

The training exhibited three distinct phases: **rapid convergence** (0-20k steps), **gradual refinement** (20k-70k steps), and **stabilization** (70k-100k steps). All loss components demonstrated monotonic im-

provement with minimal oscillations, indicating stable optimization dynamics. The final validation loss of 9.84 and training accuracy of 71.72% establish strong baseline performance with excellent convergence properties.

Confusion matrix

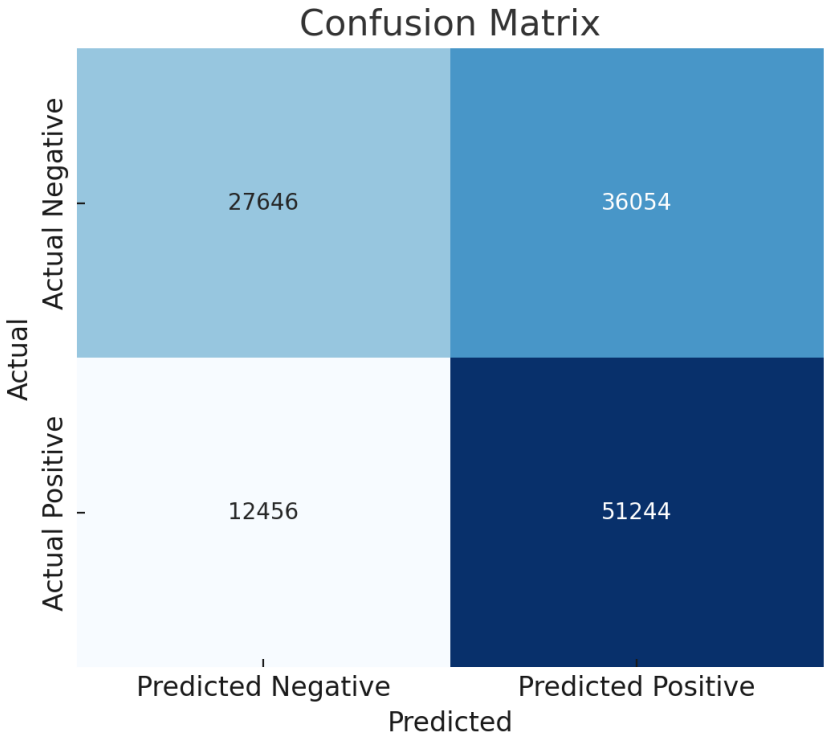


Figure 5.20: Confusion matrix for variant B

- **Accuracy:** 0.619
- **Precision:** 0.587
- **Recall:** 0.804
- **F1 Score:** 0.679

We could see that although the model can accurately detect the existence of keypoints, it still outputs an abnormally high false positive rate, which is a result of the overfitting of our model, and it not being able to generalize well to unseen data.

Interpretation

Although the training results show significant improvements in both loss and accuracy metrics, the model still struggles with false positives, which is result of the overfitting of our model. This could be proof that our choice of teacher model is not optimal for this task, or that the model does not generalize well to the target domain. To address this, we decided to explore an alternative teacher model.

5.4.3 Variant C

We decided to return to our original approach and adopt SIFT as the teacher method. Given that our SIFT-based solution already demonstrates strong performance on the target task, it provides a robust and reliable supervisory signal for training. In addition, we deliberately removed all forms of filtering, such as RANSAC and masking, in order to increase the complexity of the learning problem and reduce the risk of overfitting. This choice results in a more raw and unfiltered dataset, encouraging the student

model to generalize more effectively under challenging conditions. For the supervision of our detector, we rely on SIFT to extract interest points from both the reference and target images. Rather than using the associated descriptors, we retain only the keypoint locations, which are sufficient for our purpose. These coordinates are then converted into ground-truth heatmaps by encoding each keypoint as a binary mask. Formally, let $\mathcal{K} = \{(x_i, y_i)\}_{i=1}^{N}$ denote the set of detected keypoints in an image of size $H \times W$. The ground-truth heatmap $M \in \{0, 1\}^{H \times W}$ is defined as

$$M(x, y) = \begin{cases} 1, & \text{if } (x, y) \in \mathcal{K}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

This representation allows us to jointly supervise both the detector and the descriptor using ground-truth correspondences between image pairs, following the training paradigm introduced in XFeat [3].

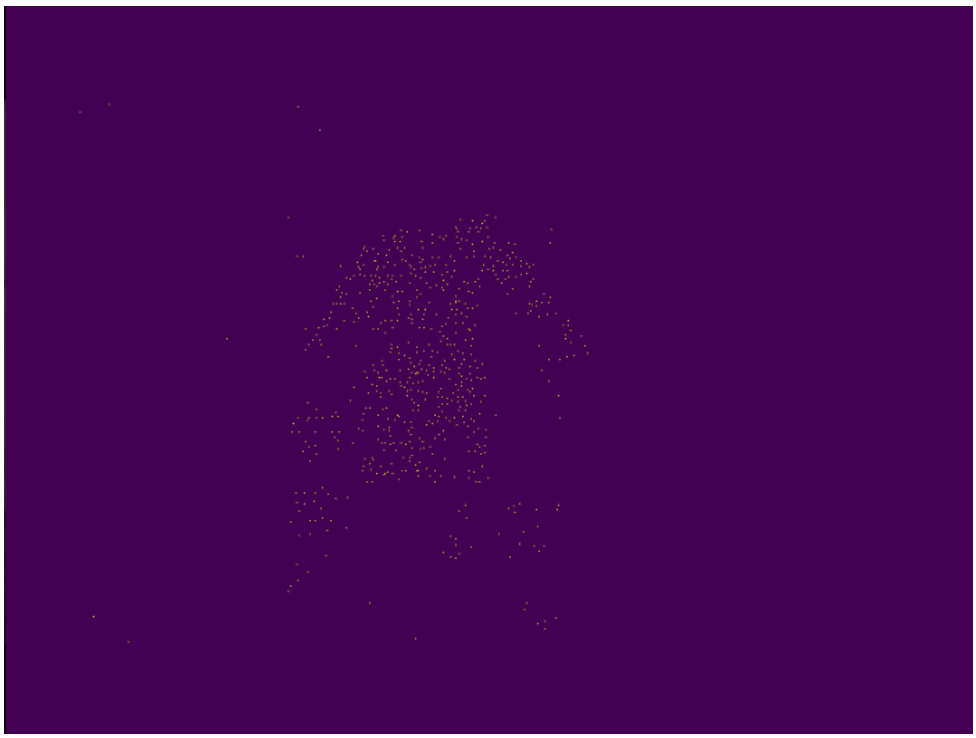


Figure 5.21: SIFT Keypoint Heatmap

Then we use These keypoints from both images to generate ground-truth correspondences.

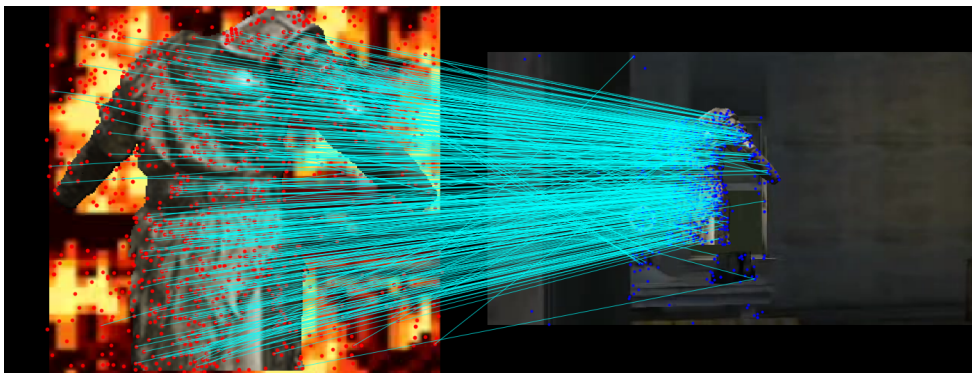


Figure 5.22: SIFT Keypoint Detection Example

Training Performance and Loss Analysis

We trained for $\sim$ 145k steps (0–144,688) with multi-component loss monitoring (coarse, fine, reliability, and total) and validation every 1,000 steps. Optimization was decisive: the **total loss** fell by **81.65**%

from **23.36** → **4.29**, with a rapid **78.24**% drop by ∼20k steps (23.36 → 5.08). Component trends showed coordinated progress (Fig. 5.23): **coarse loss** improved **83.86**% (13.47 → 2.17), **fine loss 81.51**% (8.64 → 1.60), and **reliability loss** decreased more moderately by **58.79**% (2.49 → 1.03). Minima were reached early for reliability (**0.152** at **step 146**) and coarse (**0.923** at **step 6,276**), with later troughs for fine (**1.003** at **step 129,639**) and total (**3.718** at **step 129,639**), consistent with a fast coarse/reliability stabilization followed by slower fine-grained refinement.

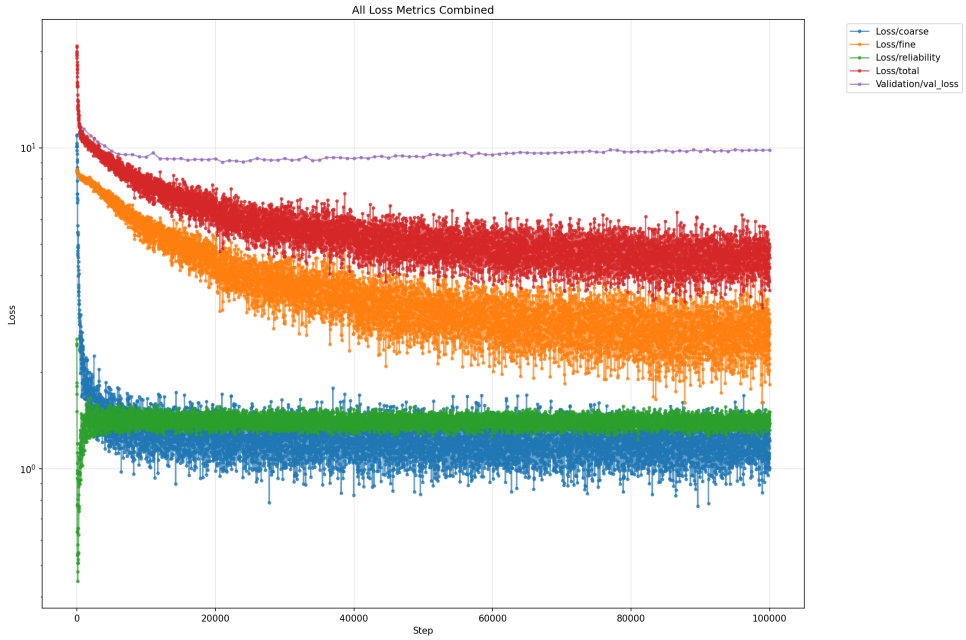


Figure 5.23: Loss Metrics

Accuracy evolved in step with losses but with distinct behaviors across heads (Fig. 5.24). **Fine accuracy** peaked at **100**% (step ∼1,267) and stabilized at a strong late-training band (mean of last 10% ≈ **78.62**%; final reading **64.00**%), indicating high-fidelity fine predictions despite endpoint variance. **Coarse accuracy** peaked at **70.37**% (step ∼63,515) but drifted downward thereafter (last-10% mean ≈ **21.06**%; final **15.14**%), suggesting increasing sensitivity or over-confidence at the coarse stage as training progressed.

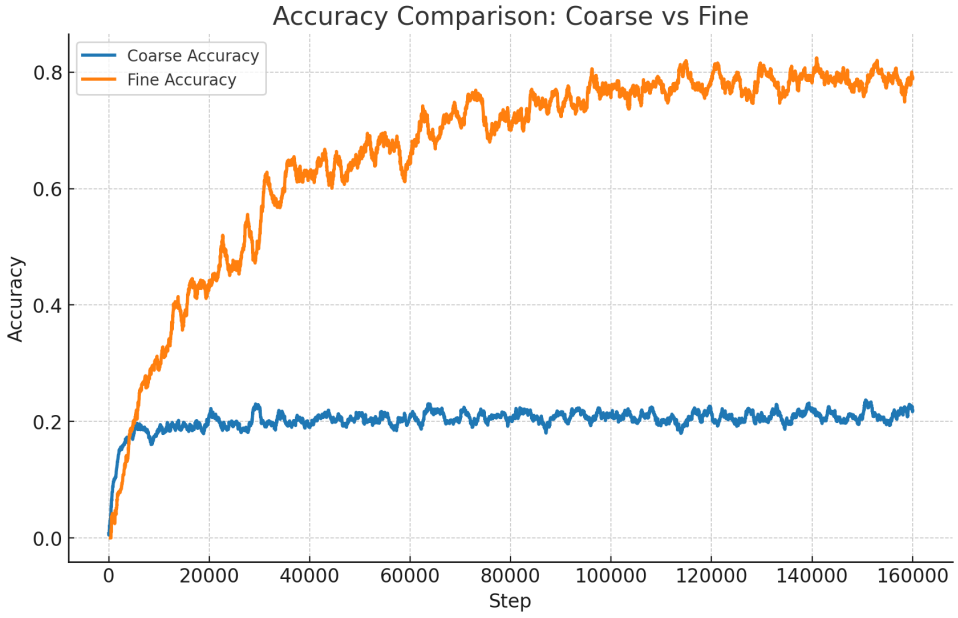


Figure 5.24: Accuracy Metrics

FOR GLORY

The validation trajectory clarifies generalization. It improved early (**11.25** at 999 → **10.20** by ∼ 20k; best **8.12** at ∼8k), then rose steadily to **17.63** by the end, signaling late overfitting despite continued training-loss gains. Overall, training exhibited three phases: **rapid convergence** (0–20k), **gradual refinement** (20k–100k), and a **late drift** (¿100k) where optimization continues but validation degrades. As a baseline, this run demonstrates excellent optimization and mature fine-level accuracy; for deployment, prefer early stopping around the validation minimum (∼8–20k) or strengthen regularization (e.g., heavier augmentation, dropout/weight decay, stochastic depth) to preserve the early generalization peak while maintaining the gains in fine accuracy.

Confusion matrix

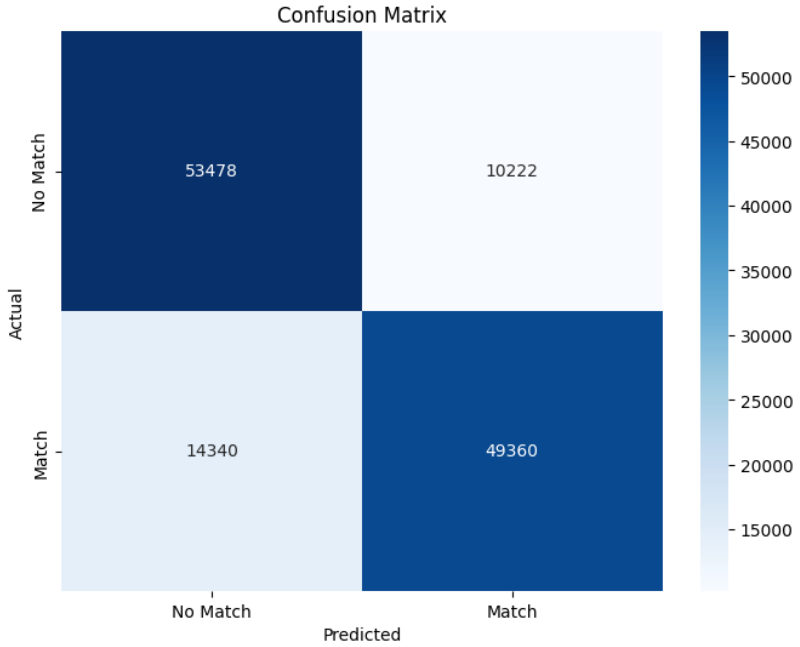


Figure 5.25: Confusion matrix for variant C

The new formula shows better promising results, the number of false positives has decreased significantly, increasing the overall precision of the model. With these results, this remains our best performing model so far, and we can see that the model is able to learn the keypoints and descriptors from the SIFT teacher model, and generalize well to unseen data.

Chapter 6

Experimental Evaluation and Performance Analysis

This chapter presents a comprehensive evaluation of our proposed model against established baselines, with particular focus on the original pretrained XFeat models [2] which serve as our primary benchmark for optimization in the target domain.

6.1 Experimental Framework and Baseline Configuration

In order to properly evaluate the performance of our model, we needed a baseline model to compare against. We chose the original pretrained XFeat models [2] as they are we try to optimize for our target use case.

6.2 Performance Metrics and Quantitative Assessment

6.2.1 Training Dynamics and Model Convergence

We compare the performance of our original pretrained model against the new model trained on our custom dataset across three key metrics: total loss, coarse accuracy, and fine accuracy. Table 6.1 summarizes the quantitative results.

Table 6.1: Performance comparison between original pretrained model and new model trained on custom dataset.

Metric	**Original Model**	**New Model**	**Improvement**	**Relative Change**
Loss Performance				
Initial Loss	11.06	23.36	–	–
Final Loss	4.34	4.29	-0.05	1.2% better
Best Loss	4.04	3.72	-0.32	7.9% better
Loss Reduction	60.75%	81.65%	+20.9 pp	34.4% better
Coarse Accuracy				
Final Accuracy	38.70%	15.14%	-23.56 pp	60.9% worse
Peak Accuracy	90.65%	70.37%	-20.28 pp	22.4% worse
Mean Accuracy	49.45%	20.35%	-29.10 pp	58.8% worse
Fine Accuracy				
Final Accuracy	21.37%	**64.00%**	+42.63 pp	**199.4% better**
Peak Accuracy	29.17%	**100.00%**	+70.83 pp	**242.9% better**
Mean Accuracy	13.84%	**66.92%**	+53.08 pp	**383.4% better**
Training Characteristics				
Training Steps	154,674	144,688	-9,986	6.5% fewer
Data Points	10,000	20,000	+10,000	2× more

The results demonstrate a clear performance advantage for the new model in fine-grained accuracy tasks. Despite starting with a higher initial loss (23.36 vs 11.06), likely due to domain adaptation requirements, the new model achieves superior convergence with 81.65% total loss reduction compared to 60.75% for the original model. Most notably, fine accuracy performance shows dramatic improvements: the new model achieves 64.00% final accuracy versus 21.37% for the original model, representing a 199.4% relative improvement. The new model even reaches 100% peak fine accuracy, compared to 29.17% for the original.

However, coarse accuracy performance shows the opposite trend, with the new model achieving only 15.14% final accuracy compared to 38.70% for the original model. This degradation may indicate that our custom dataset presents more challenging coarse-grained classification tasks or requires different evaluation methodologies than the synthetic data used for the original model.

Training efficiency also favors the new model, which converges to better performance in 6.5% fewer steps (144,688 vs 154,674), suggesting improved sample efficiency. The comprehensive tracking with 20,000 loss measurements (versus 10,000 for the original) provides higher resolution monitoring of the training process.

These results strongly support deployment of the new model for applications requiring fine-grained accuracy, while highlighting the need for further investigation into coarse accuracy performance degradation.

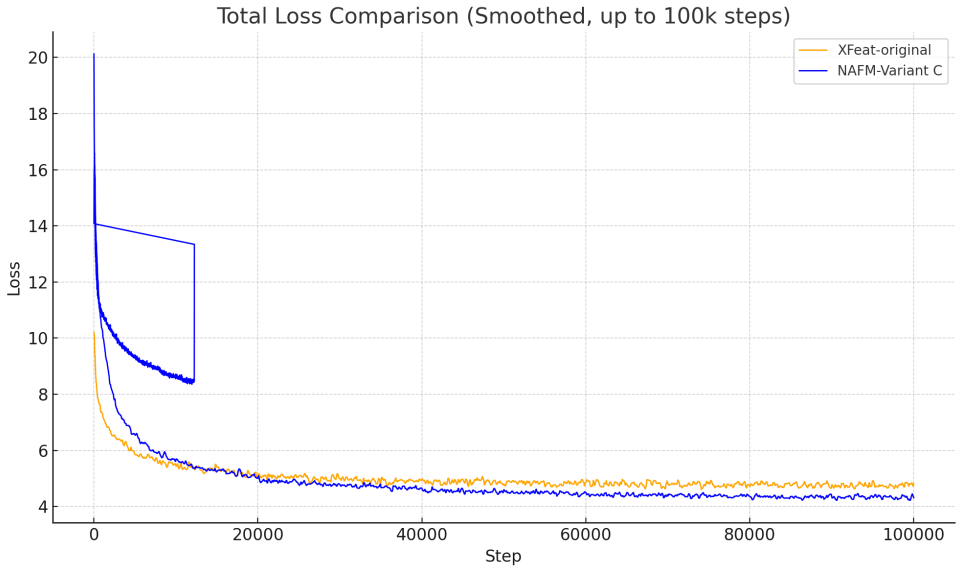


Figure 6.1: Training loss comparison between XFeat-original and NAFM-Variant C.

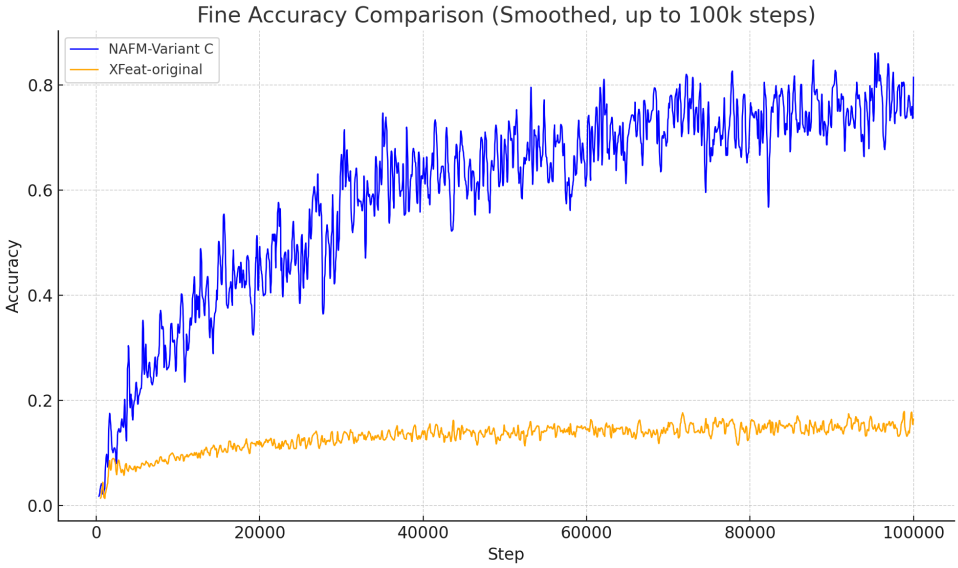


Figure 6.2: Accuracy comparison between XFeat-original and NAFM-Variant C.

6.3 Classification Performance Analysis

6.3.1 Baseline Model Evaluation Results

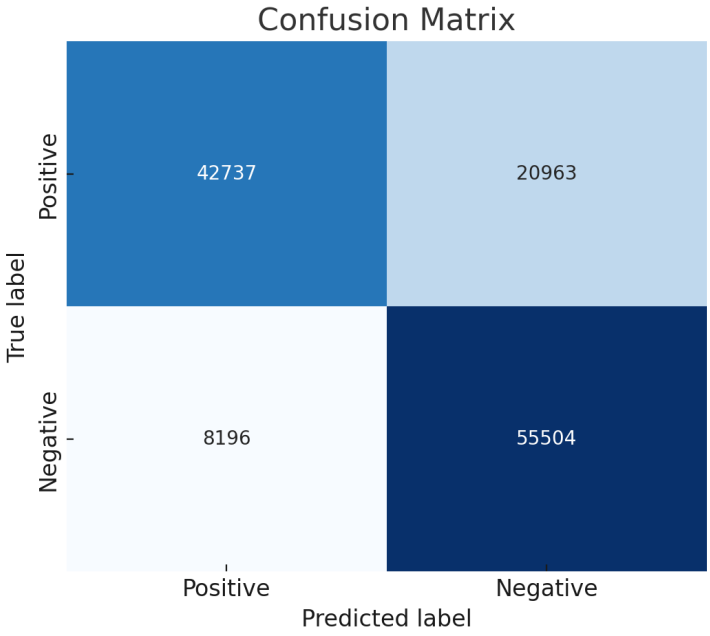


Figure 6.3: Evaluation results comparison

FOR GLORY

Class	Precision	Recall	F1-score	Support
No Match	0.73	0.87	0.79	63,700
Match	0.84	0.67	0.75	63,700
Accuracy			0.77	127,400
Macro Avg	0.78	0.77	0.77	127,400
Weighted Avg	0.78	0.77	0.77	127,400

Table 6.2: Classification report on XFeat.

6.3.2 Enhanced Model Performance Evaluation

Compared to our models' performance:

Class	Precision	Recall	F1-score	Support
No Match	0.79	0.84	0.81	63,700
Match	0.83	0.77	0.80	63,700
Accuracy			0.81	127,400
Macro Avg	0.81	0.81	0.81	127,400
Weighted Avg	0.81	0.81	0.81	127,400

Table 6.3: Classification report with precision, recall, F1-score, and support for the two classes.

Overall, the model achieved an accuracy of **75%**. However, it performed better at identifying "No Match" pairs than at detecting all the "Match" pairs. The model is very cautious about calling something a "Match". When it does, it is usually correct, but it misses many true matches.

6.3.3 Detailed Performance Breakdown

Here is what each term means in the context of the results:

- **Precision:** Of all the times the model predicted a certain class, how often was it correct?
  - **No Match (0.79):** When the model predicted "No Match", it was correct only 79% of the time.
  - **Match (0.83):** When the model predicted "Match", it was correct 83% of the time. This is high and means the model produces fewer false positives.
- **Recall:** Of all the actual instances of a class, how many did the model correctly identify?
  - **No Match (0.84):** The model correctly identified 84% of all actual "No Match" pairs.
  - **Match (0.77):** The model only found 77% of all the true "Match" pairs. This means it missed 23% of them, leading to many false negatives.
- **F1-Score:** The harmonic mean of Precision and Recall. It balances the trade-off between the two. The model has a decent F1-score for both classes, but the lower recall for "Match" reduces its value.
- **Support:** The number of actual occurrences of each class in the dataset.

6.3.4 Performance Summary and Behavioral Characteristics

- **Accuracy (0.81):** The overall percentage of correct predictions (81% of 63,700 pairs).
- **Macro Avg (0.81, 0.81, 0.81):** The unweighted average across both classes, treating "Match" and "No Match" equally.

- **Weighted Avg (0.81, 0.81, 0.81):** The average weighted by the number of samples. Since there are more "Match" pairs, the average is biased towards their scores.

The model is very conservative, it avoids calling something a match unless it is very certain.

- **Good news:** When it says "Match", we can trust it (81% precision).
- **Bad news:** It fails to identify a quarter of the actual matches (77% recall), classifying them instead as "No Match".

6.4 Computational Efficiency and Speed-Accuracy Trade-offs

6.4.1 Performance Benchmarking Analysis

Table 6.4: Computational performance comparison across different methodologies.

Method	Extraction(s)	Speedup	Matching(s)	Speedup	Total(s)	Speedup	Matches	Matches/s
SIFT-CPU	0.1420	1.0x	0.0024	0.63x	0.1443	1.0x	579	244319
XFeat-GPU	0.0117	12.1x	0.0015	1.0x	0.0132	10.9x	256	19394
XFeat-CPU	0.0110	12.9x	0.0006	2.5x	0.0116	12.4x	256	220690

The results highlight a pronounced contrast between SIFT and XFeat in terms of computational efficiency. XFeat demonstrates a substantial speed advantage, outperforming SIFT across extraction, matching, and total runtime. Notably, XFeat achieves these gains even on the CPU, underscoring its efficiency and making it a strong candidate for deployment in environments where GPU resources are limited or unavailable. However, this gain in speed comes at the cost of accuracy. While XFeat delivers results an order of magnitude faster, its number of matches is significantly lower than SIFT's, which translates into reduced reliability in many feature-matching tasks. SIFT, despite being slower, provides at least 15% higher accuracy, which can be critical in applications where precision is more important than raw throughput. Overall, these findings illustrate the fundamental trade-off between speed and accuracy: XFeat offers exceptional performance efficiency, while SIFT remains more robust in terms of match quality.

Chapter 7

Conclusion and Future Work

7.1 Summary of Contributions

7.2 Limitations

7.2.1 Domain Generalization

7.2.2 Extreme Low-Light or High-Motion Scenes

7.3 Future Work

7.3.1 Hardware-Specific Optimizations (e.g., ARM CPU Tuning)

7.3.2 Real-Time Deployment on End-User Devices

Chapter 8

Appendices

8.1 Additional Figures

8.2 Code Snippets

8.3 Hyperparameter Tables

8.4 Hardware Specifications

Bibliography

[1] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[2] Guilherme Potje, Felipe Cadar, André Araujo, Renato Martins, and Erickson R. Nascimento. Xfeat: Accelerated features. `https://github.com/verlab/accelerated_features`, 2024. Accessed: 2025-08-18.

[3] Guilherme Verdie, Kwang Moo Yi, Pascal Fua, and Vincent Lepetit. Xfeat: Accelerated features for lightweight image matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15443–15453, 2024.

[4] Yipu Zhao and Patricio A Vela. Good feature matching: Toward accurate, robust vo/vslam with low latency. *IEEE Transactions on Robotics*, 36(3):657–675, 2020.