**BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA**
**FACULTY OF MATHEMATICS AND COMPUTER SCIENCE**
**SPECIALIZATION COMPUTER SCIENCE IN ENGLISH**

# DIPLOMA THESIS

# NeRF-Augmented Training of Vision Transformers (ViTs)

**Supervisor**
**Professor, PhD. Darabant Sergiu Adrian**

*Author*
*Pop Denis-Vasile*

2025

**UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA**
**FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ**
**SPECIALIZAREA INFORMATICĂ ÎN LIMBA ENGLEZĂ**

# LUCRARE DE LICENŢĂ

# Antrenarea Augmentată cu NeRF a Transformatoarelor Vizuale (ViTs)

**Conducător științific**
**Profesor, Dr. Darabant Sergiu Adrian**

*Absolvent*
*Pop Denis-Vasile*

2025

ABSTRACT

Neural Radiance Fields (NeRF) have emerged as a way to generate photorealistic images with depth, offering a source of synthetic ground-truth data for training vision models. This thesis investigates leveraging NeRF-generated synthetic Red-Green-Blue-Depth (RGB-D) data to augment the fine-tuning of Transformer-based vision models, focusing on two use cases: dense stereo feature matching using Tiny RoMa and monocular depth estimation using Depth-Anything-V2. By integrating NeRF-rendered data into existing RGB-D datasets, I investigate how synthetic augmentation influences the training of Transformer-based models. To support this, I created a custom dataset by augmenting real-world indoor scenes from the 7-Scenes dataset and fully synthesizing scenes from NeRF-Stereo, using Instant Neural Graphics Primitives (Instant-NGP) and post-processing effects to improve data quality. This setup enabled controlled experiments on both partially augmented and fully synthetic datasets.

Under the experimental conditions of this thesis, NeRF-based data proved modestly helpful for stereo matching tasks, with a slight improvement in coarse matching accuracy when a small proportion of synthetic frames were included during training. However, models trained entirely on synthetic data exhibited poor generalization. For monocular depth estimation, no benefits were observed from NeRF augmentation, and performance decreased when synthetic data was included, by approximately 25% in the case of hybrid training, especially for larger Vision Transformer (ViT) architectures. These findings suggest that while NeRF-generated data holds promise, its effectiveness depends strongly on model type, task, and data balance.

To complement the experiments, I developed an interactive application using React, FastAPI, and Gradio, which enables visual testing of the fine-tuned models via a clean application programming interface (API). This tool facilitates further experimentation and qualitative assessment. Overall, the study shows that NeRF-generated synthetic data can support Transformer-based vision training in specific contexts, but careful integration and evaluation are necessary to ensure consistent benefits.

# Contents

# Chapter 1

# Introduction

## 1.1 Context and Motivation

Understanding spatial structure from images is a cornerstone of computer vision, underpinning applications such as 3D reconstruction, robotics, Augmented Reality (AR), and autonomous navigation. Two essential tasks in this domain are **depth estimation**, which predicts the distance of scene elements from the camera, and **feature matching**, which identifies correspondences between images (often serving as a foundation for stereo reconstruction and visual localization).

Traditionally, **depth estimation** has been approached through stereo matching, which infers depth by detecting and triangulating corresponding features across image pairs. More recently, **monocular depth estimation** (predicting depth from a single image) has gained prominence due to its advantage in scenarios where stereo data is not available. In parallel, **feature matching** has evolved from hand-crafted descriptors to dense, learning-based approaches, enabling correspondence even under significant viewpoint or illumination changes.

Both tasks have benefited from advances in deep learning, with **Convolutional Neural Network (CNN)**-based approaches gaining early popularity and, more recently, Transformer-based architectures such as **Vision Transformer (ViT)s** gaining traction. However, these models require large amounts of diverse and accurately labeled data, which is often costly or impractical to obtain.

**Neural Radiance Fields (NeRF)** have emerged as a powerful solution for synthesizing novel views from sparse image sets, capturing both photometric appearence and 3D geometry. Modern implementation like Instant Neural Graphics Primitives (Instant-NGP) allow rapid training and high-fidelity rendering, making NeRF a compelling tool for generating **synthetic Red-Green-Blue-Depth (RGB-D) data** in scenarios where real-world ground truth is limited or unavailable.

This thesis explores the use of NeRF-generated synthetic data to train Transformer-based models for spatial understanding tasks. Specifically, it investigates the impact of NeRF-based supervision on two architectures: Tiny Robust Matcher (RoMa), a lightweight dense feature matcher for stereo correspondence, and Depth-Anything-V2, a state-of-the-art monocular depth estimation model. Through experiments across partially and fully synthetic datasets, this work evaluates the feasibility, limitations, and benefits of using NeRF as a data augmentation strategy in training vision transformers.

## 1.2 Objectives

The primary objective of this thesis is to evaluate the utility of NeRF-generated data in training Transformer-based models for depth estimation and feature matching. The specific goals include:

- Train NeRF models using **nerf-template** on real datasets.
- Generate synthetic RGB-D data and stereo image pairs from the trained NeRF models.
- Post-process the NeRF-generated data to enhance quality and mitigate artifacts.
- Fine-tune and evaluate:
  - **Tiny RoMa** for dense feature matching in stereo images.
  - **Depth-Anything-V2** for monocular depth estimation.
- Analyze the impact of NeRF-augmented data on model performance, generalization, and training efficiency.
- Create an app that allows the user to interact with the fine-tune models using **React**, **FastAPI**, and **Gradio**.

## 1.3 Personal Contributions

The contributions of this thesis are as follows:

- Trained NeRF models on subsets of the **7-Scenes** and **NeRF-Stereo** datasets using the **nerf-template** framework.

- Created a partially synthetic dataset by augmenting real **7-Scenes** sequences with NeRF-generated frames, resulting in 5 scenes variable percentages of NeRF augmentation: 0.83% – 10%
- Created a fully synthetic dataset by utilizing NeRF to generate RGB-D pairs of two scenes from **NeRF-Stereo** (0080 and 0097)
- Fine-tuned and evaluated **Tiny RoMa** and **Depth-Anything-V2** models on both real and NeRF-augmented data.
- Conducted quantitative and qualitative experiments to assess the benefits and limitations of NeRF-based supervision.
- Implemented a demo application using **React**, **FastAPI**, and **Gradio** to visualize inference results from the trained models.

## 1.4 Thesis Structure

This thesis is organized into eight chapters, each addressing a core aspect of the research:

- **Chapter 1** introduces the motivation, objectives, contributions, and outlines the use of AI-assisted tools in the writing process.
- **Chapter 2** presents theoretical foundations in computer vision, including depth estimation, feature detection and matching, NeRF, and Transformer-based vision models.
- **Chapter 3** reviews related work on depth prediction, feature matching, and synthetic data, with a focus on **RoMa** and **Depth-Anything-V2**.
- **Chapter 4** details the generation and post-processing of synthetic RGB-D datasets using NeRF and the **nerf-template** framework.
- **Chapter 5** presents the fine-tuning of **Tiny RoMa** on both NeRF-generated partial and fully synthetic datasets, comparing different model variants.
- **Chapter 6** presents the fine-tuning of **Depth-Anything-V2** on both NeRF-generated partial and fully synthetic datasets, comparing different model variants.
- **Chapter 7** introduces a full-stack application for model inference, including a **FastAPI** backend, **React** frontend, and **Gradio** interface.
- **Chapter 8** concludes with a summary of results, discussion of limitations, and potential directions for future work.

## 1.5 Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author used ChatGPT by OpenAI in order to assist with idea structuring, improving clarity and grammar, analyzing data, and structuring data into tables. After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the thesis.

# Chapter 2

# Theoretical Foundations

## 2.1 Computer Vision Fundamentals

**Computer vision** allows machines to see, interpret, and process visual information from the world. A foundational concept is the projection of 3D scenes onto a 2D image, which is called image formation and is typically modeled using simplified camera models such as the pinhole camera. While not explicitly addressed in this thesis, the principles of image formation support tasks like feature matching, depth estimation, and 3D reconstruction, which are central to this work.

An essential concept is **depth estimation**, which refers to the process of inferring the distance between objects in a scene and the camera. This can be achieved through a series of techniques that will shortly be discussed.

**Feature detection and matching** is another critical concept which involves identifying distinctive points or patterns in images and establishing matches between them. Hand-made algorithms such as Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF) and Oriented FAST and Rotated BRIEF (ORB) are commonly used for this purpose. These methods detect keypoints and compute descriptors that are invariant to changes in scale, rotation, and illumination, facilitating robust matching in different images [Ken21].

**Camera calibration** is the process of determining intrinsic parameters (focal length, optical center, etc.) and extrinsic parameters (the camera's position and orientation in space) that define the mappings between the 3D world coordinates and the 2D image coordinates. Accurate calibration is extremely important for tasks like 3D reconstruction and augmented reality applications [Elg05].

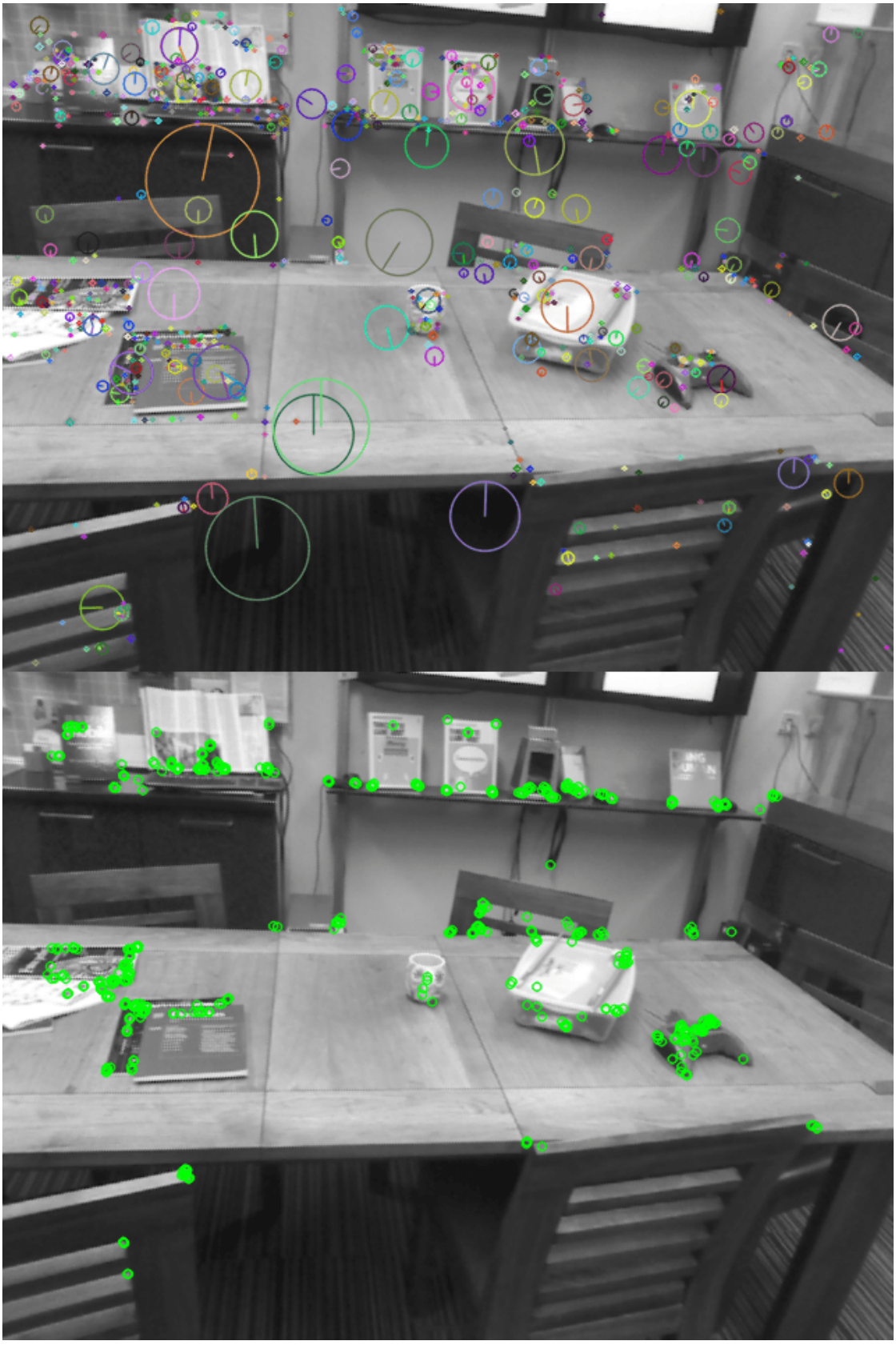

Figure 2.1: Comparison of keypoint detection: SIFT (top) and ORB (bottom). SIFT identifies more robust and dense keypoints suitable for tasks requiring high precision, while ORB detects fewer but computationally efficient features.

**Depth estimation** involves inferring the distance of objects from the camera. This can be achieved through various methods:

- *Stereo vision* uses two or more images taken from different viewpoints to compute depth based on disparities between corresponding points.
- *Monocular depth estimation* predicts depth from a single image by leveraging cues such as texture gradients, perspective, and learned patterns.
- *Multi-View Stereo (MVS)* combines multiple images to reconstruct detailed 3D structures, enhancing accuracy and completeness.
- *Active sensors*, like LiDAR or structured light systems, directly measure depth by emitting signals and analyzing their reflections.

These concepts form the backbone of more advanced topics such as multi-view stereo, Structure-from-Motion (SfM), and neural rendering techniques.

## 2.2 Multi-View Stereo and COLMAP

### 2.2.1 Multi-View Stereo (MVS)

**MVS** is a technique that aims to reconstruct detailed 3D structures from a plethora of images captured from different viewpoints. Unlike traditional stereo vision, which typically uses two images, MVS utilizes a larger set of images to improve the 3D reconstruction process in both accuracy and completeness. It involves estimating depth for each pixel by finding correspondences across images, ensuring photometric consistency (i.e., the appearance of points remains consistent across different views), and building dense 3D models from the estimated depth maps [Wal20].

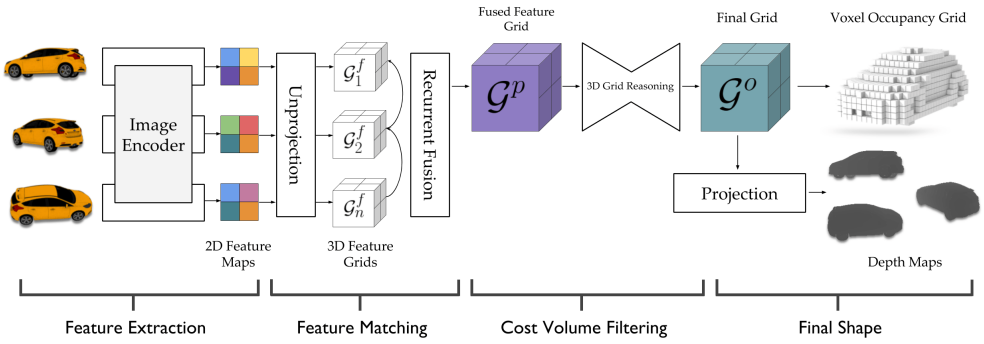


Figure 2.2: Overview of a typical MVS pipeline [KHM17].

### 2.2.2 Structure-from-Motion (SfM)

**SfM** is a technique that simultaneously estimates camera poses and 3D structure from a set of unordered images. The typical SfM pipeline includes:

1. **Feature detection and matching**: Identifying keypoints and establishing correspondences between images.
2. **Initial pose estimation**: Computing relative camera positions using matched features.
3. **Incremental reconstruction**: Gradually adding images to refine camera poses and 3D points.
4. **Bundle adjustment**: Optimizing the entire reconstruction to minimize reprojection errors [Pro19].

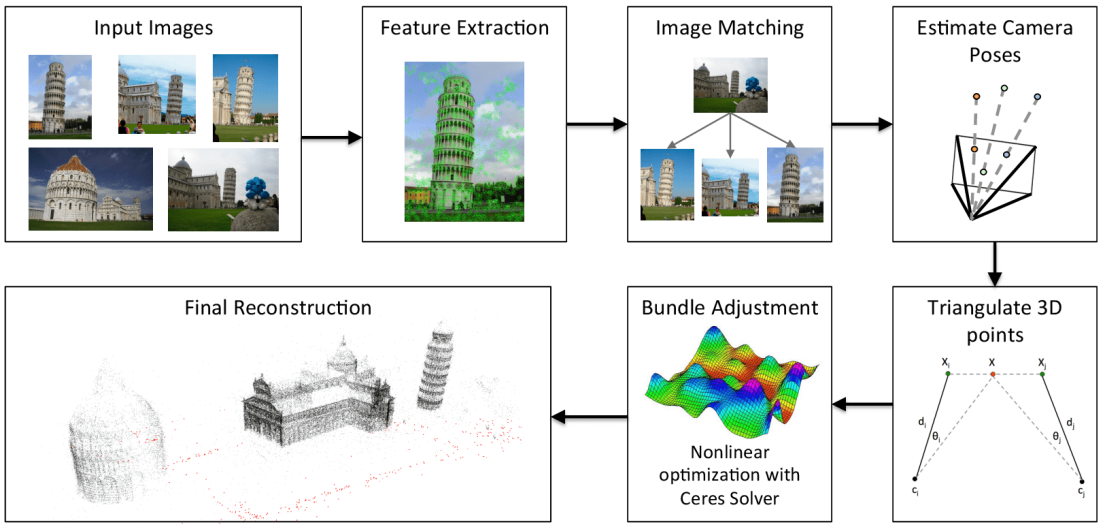


Figure 2.3: Stages of a typical SfM workflow [SHT15].

SfM is foundational for applications requiring 3D reconstruction from images, such as photogrammetry and robotics.

### 2.2.3 COLMAP

**COLMAP** is an open-source software that implements both SfM and MVS pipelines. It provides an extensive array of tools for 3D reconstruction, including feature extraction and matching using algorithms like SIFT, sparse reconstruction to build an initial 3D model using SfM, and dense reconstruction to refine the initial model with MVS for detailed surface geometry [Sch20].

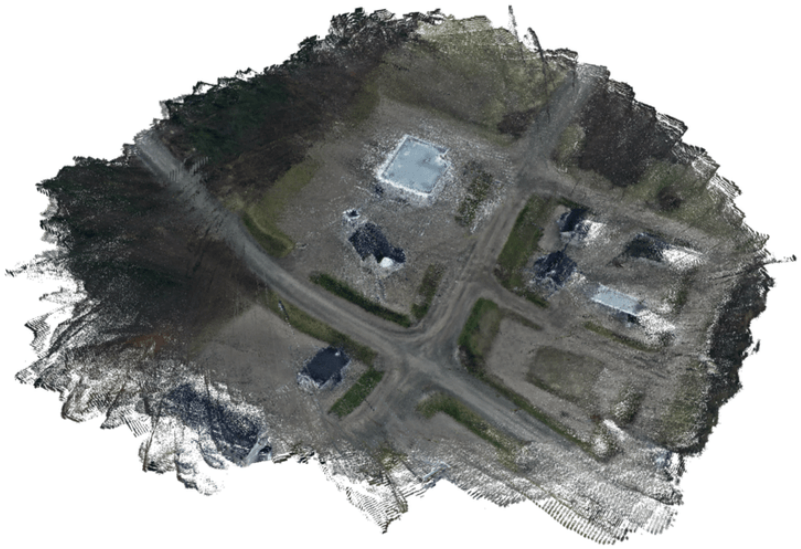

Figure 2.4: COLMAP interface showing sparse and dense reconstruction output [Sch20].

## 2.3 Monocular Depth Estimation

**Monocular depth** estimation refers to the process of predicting depth from a single image. This is inherently an ill-posed problem due to the lack of stereo cues. However, recent advancements in deep learning have enabled significant progress in this area. Techniques include supervised learning (where models are trained on datasets with ground-truth depth information), self-supervised learning (which leverages image reconstruction losses to learn depth without explicit ground truth), and Transformer-based models that utilize attention mechanisms to capture global context for improved depth estimation [Hug23].

Monocular depth estimation is crucial for applications where stereo setups are impractical, such as mobile robotics and augmented reality.

## 2.4 Neural Radiance Fields (NeRF)

**NeRF** represent a scene as a continuous 5D function that outputs color and density for any given 3D location and viewing direction. Key aspects include volumetric rendering (which integrates color and density along rays to synthesize views), scene representation (capturing fine details and complex geometry with a compact neural network), and training requirements, as NeRF requires accurate camera poses and a sufficient number of images for effective training [MST+21].

NeRF has revolutionized view synthesis and 3D reconstruction, enabling photo-realistic rendering from sparse image sets.

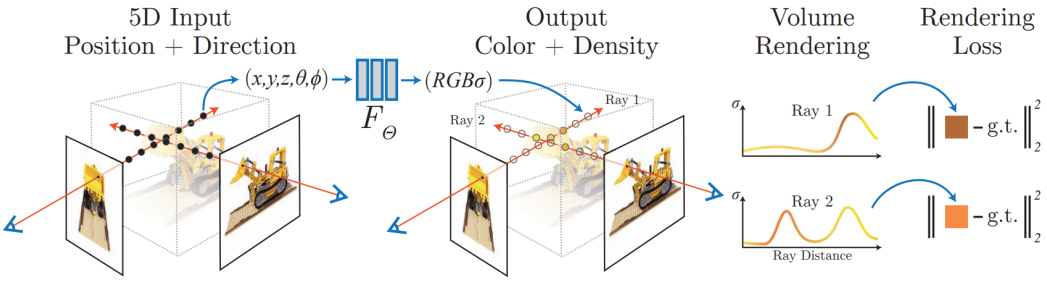


Figure 2.5: Diagram of NeRF ray tracing and volume rendering process [XTS+22].

### 2.4.1 Instant Neural Graphics Primitives (Instant-NGP)

**Instant-NGP** is a framework that accelerates the training and rendering of NeRF models. It introduces multi-resolution hash encoding (efficiently encoding spatial information to speed up learning), optimized CUDA kernels (for leveraging GPU acceleration for real-time performance), and support for various primitives extending beyond NeRF to include Signed Distance Functions (SDFs) and other representations [MESK22].

Instant-NGP significantly reduces the computational resources required for NeRF training, making it accessible for a broader range of applications.

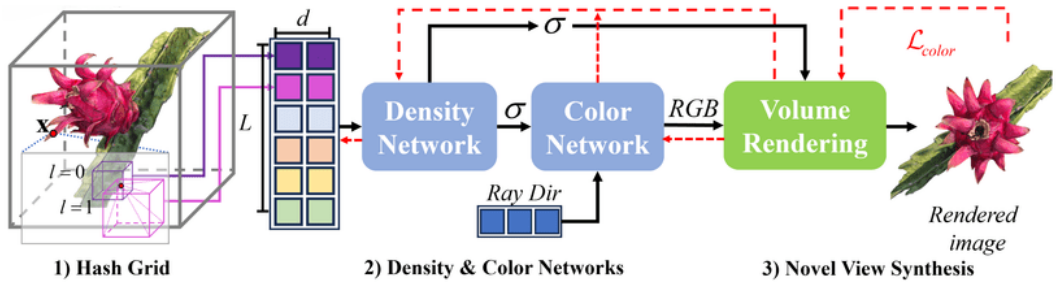


Figure 2.6: Hash encoding scheme used in Instant-NGP to accelerate rendering [HYP+24].

## 2.5 Transformer-Based Vision Models

Transformers, originally developed for natural language processing tasks, have been successfully adapted for computer vision, leading to the development of the ViT architecture. Introduced by Dosovitskiy et al. in 2020 [DBK+20], ViTs have demonstrated remarkable performance across various visual tasks, often surpassing traditional CNNs architectures.

### 2.5.1 Architecture of Vision Transformers

The **ViT** architecture processes images by dividing them into fixed-size patches, typically of size $16 \times 16$ pixels. Each patch is then flattened and linearly projected into a lower-dimensional embedding space. These embeddings are supplemented with positional encodings to retain spatial information and are then fed into a standard transformer encoder composed of multi-head self-attention and feed-forward layers [DBK+20].

Unlike CNNs, which inherently capture local spatial hierarchies through convolutional operations, ViTs rely on self-attention mechanisms to model global relationships between image patches. This allows ViTs to capture long-range dependencies and contextual information more effectively, which is particularly beneficial for complex visual understanding tasks [NRK+21].

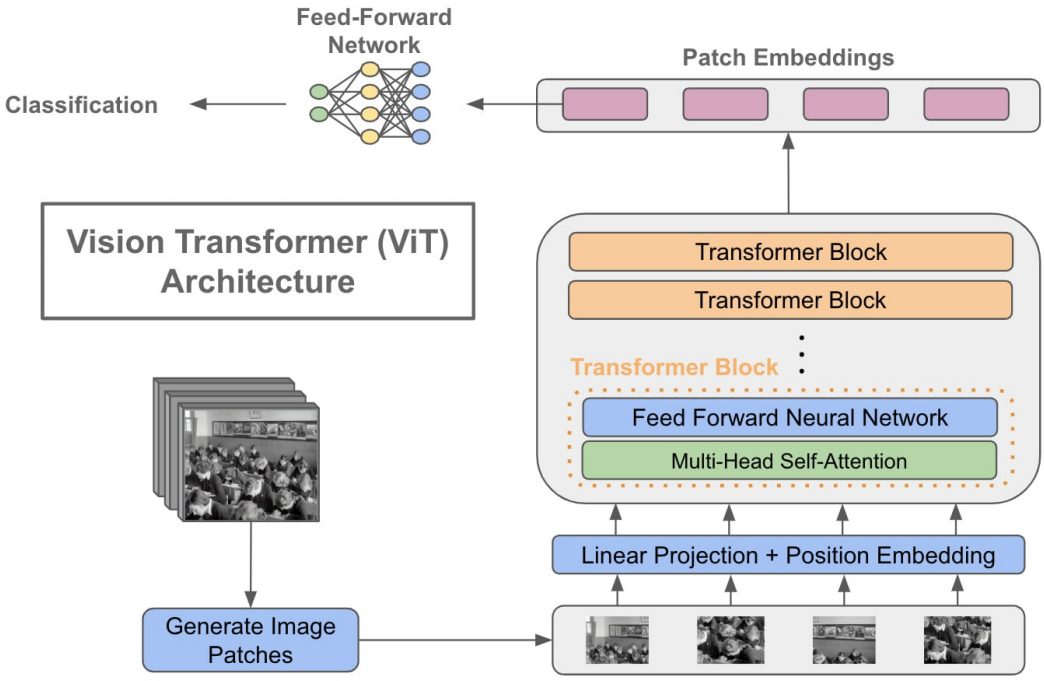


Figure 2.7: Architecture of the ViT [Kum24].

### 2.5.2 Advantages and Challenges

One of the significant advantages of ViTs is their ability to model global context without the inductive biases inherent in CNNs. This flexibility allows ViTs to achieve superior performance in tasks such as image classification, object detection, and semantic segmentation, especially when trained on large-scale datasets [DBK+20].

However, ViTs also present certain challenges. They typically require large amounts of training data to achieve optimal performance, as they lack the locality and translation invariance biases of CNNs. Additionally, the computational complexity of self-attention mechanisms can be a limiting factor, particularly for high-resolution images [DBK+20].

### 2.5.3 Hybrid Models and Variants

To address the data efficiency and computational challenges of ViTs, several hybrid models and variants have been proposed:

- **Hybrid CNN-Transformer Models**: Combining the local feature extraction capabilities of CNNs with the global context modeling of transformers. These models aim to utilize the strengths of both architectures [CLY+21].
- **Swin Transformer**: This hierarchical vision transformer introduces shifted windows for computing self-attention, reducing computational complexity while maintaining performance [LLC+21].
- **Data-efficient Image Transformers (DeiT)**: DeiT models are trained with knowledge distillation techniques to improve data efficiency, allowing ViTs to perform well even with limited training data [TCD+21].

# Chapter 3

# State of the Art and Related Work

## 3.1 Learned Depth Estimation

**Depth estimation** has evolved significantly with the advent of deep learning. Traditional methods relied on stereo vision and SfM, which required multiple images and known camera poses. Deep learning approaches, particularly those based on CNNs, have enabled monocular depth estimation by learning depth cues from large datasets. These methods can predict dense depth maps from a single Red-Green-Blue (RGB) image, making them suitable for applications where stereo data is not obtainable.

Recent achievements have introduced self-supervised and semi-supervised learning techniques, reducing the reliance on ground-truth depth data. These new methods leverage photometric consistency and geometric constraints between consecutive frames to learn depth estimation. Integrating attention mechanisms and transformer architectures, such as ViTs, have further improved the accuracy and generalization of these models.

### 3.1.1 Depth-Anything-V2

**Depth-Anything-V2** is a Transformer-based model for monocular depth estimation, designed to predict dense depth maps from single RGB images [YKH+24]. It uses large-scale pretraining and fine-tuning strategies to achieve high accuracy across diverse datasets.

The architecture of **Depth-Anything-V2** comprises the following components:

- **Encoder**: Utilizes variants of the **DINOv2** ViT as the backbone encoder [ODM+23]. The model supports multiple scales:
  - **ViT-Small (ViT-S)**: 24.8 million parameters, suitable for resource-constrained environments.
  - **ViT-Base (ViT-B)**: 97.5 million parameters, offering a balance between performance and efficiency.
  - **ViT-Large (ViT-L)**: 335.3 million parameters, providing enhanced accuracy for more demanding applications.
  - **ViT-Giant (ViT-G)**: 1.3 billion parameters, designed for scenarios requiring the highest level of detail and precision.
- **Decoder**: Employs a Dense Prediction Transformer (DPT) to process the extracted features and generate high-resolution depth maps.
- **Training Strategy**: Implements a teacher-student framework. The teacher model, based on the ViT-G encoder, is trained on 595K synthetic labeled images. It then generates pseudo-labels for over 62 million real-world unlabeled images. Student models with smaller encoders (ViT-S, ViT-B, ViT-L) are subsequently trained on this pseudo-labeled data, enhancing generalization to real-world scenarios [YKH+24].
- **Loss Functions**: Combines scale and shift-invariant loss with gradient matching loss to ensure depth consistency and sharpness in predictions.

**Depth-Anything-V2** demonstrates state-of-the-art performance in monocular depth estimation, offering a balance between accuracy and computational efficiency.

| Model Variant | Parameters | Use Case |
|---|---|---|
| ViT-Small (ViT-S) | 24.8M | Edge devices, real-time applications |
| ViT-Base (ViT-B) | 97.5M | Balanced performance and efficiency |
| ViT-Large (ViT-L) | 335.3M | High-accuracy requirements |
| ViT-Giant (ViT-G) | 1.3B | Maximum detail and precision |

Table 3.1: Depth-Anything-V2 Model Variants

## 3.2 Feature Matching

**Feature matching** is a fundamental task in computer vision, used to identify corresponding points across multiple images. These correspondences are essential for downstream tasks such as 3D reconstruction, camera pose estimation, and stereo depth computation.

While traditional methods that relied on handcrafted feature descriptors like SIFT, SURF, and ORB are effective in controlled environments, they often fail under variations in illumination, scale, or viewpoint.

**Dense Feature Matching** has emerged as a more robust alternative, attempting to establish pixel-wise correspondences between entire image pairs. Instead of detecting a few salient keypoints, dense matchers produce per-pixel correspondences or probability distributions over possible matches, often guided by learned confidence maps.

### 3.2.1 RoMa and Tiny RoMa

**RoMa** is a Transformer-based model designed for dense feature matching [ESB+24]. It features a hierarchical architecture that includes both self-attention and cross-attention mechanisms to robustly establish correspondences across challenging image pairs. RoMa has been shown to perform reliably under variations in viewpoint, lighting, and occlusion.
Its architecture includes:

- **Feature Extraction**: A frozen **DINOv2** backbone extracts coarse global features from the input images [ODM+23].
- **Fine Feature Enhancement**: Complements the coarse global features with fine-grained features from a dedicated CNN block to boost spatial accuracy.
- **Transformer Matching Decoder**: Predicts a probability distribution over a grid of anchor points using cross-attention and classification-based regression.
- **Loss Function**: Uses a regression-by-classification strategy to improve stability and better handle multi-modal distributions.

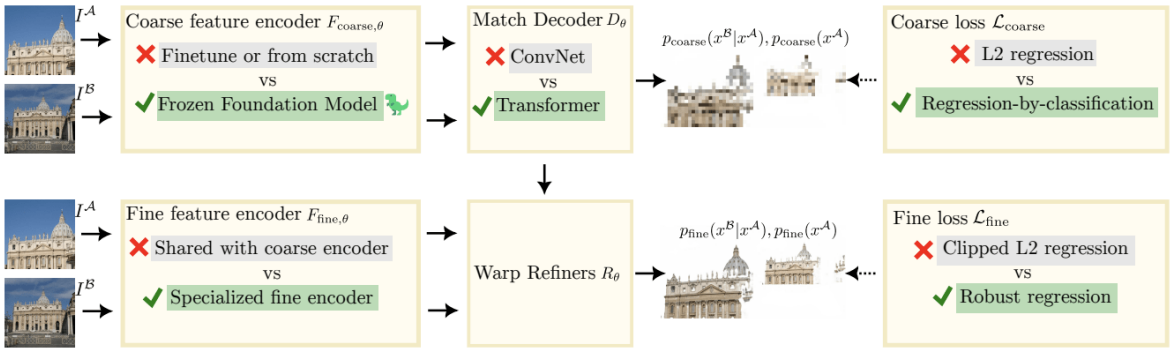


Figure 3.1: Architecture of RoMa [ESB+24].

**Tiny RoMa** is a streamlined version of RoMa that emphasizes efficiency and deployability on edge devices. It retains the core logic of RoMa but replaces the heavy backbone with lightweight CNN blocks and reduces the transformer depth [ESB+24]. Key components include:

- **Compact Encoder**: Based on **XFeat** with modifications to support transformer-style processing [PCA+24].
- **Matching Decoder**: Maintains a Transformer-based attention decoder for anchor-based classification, enabling accurate yet fast correspondence estimation.
- **Design Trade-offs**: Reduces the number of attention layers and parameters without significantly compromising matching performance.

Its suitability for real-time applications like simultaneous localization and mapping (SLAM), visual-inertial odometry, and mobile AR makes Tiny RoMa a compelling solution for resource-limited deployments.

| Benchmark | Model | AUC@5° | AUC@10° | AUC@20° / mAA@10 |
|---|---|---|---|---|
| MegaDepth-1500 | XFeat | 46.4 | 58.9 | 69.2 |
| | XFeat* | 51.9 | 67.2 | 78.9 |
| | Tiny RoMa v1 | 56.4 | 69.5 | 79.5 |
| Mega-8-Scenes | XFeat | - | - | - |
| | XFeat* | 50.1 | 64.4 | 75.2 |
| | Tiny RoMa v1 | 57.7 | 70.5 | 79.6 |
| IMC22 | XFeat | - | - | 42.1 |
| | XFeat* | - | - | - |
| | Tiny RoMa v1 | - | - | 42.2 |

Table 3.2: Performance comparison of lightweight matchers on **MegaDepth-1500**, **Mega-8-Scenes**, and **IMC22** benchmarks.

## 3.3 Synthetic Data in Computer Vision

Acquiring large-scale, annotated datasets for computer vision tasks, like depth estimation, is both challenging and resource-intensive. Synthetic data has emerged as a viable alternative, offering diverse and accurately labeled datasets generated from virtual environments. These datasets can simulate various conditions, including different lighting, weather, and object arrangements.

Models trained on synthetic data have demonstrated promising generalization to real-world scenarios, especially when combined with domain adaptation techniques such as adversarial training, style transfer, or self-supervised fine-tuning on target domains. The use of synthetic data also facilitates the exploration of rare or hazardous scenarios, which are difficult to capture in real life.

### 3.3.1 NeRF-Supervised Deep Stereo

**NeRF-Supervised Deep Stereo** is a novel framework introduced by Tosi et al. [TTDGP23b] that leverages NeRF to train deep stereo networks without the need for ground-truth depth or stereo camera setups. This approach enables the generation of high-quality stereo training data from monocular image sequences captured with a single high-res, handheld camera.

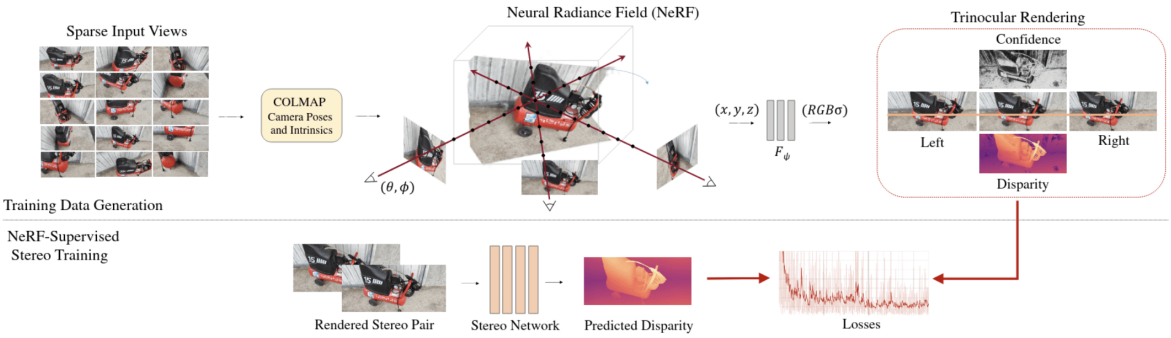


Figure 3.2: Overview of the NeRF-Supervised Deep Stereo pipeline [TTDGP23b].

The pipeline comprises the following stages:

- **Data Acquisition and Preprocessing**: Monocular image sequences are collected using a handheld camera. Camera intrinsics and extrinsics are estimated using COLMAP to prepare the data for NeRF training.
- **NeRF Training**: An independent NeRF model is trained for each scene using Instant-NGP, optimizing an L2 loss between rendered and actual pixel colors.
- **Stereo Pair Rendering**: Virtual stereo cameras are defined within the trained NeRF model to render binocular stereo pairs and corresponding disparity maps at arbitrary resolutions. Additionally, a third image is rendered to the left of the reference frame to produce rectified stereo triplets.
- **Stereo Network Training**: The rendered image triplets and disparity maps are used as training data for stereo networks. A NeRF-Supervised (NS) loss is employed, combining a triplet photometric loss—measuring the photometric difference between warped reference images using Structural Similarity Index Measure (SSIM) and absolute pixel difference, and a disparity regression loss using the rendered depth maps as proxy labels.

This methodology results in stereo networks capable of predicting sharp and detailed disparity maps. Experimental evaluations demonstrate that models trained under this regime yield a 30–40% improvement over existing self-supervised methods on challenging datasets like **Middlebury**, bridging the gap to supervised models and, in many cases, outperforming them in zero-shot generalization scenarios [TTDGP23b].

# Chapter 4

# NeRF-Based Data Generation

## 4.1 Overview

In this chapter, I detail the methodology employed to generate the NeRF generated synthetic datasets used for testing the viability of training Transformer-based architectures for tasks such as stereo matching and depth estimation. The approach integrates data from two primary sources: **NeRF-Stereo Dataset** and the **7-Scenes Dataset**.

**NeRF-Stereo Dataset**: Proposed by Tosi et al. [TTDGP23a], this dataset includes both NeRF-rendered image triplets and the original real images used for generating these triplets. In this thesis the original images (raw data) were used. The dataset contained a large number of scenes, each scene containing 100 high-resolution images, along with camera intrinsics and extrinsics computed using **COLMAP**.

Figure 4.1: Sample of images from scene 0097 of the NeRF-Stereo Dataset.

**7-Scenes Dataset**: Collected by Microsoft Research [SGZ+13], this indoor dataset features RGB-D sequences captured with a Kinect sensor at a resolution of 640×480. Each scene includes 500 to 1000 frames with ground-truth camera poses obtained via KinectFusion. The dataset comprises seven scenes: *Chess*, *Fire*, *Heads*, *Office*, *Pumpkin*, *Red Kitchen*, and *Stairs*.

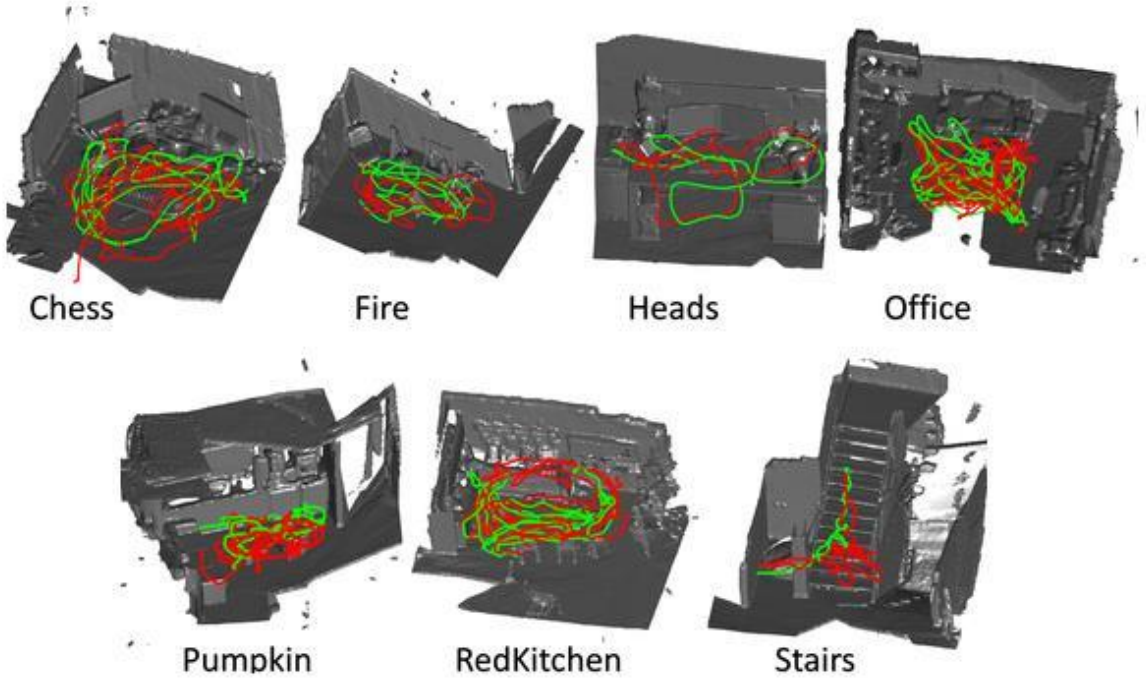


Figure 4.2: Overview of 7 Scenes dataset [SGZ+13].

Using these two datasets, I constructed a new hybrid dataset with the following components:

- **NeRF-Stereo Scenes**: I selected two scenes (*0080* and *0097*) from the NeRF-Stereo dataset and trained NeRF models on them to produce complete synthetic RGB-D image sets.
- **7-Scenes Augmented Scenes**: I partially augmented five scenes from the 7-Scenes dataset — *Stairs*, *Heads*, *Pumpkin*, *Red Kitchen*, and *Fire* — by replacing a portion of their frames with NeRF-generated versions. The proportion of replaced frames varied between 0.83% and 10%, depending on the scene.

To minimize artifacts introduced during rendering, I applied a post-processing pipeline that included color jittering, Gaussian blurring, and various depth map denoising steps.

### 4.1.1 Torch-NGP and nerf-template

For theNeRF training, I initially used the **Torch-NGP** implementation of Instant-NGP, developed by ashawkey [Tan22b]. This PyTorch-based framework supports multi-resolution hash encodings and GPU-accelerated training.

Later, I switched to **nerf-template** [Tan22a], a more stable and up-to-date fork by the same author. This version offered improved compatibility with recent PyTorch versions.

#### Evaluation Metrics: PSNR and SSIM

In order to assess the performance of the trained NeRF models, I employed the two widely adopted following image quality metrics: Peak Signal-to-Noise Ratio (PSNR) and SSIM.

- **Peak Signal-to-Noise Ratio (PSNR)**: PSNR measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. In the context of image reconstruction, it evaluates the pixel-wise differences between the synthesized image and the ground truth. A higher PSNR value indicates that the reconstructed image is closer to the original, signifying better reconstruction quality [WBSS04].
- **Structural Similarity Index Measure (SSIM)**: SSIM assesses the similarity between two images by comparing their luminance, contrast, and structural information. Unlike PSNR, which focuses solely on pixel-wise errors, SSIM considers perceptual aspects of image quality, aligning more closely with human visual perception. A SSIM value closer to 1 indicates higher structural similarity between the reconstructed and reference images [Ori11].

These metrics provide complementary insights into the fidelity of the NeRF-generated images, with PSNR focusing on absolute errors and SSIM capturing perceptual quality aspects. Using both allows for a comprehensive evaluation of the models' performance in synthesizing photorealistic views.

## 4.2 7-Scenes: Partial Synthetic Augmentation

To augment the **7-Scenes dataset**, I selected five out of the seven indoor scenes, each containing multiple video sequences. I focused on one or two sequences per scene.

Given the low resolution of the dataset and the need for consistent camera motion, I was forced to restrict the NeRF training to small 50-frame subets. Prior to making this decision I tried to train NeRF models on bigger batches (100-500) frames and found that the inconsistency of camera motion paired with the low resolution prevented the model from learning at all. Each subset was subsequently split into a 9 to 1 train-validation split (45 train and 10 validation) and each NeRF model was trained for approximately 500 epochs (20000 steps) using the default configuration settings provided by the **nerf-template** framework. I decided to use the defaults after preliminary experiments showed that modifications, such as increasing the number of rays, led to poorer rendering quality likely due to the low resolution nature of the images. After training, I rendered all 50 frames using the resulting best NeRF model and replaced the original frame subset from the dataset with the NeRF generated one.

| Scene | Total Training Frames | NeRF-Generated Frames (%) |
|---|---|---|
| Stairs | 2000 | 2.5% |
| Heads | 1000 | 10.0% |
| Pumpkin | 6000 | 0.83% |
| Red Kitchen | 7000 | 1.42% |
| Fire | 2000 | 7.5% |

Table 4.1: Percentage of NeRF-generated frames used in each scene's training set.


Figure 4.3: Original frame vs NeRF-genereated counterpart.

To improve the quality of the NeRF outputs and try to eliminate any NeRF artifacts, I applied the following post-processing pipeline:

- **RGB Images**:
  - Color jittering to simulate real-world lighting variation.
  - Gaussian blur to reduce rendering artifacts.
- **Depth Maps**:
  1. Channel flattening and masking to convert multi-channel inputs into valid float32 depth maps.
  2. Median blur (5×5) to remove salt-and-pepper noise.
  3. Morphological closing with a 7×7 elliptical kernel to fill holes and gaps.
  4. Clamping extremely small depth values to avoid numerical instability.
  5. Normalization to the [0, 1] range for uniform scale across scenes.

Figure 4.4: Post-processing RGB-D pair.

### 4.2.1 Fire Scene

For the experiments presented later in this thesis I used specifically the **Fire** scene. The orginal training split included two sequences of 1000 frames each, but only *Sequence 1* contained NeRF-generated frames. To increase the influence of synthetic data in training I excluded *Sequence 2*. This modification resulted in a training set comprising 1000 frames, of which 15% were generated using NeRF.

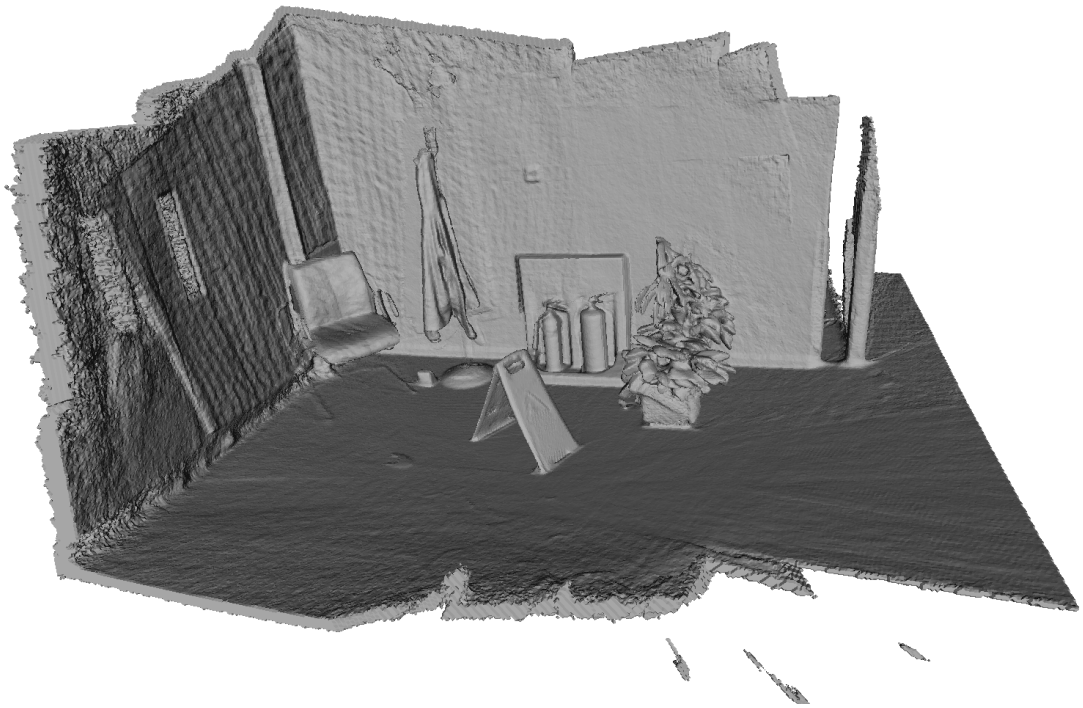

Figure 4.5: Overview of the Fire scene used throughout the experiments [SGZ+13].

To train the NeRF models that produced these synthetic frames, three separate subsets of the scene were selected: **Subset 1:** Frames 0–49, **Subset 2:** Frames 950–999, **Subset 3:** Frames 750–799.

These subsets were chosen to represent diverse regions of the scene, both temporally and spatially, and because they contained minimum camera motion that allowed the NeRF models to achieve actual results. **Subset 1** captures the early part of the sequence, **Subset 2** focuses on the tail end, and **Subset 3** targets a mid-range portion with consistent structure.

Each of these subsets was used to train a separate NeRF model. The training losses over epochs for all three models are illustrated in Figure 4.6. All models demonstrate consistent convergence, with **Subset 2** exhibiting the fastest reduction in loss and achieving the lowest final loss value. This suggests that frames 950–999 offered better coverage or more favorable geometry for radiance field learning.

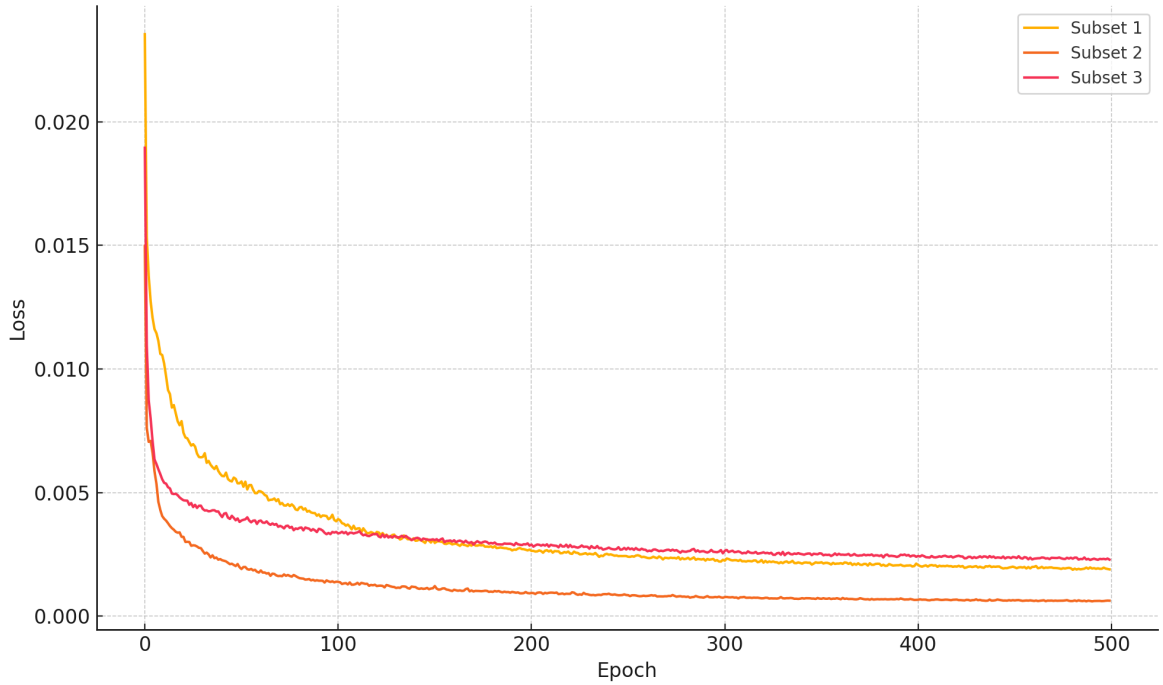


Figure 4.6: Loss over epochs for the three NeRF models trained on the Fire scene.

| Metric | Subset 1 (0–49) | Subset 2 (950–999) | Subset 3 (750–799) |
|---|---|---|---|
| PSNR ↑ | 0.5755 | 0.7940 | 0.4748 |
| SSIM ↑ | 0.5755 | 0.7940 | 0.4748 |

Table 4.2: Final evaluation metrics (PSNR and SSIM) for the three NeRF models trained on different frame subsets of the Fire scene.

Evaluation metrics for the trained NeRF models are summarized in Table 4.2.1. **Subset 2** outperforms the other two across both PSNR and SSIM, indicating better photometric accuracy and perceptual quality of the synthesized views. **Subset 1** achieves moderate performance, while **Subset 3** shows comparatively lower values, potentially due to suboptimal frame selection or scene complexity within that range.

## 4.3 NeRF-Stereo: Fully Synthetic Scenes

To complement the partially synthetic data from 7-Scenes, I generated fully synthetic datasets using scenes **0080** and **0097** from NeRF-Stereo. For each scene, I used 90 frames for training and 10 for validation. The NeRF models were trained for 300 epochs (27000 steps) using the **nerf-template** framework. Post-training, all 100 frames were rendered utilizing the best model to produce high-fidelity RGB-D pairs.

Figure 4.7: Original frame from scene 0097.

The same post-processing pipeline described earlier was applied to clean the NeRF outputs, ensuring consistency across datasets. This included color jittering, Gaussian blur for RGB images, and median blur, morphological closing, clamping, and normalization for depth maps.

| **Epoch** | **PSNR ↑** | **SSIM ↑** |
|---|---|---|
| 10 | 0.7714 | - |
| 50 | 0.7874 | - |
| 100 | 0.7926 | - |
| 200 | 0.7964 | - |
| 300 | 0.7911 | 0.7911 |
| 300* | 0.8246 | 0.8246 |

Table 4.3: PSNR and SSIM values for the final evaluations.

Figure 4.8: NeRF-generated and augmented RGB-D pair from scene 0097.

The NeRF models were trained using the **nerf-template** framework with its default configuration settings. While I briefly explored alternative configurations, such as increasing the number of rays per rendering pass to improve detail, these changes consistently led to degraded image quality, likely due to overfitting or reduced generalization. As a result, I continued with the default parameters provided by the framework, as they offered the best balance between training stability and output fidelity across the evaluated scenes.

Table 4.3 summarizes the PSNR and SSIM values at the final evaluation epoch. The entry marked with an asterisk (*) corresponds to the evaluation conducted using the maximum number of rays, resulting in higher fidelity metrics due to the increased sampling density during rendering.

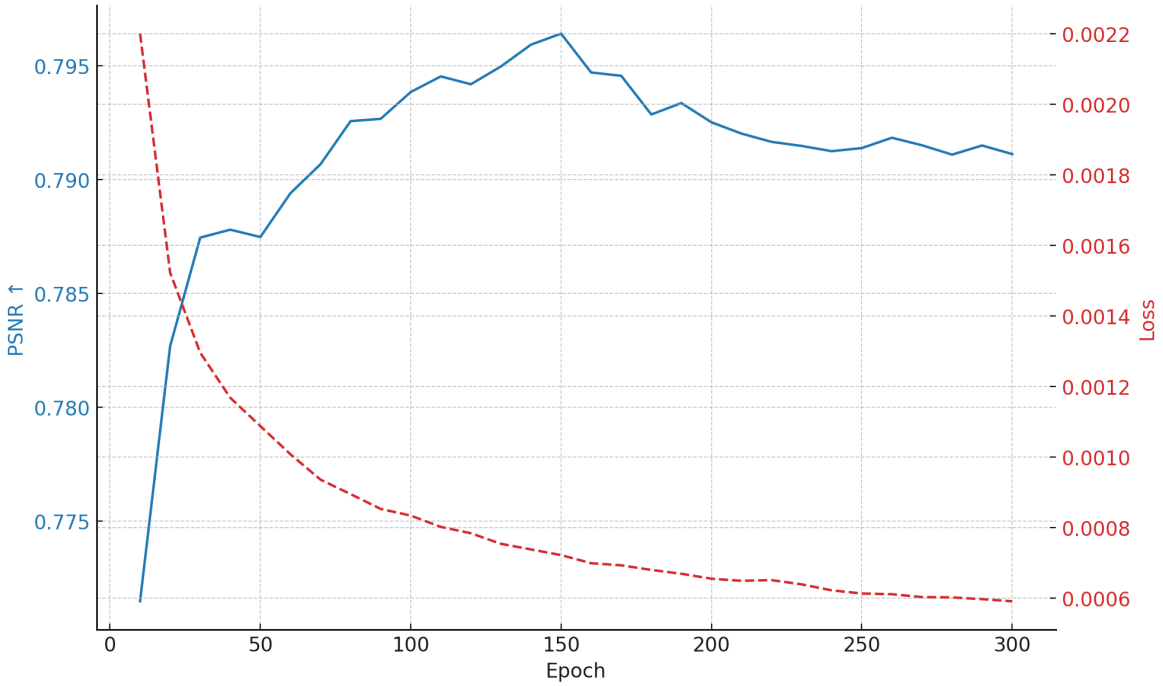


Figure 4.9: Loss and PSNR over epochs for scene 0097.

# Chapter 5

# Experiments with Tiny RoMa

## 5.1 Model Overview & Evaluation Metrics

### 5.1.1 Model Overview

This chapter focuses on the **Tiny RoMa** model, a streamlined variant of the RoMa architecture, which I used to explore dense feature matching in my experiments. While a more detailed discussion of RoMa and Tiny RoMa is provided in Section 3.2.1, here I briefly summarize the key aspects relevant to my work.

**RoMa** is a Transformer-based model that uses self-attention and cross-attention mechanisms to establish robust correspondences across challenging image pairs, handling variations in viewpoint, lighting, and occlusion [ESB+24].

**Tiny RoMa** retains the architectural principles of RoMa but focuses on efficiency and suitability for deployment on edge devices. It replaces the heavy backbone with lightweight CNN blocks and reduces the transformer depth, making it practical for real-time applications like SLAM or mobile AR.

During training, the model does not process single images but instead operates on pairs of images that include geometric context. Each training pair consists of:

- Two RGB images ($\mathbf{I}_A$, $\mathbf{I}_B$),
- Their corresponding depth maps ($\mathbf{D}_A$, $\mathbf{D}_B$),
- The relative pose between the views, computed as $\mathbf{T}_{A\to B} = \mathbf{T}_B^{-1} \cdot \mathbf{T}_A$.

To construct such input, I used a balanced pair sampling approach. For each image in the dataset, I randomly selected a fixed number of unique image pairs with the following constraints:

- No image was paired more than a predefined number of times (e.g., 40 during training),
- Each pair had valid depth and pose data,
- Pairs were symmetrically and uniformly distributed across the dataset.

### 5.1.2 Evaluation Metrics

To evaluate the performance of Tiny RoMa in my experiments, I used several common metrics in dense feature matching and correspondence estimation:

- **Percentage of Correct Keypoints (PCK)**: This metric measures how many predicted keypoints fall within a given distance threshold of the ground truth. I computed it at multiple pixel thresholds (1px, 3px, and 5px) to assess precision across scales [ESB+24]. Higher is better.
- **Area Under the Curve (AUC)**: AUC corresponds to the area under the PCK-vs-threshold curve and provides a more comprehensive summary of model performance than individual PCK scores [ESB+24]. Higher is better.
- **Mean Average Accuracy (mAA)**: This metric averages the matching accuracy over multiple thresholds, similar to how Mean Average Precision (mAP) is used in object detection, giving a single score for overall matching quality [ESB+24]. Higher is better.

These metrics allowed me to compare different training strategies involving real and synthetic data.

## 5.2 Fine-Tuning on the Fire Scene

### 5.2.1 Dataset Overview

I selected the **Fire** scene from the 7-Scenes dataset primarily because, after excluding **Sequence 2**, it offered the highest proportion of NeRF-generated frames. This made it an ideal candidate for studying the effects of partial synthetic augmentation on feature matching.

As outlined in Section 4.2.1, I constructed two training variants of the dataset:

- A **real dataset**, consisting entirely of original frames from the 7-Scenes Fire sequence.
- A **NeRF-augmented dataset**, in which approximately 15% of the frames were replaced with synthetic frames rendered by NeRF. These were generated from three 50-frame subsets of **Sequence 1** using separately trained NeRF models, followed by post-processing to refine RGB-D quality.

To generate the training data I created pairs such that each training image appears in no more than 40 unique pairs, and each validation image appears in no more than 10 pairs. This constraint ensured a balanced and diverse distribution of pairs while avoiding overrepresentation of individual frames. I used the exact same set of image pair IDs for both dataset variants (i.e, in the NeRF-augmented version, any frame that had a NeRF-rendered counterpart simply replaced the corresponding real frame in the pair). As a result, the pair structure remained identical across both datasets. This yielded a total of 19800 training pairs and 9608 validation pairs for each configuration.

### 5.2.2 Fine-Tuning Configuration

To ensure a consistent evaluation between the real and NeRF-augmented datasets, identical training setup were used for both experiments. The fine-tune process was conducted using the Tiny RoMa model with the pretrained weights on an outdoor set of data. The training spanned 100 epochs.

The key hyperparameters and training settings were as follows:

- **Image resolution:** $560 \times 560$ pixels
- **Batch Size:** 40 for both training and validation phases
- **Learning Rate:** Initialized at $5 \times 10^{-6}$, adjusted using a cosine annealing schedule with $T_{\max} = 100$ and a minimum learning rate of $1 \times 10^{-6}$

- **Weight Decay:** $1 \times 10^{-4}$
- **Backbone:** All layers were unfrozen, allowing full fine-tuning of the model

I chose this configuration to balance computational efficiency with model performance, adhering to best practices in training Transformer-based architectures for dense feature matching tasks.

### 5.2.3 Results on the Fire Scene

#### Training & Evaluation Loss

The training loss curves for both the real and NeRF-augmented versions of the Fire scene followed a similar pattern. The model trained on the real dataset exhibited a steady decrease in loss over the initial epochs, stabilizing around **0.38** by **epoch 5**. The NeRF-augmented version started with a slightly higher loss but quickly converged to a comparable value by the same point in training.

This convergence behavior suggests that the introduction of NeRF-generated frames did not negatively impact optimization or training stability. In both cases, the loss curves indicate that the model was able to learn consistently without overfitting.

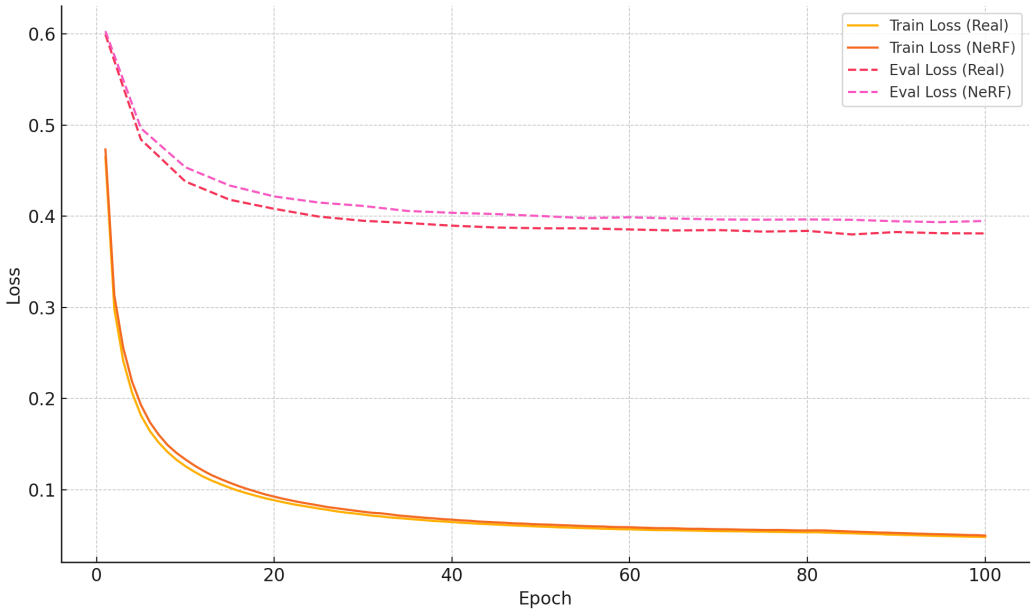


Figure 5.1: Comparison of training and evaluation loss over epochs for the real and NeRF-augmented Fire datasets.

### Evaluation Metrics

At **epoch 100**, both models achieved similar results in terms of dense correspondence accuracy. The NeRF-augmented configuration showed a marginal improvement in PCK@5px and mAA, suggesting a slight benefit for coarse keypoint matching. However, performance at finer thresholds (PCK@1px, PCK@3px) remained effectively unchanged between the two versions.

Interestingly, despite expectations for improved performance over time, the PCK metrics exhibited a decline in later epochs. This counterintuitive trend is likely attributable to overfitting.

These results imply that while synthetic data can be integrated without degrading model performance and may even slightly enhance generalization in lower-precision regimes, caution must be exercised to prevent overfitting.

| Metric | Real | NeRF-Augmented |
|---|---|---|
| PCK@1px ↑ | 0.00010 | 0.00010 |
| PCK@3px ↑ | 0.00079 | 0.00079 |
| PCK@5px ↑ | 0.00195 | 0.00203 |
| AUC ↑ | 0.00020 | 0.00021 |
| mAA ↑ | 0.00518 | 0.00535 |

Table 5.1: Performance Metrics at Epoch 100: Real vs. NeRF-Augmented Fire Scene

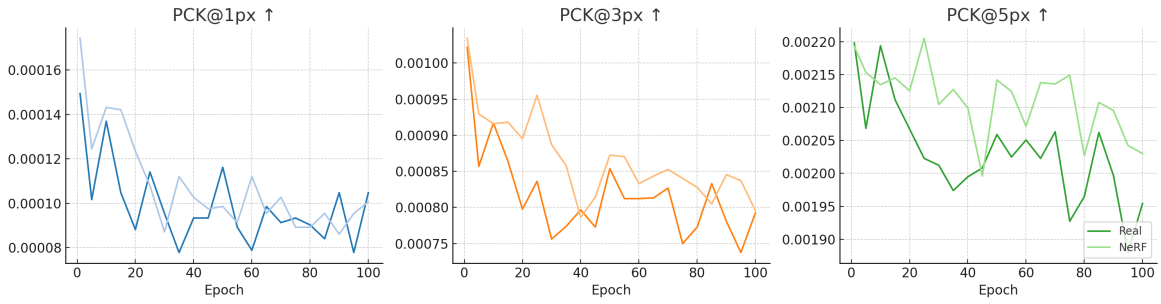


Figure 5.2: Comparison of PCK@1px, PCK@3px, and PCK@5px over epochs for real and NeRF-augmented Fire datasets.

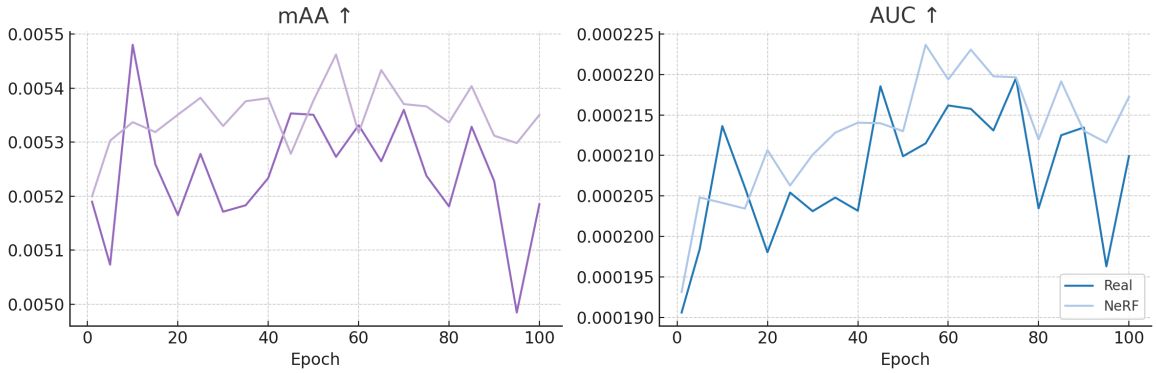


Figure 5.3: Comparison of mAA and AUC over epochs for real and NeRF-augmented Fire datasets.

## 5.3 Fine-Tuning on the 0080 Scene

### 5.3.1 Dataset Overview

To evaluate the effectiveness of fully synthetic training data, I used scene **0080** from the NeRF-Stereo dataset. As detailed earlier in Section 4.3, this dataset was generated entirely through NeRF rendering.

To generate training and validation pairs, I applied the same uniform sampling algorithm described in Section 5.2.1, but with a cap of 25 appearances per frame in both the training and validation splits. This resulted in 1115 training pairs and 90 validation pairs for the **0080** scene.

### 5.3.2 Fine-Tuning Configuration

The training setup mirrored the configuration used for the **Fire** scene (see Section 5.2.2), with the exception of the learning rate schedule. The Tiny RoMa model was fine-tuned using the pretrained weights (on an outdoor set of data) for a total of 100 epochs.

Due to instability encountered in early training (including loss values diverging to NaN), I introduced a warm-up phase for the first 20 epochs. During this phase, I used a lower learning rate and disabled weight decay to stabilize convergence.

The specific configuration was:

- **Epochs 1–20 (warm-up):**
  - Learning rate: $2 \times 10^{-6}$ with cosine annealing ($T_{\text{max}} = 20$, $\eta_{\text{min}} = 1 \times 10^{-6}$)
  - Weight decay: $0$
- **Epochs 21–100 (main training):**
  - Learning rate: $5 \times 10^{-6}$ with cosine annealing ($T_{\text{max}} = 80$, $\eta_{\text{min}} = 1 \times 10^{-6}$)
  - Weight decay: $1 \times 10^{-4}$

This two-stage configuration was designed to avoid early instability, while still allowing the model to fully fine-tune with more aggressive regularization once stable convergence had been achieved.

While after 100 epochs it could be seen that no plateau was reached and that the loss continued to decrease, I decided to stop the fine-tuning due to noticing the poor evaluation metrics that will be discussed shortly.

### 5.3.3 Results on the 0080 Scene

#### Training & Evaluation Loss

The training loss for the **0080** scene decreased rapidly during the initial epochs, reaching extremely low values (on the order of $10^{-4}$ to $10^{-5}$) by **epoch 10**. At **epoch 21**, a noticeable spike in loss occurred, which corresponds to the point at which the learning rate and weight decay were reset for the second phase of training. This brief instability was expected due to the more aggressive learning settings.

After this transition, the loss quickly reconverged to a low regime. However, rather than stabilizing fully, the loss continued to exhibit some jitter in subsequent epochs. These fluctuations, while relatively minor with some occasional spikes, suggest sensitivity to learning dynamics likely amplified by the limited size and homogeneity of the synthetic dataset.

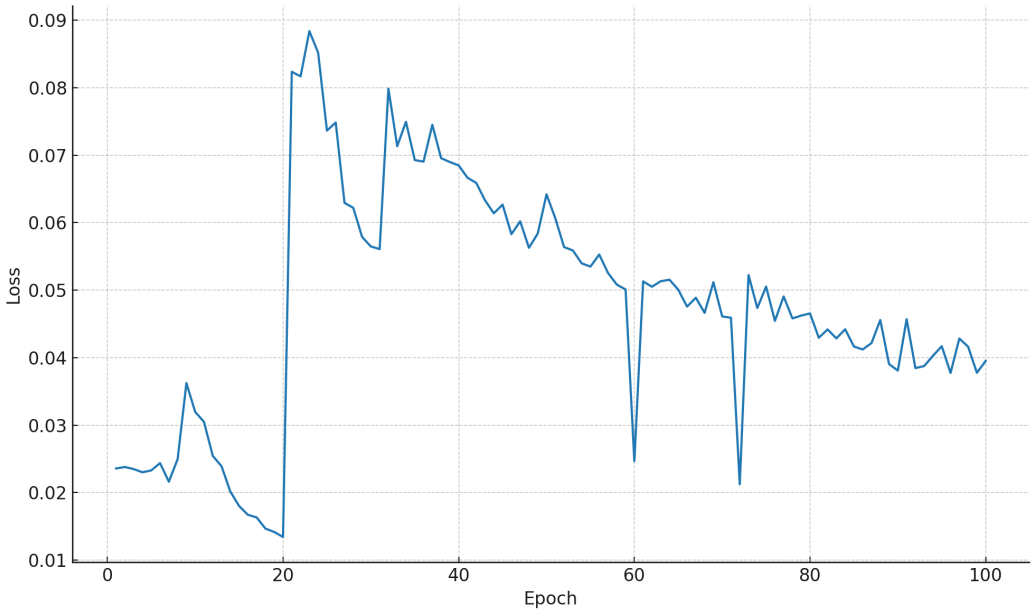


Figure 5.4: Training loss over epochs for Tiny RoMa on the fully synthetic 0080 scene.

### Evaluation Metrics

Despite achieving a low training loss, all evaluation metrics remained at zero throughout training. This suggests a complete failure of the model to establish meaningful correspondences on the validation set. There are two likely contributing factors:

1. The model overfit the synthetic training pairs and failed to generalize even to other synthetic frames, likely due to the lack of visual diversity and real-world complexity.

2. The fact that all the metric values were zero shows that the model was totally unable to detect common ground-truth point between the images of the pairs. Because no common features were found no matches could be established and the metrics could no be computed.

Together, these findings suggest that, at least in this setup, training dense correspondence models solely on NeRF-generated synthetic data may be unreliable. While synthetic data can complement real datasets, as shown in the **Fire** scene experiments, it appeared insufficient when used on its own in the context of the synthetic dataset generated for this thesis.

## 5.4 Comparative Analysis and Discussion

The experiments conducted on the **Fire** and **0080** scenes provide insights into the role of synthetic data in training Transformer-based models for dense correspondence tasks.

In the **Fire** scene, augmenting the real dataset with approximately 15% NeRF-augmented datasets. Notably, the NeRF-augmented model achieved marginal improvements in PCK@5px and mAA metrics, indicating enhanced performance in coarse keypoint matching scenarios. However, metrics at finer thresholds (PCK@1px and PCK@3px) remained virtually unchanged, suggesting that the benefits of synthetic augmentation may be more pronounced at lower precision levels.

Conversely, training the model exclusively on the fully synthetic **0080** scene resulted in poor generalization. Despite achieving low training loss values, all evaluation metrics remained at zero throughout the training process. This outcome suggests that the model overfitted to the synthetic training data and failed to generalize even to other synthetic frames. Potential contributing factors include the lack of visual diversity in the synthetic dataset and subtle rendering artifacts inherent in NeRF-generated images.

These findings imply that while partial incorporation of high-quality synthetic data can slightly enhance model generalization without compromising performance on real data, relying solely on synthetic data for training dense correspondence models is currently unreliable. Further research is necessary to explore hybrid training approaches and improve the realism and diversity of synthetic datasets.

# Chapter 6

# Experiments with Depth-Anything-V2

## 6.1 Model Overview & Evaluation Metrics

### 6.1.1 Model Overview

This chapter focuses on the **Depth-Anything-V2** model, a Transformer-based architecture specifically designed for monocular depth estimation tasks. Depth-Anything-V2 leverages a DPT combined with **DINOv2** encoders, employing self-attention mechanisms to effectively capture comprehensive contextual and structural information within images [YKH+24].

A more extensive overview of the **Depth-Anything-V2** architecture, including encoder and decoder specifics, the teacher-student training strategy, and model variant details, is provided earlier in Section 3.1.1. For clarity, the current experiments utilized two variants of this architecture:

- **ViT-B (Base)**: A balanced variant offering an optimal trade-off between computational efficiency and prediction accuracy, suitable for practical deployment.
- **ViT-L (Large)**: A high-capacity variant designed to evaluate whether increased complexity and parameter count translate into improved performance, especially when training exclusively with synthetic data.

Both models underwent fine-tuning using pretrained checkpoints provided by the authors of Depth-Anything-V2. No architectural modifications were applied to isolate the effects arising purely from different synthetic data augmentation strategies.

Unlike the Tiny RoMa model discussed previously, Depth-Anything-V2 predicts depth maps directly from individual RGB images without requiring paired inputs or explicit geometric context, thus being inherently suited for monocular vision applications.

### 6.1.2 Evaluation Metrics

To evaluate the performance of Depth-Anything-V2 models, I used the following suite of metrics commonly adopted in monocular depth estimation research:

- **Absolute Relative Error (Abs Rel)**: Captures the mean relative error between predicted and actual depth values, providing an intuitive measure of accuracy [YKH+24]. Lower is better.
- **Squared Relative Error (Sq Rel)**: Squares relative errors, thus giving higher penalty to significant depth inaccuracies[YKH+24]. Lower is better.
- **Root Mean Squared Error (RMSE)**: Aggregates depth prediction errors into a single measure, sensitive to large errors and outliers [YKH+24]. Lower is better.
- **RMSE (log)**: Calculates RMSE in logarithmic scale, emphasizing proportional accuracy and reducing the dominance of large depth deviations [YKH+24]. Lower is better.
- **Log10 Error**: Measures the average error in terms of order-of-magnitude differences between predicted and true depth [YKH+24]. Lower is better.
- **Scale-Invariant Logarithmic Error (SILog)**: Provides an evaluation independent of absolute depth scale, critical for monocular systems where absolute scale may be ambiguous [YKH+24]. Lower is better.
- **Threshold Accuracy (D1, D2, D3)**: Indicates the fraction of depth predictions falling within specific accuracy thresholds, assessing the frequency of predictions being acceptably close to ground truth values [YKH+24]. Higher is better.

Together, these metrics facilitate detailed and nuanced assessments of depth estimation accuracy across varied synthetic data augmentation scenarios, allowing clear and direct comparisons of model performance.

## 6.2 Fine-Tuning on the Fire Scene

### 6.2.1 Dataset Overview

The **Fire** scene from the 7-Scenes dataset was reused in this experiment to maintain consistency with the setup described in Section 5.2.1. As outlined there, two variants of the dataset were prepared: one using only real frames, and one in which approximately 15% of the frames were replaced with NeRF-generated synthetic renderings.

Unlike the setup required by Tiny RoMa, Depth-Anything-V2 does not require paired image inputs. Instead, each training sample consists of a single RGB image and its corresponding depth map.

### 6.2.2 Fine-Tuning Configuration

To evaluate the effect of NeRF-based synthetic augmentation on monocular depth estimation, I fine-tuned the **ViT-B** variant of Depth-Anything-V2 using the same configuration for both the real-only and NeRF-augmented datasets. The model was initialized from the pretrained weights provided by the authors and trained without any architectural modifications.
The training spanned 60 epochs and was divided into two phases:

- **Epochs 1–10 (frozen encoder):** Only the decoder and prediction layers were updated.
- **Epochs 11–60 (unfrozen):** All layers were fine-tuned jointly.

Key training hyperparameters were as follows:

- **Image resolution:** $476 \times 476$ pixels
- **Encoder Learning rate:** $5 \times 10^{-6}$
- **Decoder Learning rate:** $5 \times 10^{-5}$
- **Scheduler:** cosine annealing and 7-epoch warm-up
- **Optimizer:** AdamW
- **Weight decay:** 0.02
- **Batch size:** 16 during the frozen phase, 8 during the unfrozen phase

### 6.2.3 Results on the Fire Scene

#### Training & Evaluation Loss

The training loss curves for the real-only and NeRF-augmented configurations of the **Fire** scene are shown in Figure 6.1. Both models started with relatively high loss values and converged steadily over the course of 60 epochs. The loss spiked at epoch 10 when the backbone was unfrozen and learning rate reset but quickly started to converge again.

Both models exhibited very similar convergence and fluctuations, however the NeRF ultimately achieved a higher loss and underperformed the model trained on 100% real data.

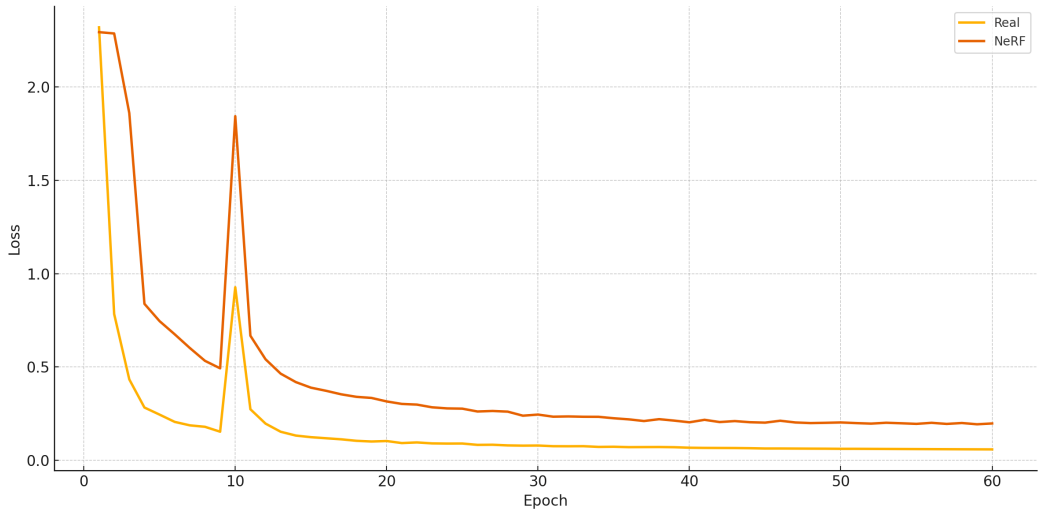


Figure 6.1: Training loss comparison between real-only and NeRF-augmented Depth-Anything-V2 (ViT-B) models on the Fire scene.

#### Evaluation Metrics

Table 6.1 reports representative evaluation metrics computed on the real validation set across the training process. For reference, the performance of the zero-shot Depth-Anything-V2 base model (without fine-tuning) is also included. As expected, both fine-tuned variants delivered dramatic improvements across all metrics, with error rates reduced by several orders of magnitude.

The model trained on real data outperformed the NeRF variant all across the board, suggesting more accurate and coherent depth predictions. Taking into consideration all metrics the NeRF variant is within 25.8% of the real data variant when averaging all percentages.

These results indicate that supplementing a real dataset with a low proportion of synthetic frames won't enhance and may damage the depth estimation performance. However, further study is needed to see if supplementing a real dataset with extra NeRF-generated frames may improve the performance.

| Metric | Base (Zero-Shot) | Real-Only | NeRF-Augmented |
|---|---|---|---|
| Abs Rel ↓ | 67.83 | 0.164 | 0.241 |
| Sq Rel ↓ | 642.20 | 0.008 | 0.017 |
| RMSE ↓ | 9.33 | 0.033 | 0.042 |
| RMSE log ↓ | 4.20 | 0.179 | 0.235 |
| Log10 ↓ | 1.82 | 0.063 | 0.087 |
| SILog ↓ | 2.98 | 0.155 | 0.199 |
| D1 ↑ | 0.0 | 0.789 | 0.666 |
| D2 ↑ | 0.0 | 0.974 | 0.934 |
| D3 ↑ | 0.0 | 0.997 | 0.986 |

Table 6.1: Representative evaluation metrics on real validation frames for the Fire scene.

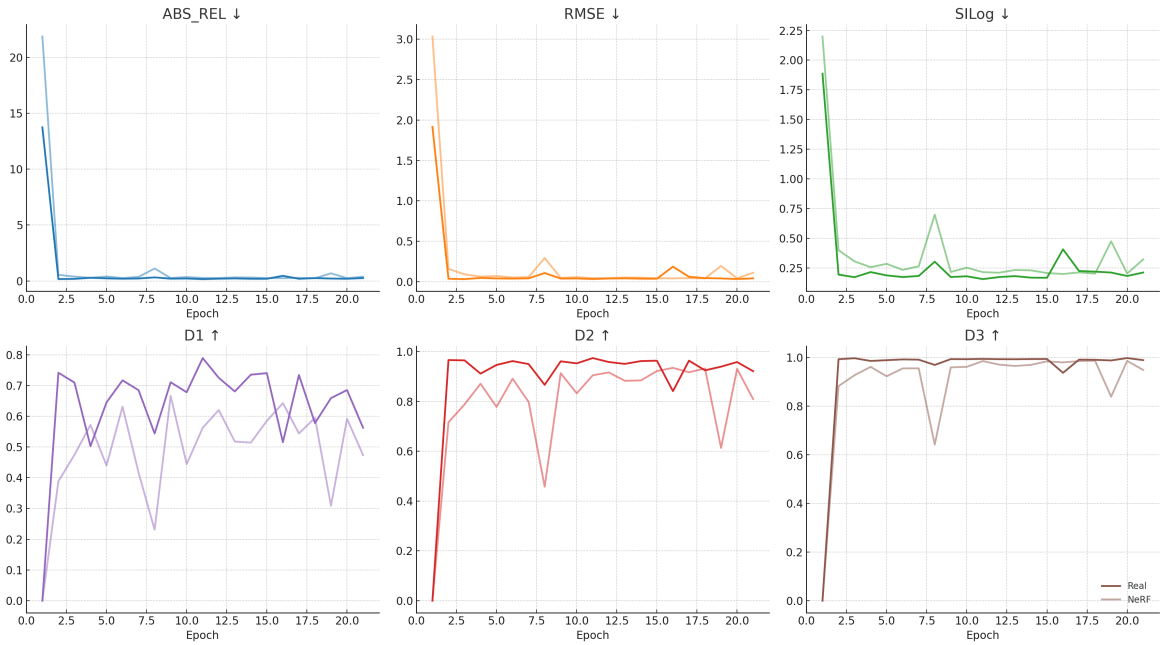


Figure 6.2: Evaluation metrics over epochs for real-only and NeRF-augmented training on Fire.

Overall, theses findings do not align with conclusions drawn in the Tiny RoMa experiment: moderate NeRF-based augmentation can be safely incorporated into fine-tuning pipelines for feature matching but seem to negatively affect monocular view depth estimation.

## 6.3 Fine-Tuning on the 0097 Scene

### 6.3.1 Dataset Overview

The dataset used for fine-tuning on the **0097** scene follows the same synthetic data generation and rendering pipeline as described in Section 5.3.1 for the **0080** scene. Like 0080, scene 0097 is part of the NeRF-Stereo dataset and was fully synthesized using NeRF, with RGB and depth maps rendered from a trained model and post-processed using the same refinement steps outlined in Section 4.3.

Unlike Tiny RoMa, which required image pairs with associated relative pose and depth, Depth-Anything-V2 operates on individual RGB-D pairs. As a result, no pair sampling algorithm was needed. I randomly selected **90** frames for training and reserved the remaining **10** for validation. Since the 0097 sequence comprises a continuous set of 100 frames, no further filtering or balancing was applied.

### 6.3.2 Fine-Tuning Configuration

Both the **ViT-B** and **ViT-L** variants of Depth-Anything-V2 were fine-tuned on the 0097 dataset using different fine-tuning settings. Each model was initialized from its respective pretrained weights (ViT-B from the DINOv2 ViT-B backbone, and ViT-L from the ViT-L variant), ensuring a fair comparison based on model capacity alone.

Fine-tuning was conducted for a total of 150 epochs. For the first 10 epochs, the encoder was kept frozen to allow the decoder and prediction heads to adapt to the synthetic domain without destabilizing pretrained features. The encoder was then unfrozen for the remaining 140 epochs, enabling end-to-end fine-tuning. The used image resolution was $518 \times 518$ pixels for both and the optimizer was AdamW. Due to the small size and visual homogeneity of the dataset, a conservative batch size of 2/4 was used to reduce overfitting risk.
ViT-B configuration:

- **Encoder Learning rate:** $2 \times 10^{-5}$
- **Decoder Learning rate:** $2 \times 10^{-4}$
- **Scheduler:** cosine annealing and 5-epoch warm-up
- **Weight decay:** 0.001 (reduced for a small dataset)
- **Batch size:** 4 (reduced to prevent overfitting)

I first tried a very similar configuration to the ViT-B when fine-tuning ViT-L, however, I encountered high training instability and the loss kept becoming NaN. Due to this, I was forced to take a more conservative approach when fine-tuning ViT-L and lowered the learning rate, increased the warm-up period and lowered the weight decay.
ViT-L configuration:

- **Encoder Learning rate:** $1 \times 10^{-6}$
- **Decoder Learning rate:** $1 \times 10^{-5}$
- **Scheduler:** cosine annealing and 20-epoch warm-up
- **Weight decay:** 0.00001 (reduced for a small dataset)
- **Batch size:** 2 (reduced even more due to memory constraints)

### 6.3.3 Results on the 0097 Scene

#### Training & Evaluation Loss

Figure 6.3 illustrates the loss progression for both ViT-L and ViT-B over 150 epochs for both training and validation. For the ViT-B variant both the training and validation losses decreased rapidly with a very small spike when the encoder was unfrozen. In contrast after the first the ViT-L model displayed complete failure to learn. The training loss fluctuated along the same range of values for the entire 150 epochs and the validation loss remained constant, with the exception of a single epoch, due to clamping being applied in validation to prevent division by zero. The model seems to never escape the range of weights that cause the division by zero, therefore never achieving a different validation loss as it always gets clamped to the minimum value.

This suggests that, while simpler ViT models like ViT-B can learn on fully NeRF-generated data, more complex models like ViT-L can not learn anything most likely due to NeRF artifacts and lack of fine-grain consistency of the NeRF-generated frames.

Compared to the **Fire** scene, even the ViT-B model displayed more perturbance and plateaued later, likely due to the lower visual diversity and absence of real-world noise in the synthetic data.

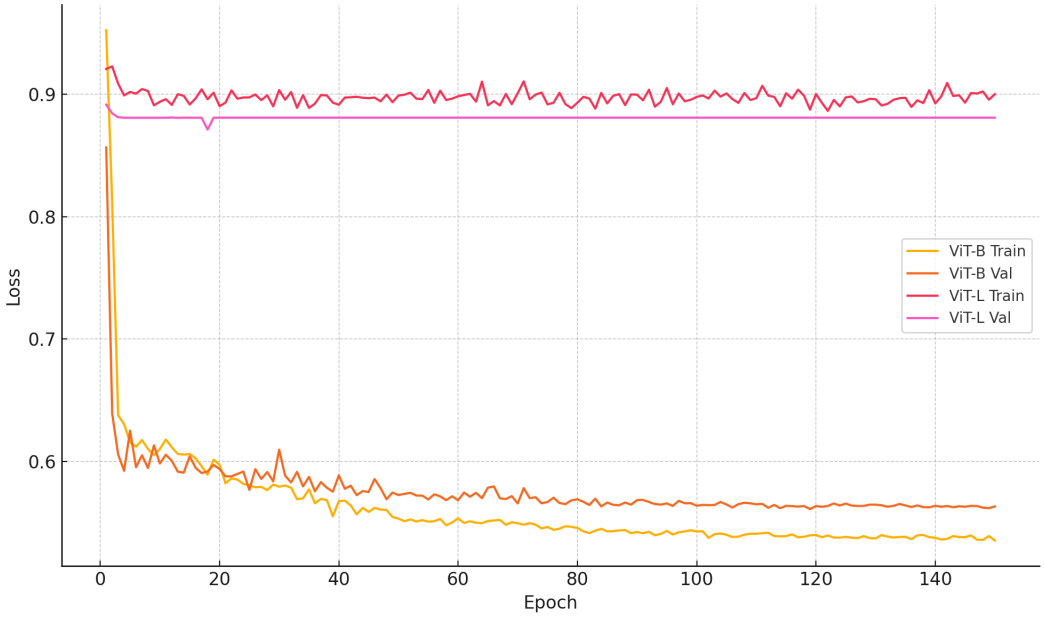


Figure 6.3: Training loss for Depth-Anything-V2 ViT-B and ViT-L on the fully synthetic 0097 scene.

### Evaluation Metrics

Table 6.2 summarizes representative evaluation metrics for both ViT-B and ViT-L variants, evaluated on the synthetic validation frames of the 0097 scene. Performance was benchmarked against the zero-shot base models to assess the impact of fine-tuning.

Starting off, a significant improvement of the Base (Zero-Shot) ViT-B model compared to the evaluation on the **Fire** scene can be seen. Although this validation set is entirely synthetic while the Fire one was 100% real the ViT-B performed much better on the NeRF likely due to the fact the Fire one was low-res and that this one is much higher resolution. In comparison the Base (Zero-Shot) ViT-L performed much worse than its smaller counterpart presenting an inherent proneness to underperform on a NeRF-generated synthetic dataset. Some metrics could not even be computed, resulting in infinity or NaN, due to division by the output of the model, which in this case was zero.

Looking at ViT-B it outperformed the base model across all metrics. Specifically, improvements in D1, D2, and D3 accuracy thresholds show that the fine-tuned model predicts within tight thresholds far more consistently. In stark contrast, the fine-tuned ViT-L model underperformed its smaller counterpart showing only small improvements when compared to the Base (Zero-Shot) ViT-L one.

| Metric | Base (Zero-Shot) ViT-B | ViT-B Fine-Tuned | Base (Zero-Shot) ViT-L | ViT-L Fine-Tuned |
|---|---|---|---|---|
| Abs Rel ↓ | 2.93 | 0.971 | 3.34 | 3.209 |
| Sq Rel ↓ | 22.8 | 3.435 | 28.02 | 26.280 |
| RMSE ↓ | 4.99 | 1.005 | 5.62 | 5.411 |
| RMSE log ↓ | 1.15 | 0.564 | inf | 1.057 |
| Log10 ↓ | 0.34 | 0.086 | inf | 0.373 |
| SILog ↓ | 1.01 | 0.560 | NaN | 0.871 |
| D1 ↑ | 0.12 | 0.827 | 0.026 | 0.047 |
| D2 ↑ | 0.26 | 0.953 | 0.140 | 0.139 |
| D3 ↑ | 0.40 | 0.973 | 0.304 | 0.305 |

Table 6.2: Evaluation metrics for Depth-Anything-V2 on the 0097 synthetic scene.

These results demonstrate that large-scale pretrained monocular depth models can generalize effectively to fully synthetic domains when fine-tuned with a tailored schedule. ViT-L's superior performance highlights the value of increased capacity, especially when training data is limited in scope and diversity.

Figure 6.4 displays the evolution of several metrics for both the ViT-B and ViT-L models when trained on the 0097 synthetic scene. Looking at ViT-B the metrics seem to plateau after some time but experience jittering during early epochs, likely due to the synthetic nature of the dataset. In contrast, the ViT-L metrics are almost entirely flat and don't seem to change. This is because clamping to minimum value was applied when computing evaluation to prevent division by zero. The model seems to never escape that range of values that cause the division by zero, with the exception of a single epoch that quickly gets reversed, and therefore the metrics stagnate at a fixed value.

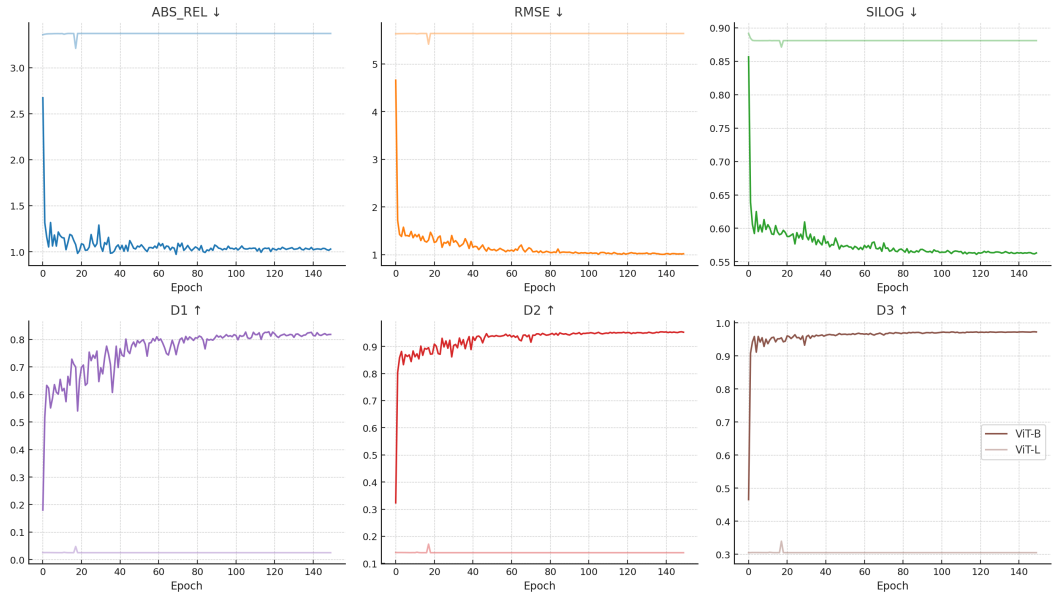


Figure 6.4: Evaluation metrics for ViT-L and ViT-B training on the 0097 scene.

## 6.4 Comparative Analysis and Discussion

The experiments conducted on the **Fire** and **0097** scenes offer important insights on the limitation and potential of using synthetic data for monocular depth estimation with Transformer-based models.

In the **Fire** scenario, where 15% of frames were replaced with NeRF-generated counterparts, the fine-tuned model showed improvements over the zero-shot baseline. However, the model trained only on real-data overperformed the NeRF one with a margin of approximately 25%. This suggests NeRF synthetic data augmentation offers limited benefit in low resolution datasets and may even introduce inconsistencies that hinder learning. Further research is needed to assess whether similar trends hold when partial NeRF augmentation is applied to high-resolution datasets.

On the other hand, the results on the fully synthetic **0097** scene highlight the impact of model size when fine-tuning on 100% synthetic data. The smaller ViT-B model, with only 97.5 million parameters, demonstrated clear improvements and generalization over the zero-shot baseline. However, the larger ViT-L model, with 335.3 million parameters, failed to converge meaningfully. Its high complexity appeared to be a poor fit for the visually inconsistent and artifact-prone nature of NeRF-generated frames. Although it did show gains over its zero-shot counterpart, theses were minimal and insufficient. These findings illustrate the limitations of purely NeRF-generated data for fine-tuning large monocular ViTs. Further experiments involving a wider range of monocular transformers sizes should help define the threshold at which model complexity begins to outweigh the benefits.

# Chapter 7

# Application

## 7.1 Overview

To provide an interactive interface for the fine-tuned **Tiny RoMa** and **Depth-Anything-V2** models I have developed an application that provides an interface that lets users easily upload images for inference on the models and tweak settings to change the way the results are displayed.

The architecture evolved from a simple **Gradio** app proof-of-concept into a full-stack application. The final design consists of a **FastAPI** backend for serving inference requests and a **React** frontend that enables user interaction and customized visual display of the results. A minimal **Gradio** interface was retained for rapid local debugging and validation of the model outputs.

Two workflows are supported within the application:

- **Stereo Feature Matching**: Accepts two RGB images and produces a match overlay using **Tiny RoMa**.
- **Monocular Depth Estimation**: Accepts a single RGB image and outputs a predicted depth map via **Depth-Anything-V2**.

Both models are served from the FastAPI backend using a custom inference interface that manages preprocessing (resizing, normalization), device allocation (e.g., GPU), and output formatting. The React frontend enables seamless interaction by issuing requests to FastAPI endpoints and rendering the visual outputs.

### 7.1.1 Requirements

**Functional Requirements**

- **Model Inference APIs:** Expose endpoints to infer depth maps (monocular input) and keypoint matches (stereo input).
- **Frontend Interaction:** Allow users to upload one or two images and receive corresponding visualizations.
- **Modular Model Support:** Enable loading and switching between multiple fine-tuned model variants (e.g., ViT-B, ViT-L).
- **Visualization:** Display depth maps and keypoint overlays in a user-friendly format and allow users to tweak the way data is displayed (e.g., colormap for depth maps, number of matches displayed for stereo matching)

**Non-Functional Requirements**

- **Startup Efficiency:** The backend must initialize large models (e.g., Depth-Anything-V2 ViT-L) with minimal delay and manage GPU memory effectively to avoid crashes.
- **Model Isolation:** Different models (Tiny RoMa and Depth-Anything-V2) must run in isolation to avoid conflicts in preprocessing, device usage, or resource loading.
- **Debuggability:** The system should support quick debugging of models and outputs through alternative lightweight interfaces like Gradio without requiring the full stack to be launched.
- **Visual Feedback Quality:** The results (depth maps, match overlays) must be rendered in a way that clearly communicates model performance, even when artifacts or low-confidence outputs are present.

## 7.2 FastAPI: Backend

I built the backend using **FastAPI**, serving as the central interface for running model inference and handling requests from the React frontend. It exposes Representational State Transfer (REST) endpoints for two tasks: monocular depth estimation with Depth-Anything-V2 and stereo feature matching with Tiny RoMa.

Models are loaded lazily on first use and stored in memory for reuse. A locking mechanism ensures that model initialization, change, and inference remain thread-safe during concurrent requests.

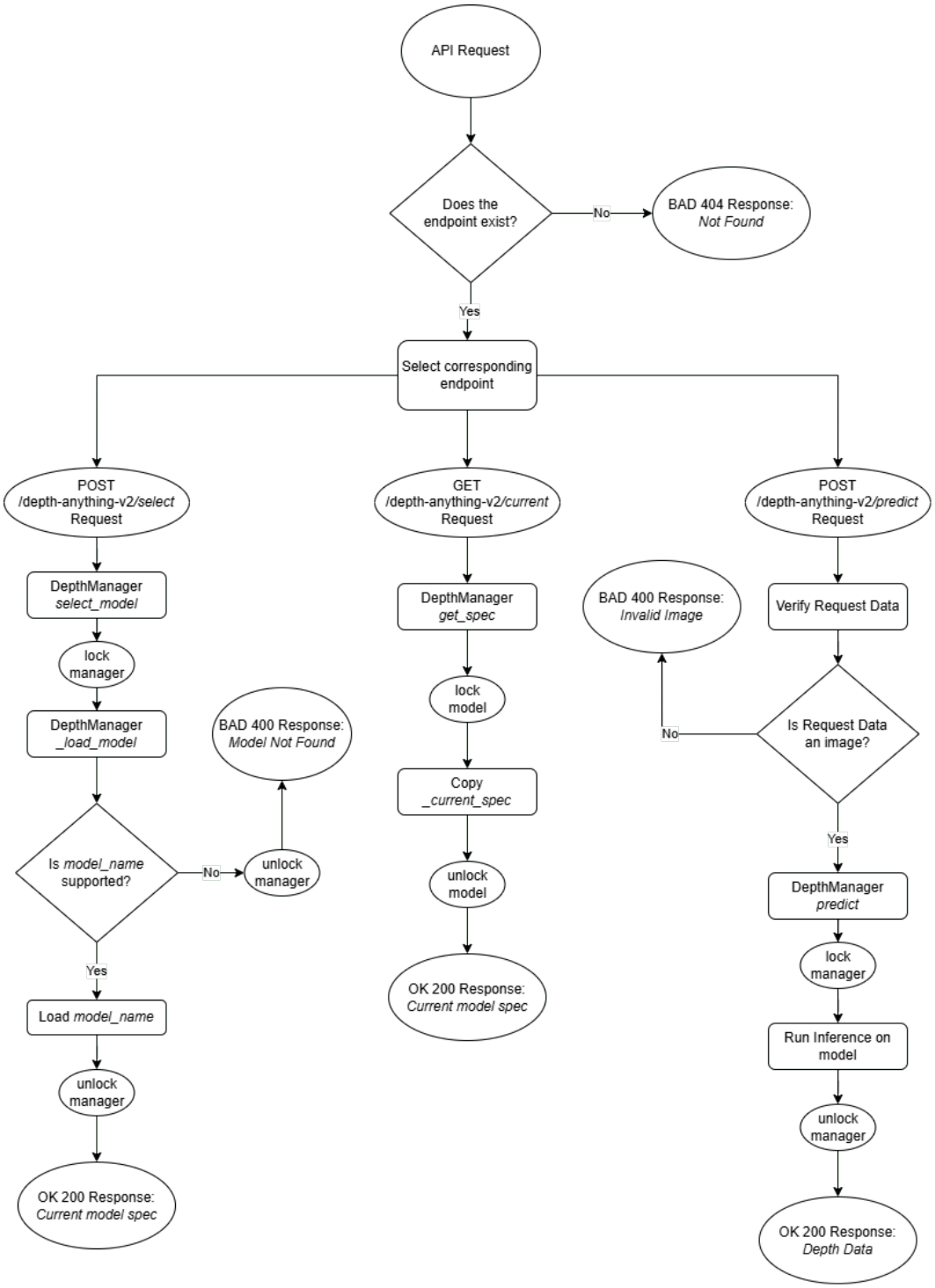


Figure 7.1: Flow diagram for DepthAnything-v2 API request handling. The back-end routes requests based on endpoint, validates inputs, locks the model, performs inference, and returns formatted responses. Thread-safe locking ensures consistent behavior during concurrent requests.

### 7.2.1 Dependencies

The backend uses a focused set of Python libraries. **torch** and **torchvision** serve as the backbone for model loading and inference. **einops** and **timm** assist with tensor reshaping and transformer-based model access. For image handling, the system relies on **numpy**, **pillow**, and **opencv-python**, while **fastapi**, **pydantic**, and **starlette** enable Application Programming Interface (API) creation, request validation, and asynchronous execution.

Two external repositories are integrated into the backend to support the vision transformer models evaluated in this thesis:

- `romatch` *(Tiny RoMa)*: Imported from the official **RoMa** repository[1], this package provides the lightweight Tiny RoMa model used for dense stereo correspondence. It includes custom layers, transformer attention blocks, and match decoding utilities.
- `depth_anything` *(Depth-Anything-V2)*: Integrated from the official **Depth-Anything-V2** repository[2], this module supports loading various ViT-based depth estimation backbones.

### 7.2.2 Model Loading and Inference

The backend manages two fine-tuned models in memory: one for dense stereo matching using **Tiny RoMa**, and one for monocular depth estimation using **Depth-Anything-V2**. To ensure thread-safe access in concurrent environments, all model operations, including loading, prediction, and retrieval are wrapped in locking mechanisms. This prevents race conditions and guarantees that the model state remains consistent across requests. Once loaded, both models persist in memory and are reused for subsequent inference, eliminating redundant initialization and improving performance.

[1] `https://github.com/Parskatt/RoMa`
[2] `https://github.com/DepthAnything/Depth-Anything-V2`

### 7.2.3 Design Patterns

The backend architecture incorporates several design patterns to promote modularity, safety, and maintainability:

- **Service Pattern:** Model-specific logic is encapsulated in manager classes (e.g., **DepthManager**, **RomaManager**), separating business logic from routing.
- **Singleton with Lazy Initialization:** Models are loaded on first use and cached for reuse, avoiding repeated initialization and reducing startup time.
- **Thread-Safe Locking:** Access to shared model instances is protected using locks, ensuring safe behavior under concurrent requests.
- **Modular Router Pattern:** FastAPI routers are split by functionality, with separate modules for each model type, supporting better code organization and scalability.
- **Dependency Separation:** Although explicit dependency injection is minimal, the structure separates API concerns from core logic, following DI-friendly principles.

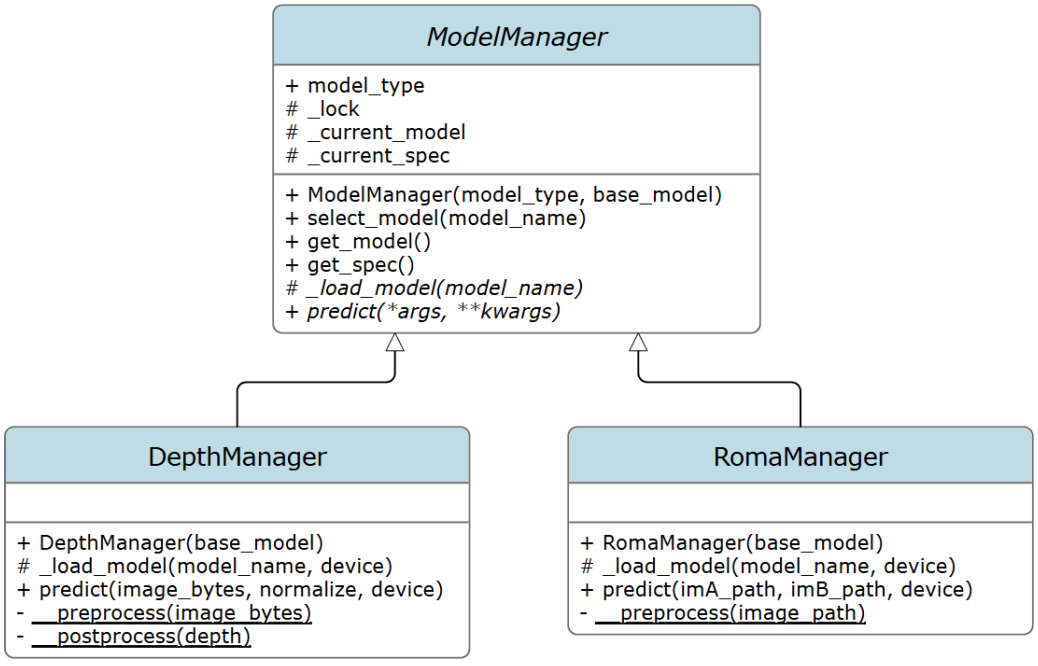


Figure 7.2: UML class diagram for the backend model managers. **ModelManager** is an abstract base class that defines shared functionality such as model selection, locking, and prediction. **DepthManager** and **RomaManager** inherit from it and implement task-specific logic for depth estimation and feature matching, respectively.

### 7.2.4 REST Endpoints

| Endpoint | Description |
|---|---|
| `POST /depth-anything-v2 /select` | Selects the depth model to load.<br>**Input:** JSON with `model_name`.<br>**Response:** Model specification or 400 error. |
| `GET /depth-anything-v2 /current` | Returns the currently active depth model.<br>**Input:** None.<br>**Response:** JSON with current `model_name`. |
| `POST /depth-anything-v2 /predict` | Runs monocular depth estimation on one image.<br>**Input:** Multipart `file` (RGB image).<br>**Response:** JSON with `height`, `width`, and `depth` array. |
| `POST /tiny-roma /select` | Selects the stereo matcher model.<br>**Input:** JSON with `model_name`.<br>**Response:** Model specification or 400 error. |
| `GET /tiny-roma /current` | Returns the currently active Tiny RoMa model.<br>**Input:** None.<br>**Response:** JSON with current `model_name`. |
| `POST /tiny-roma /predict` | Runs Tiny RoMa on two input images for feature matching.<br>**Input:** Multipart `file1`, `file2` (RGB image pair).<br>**Response:** JSON with matched coordinates and metadata. |

Table 7.1: REST API Endpoints and Descriptions

### 7.2.5 Testing the Backend

The backend is tested with around 30 automated tests using **pytest**, organized into distinct categories:

- **Unit tests:** About 10 tests cover model loading logic and utility functions. These check that models handle invalid encoders, missing checkpoints, and produce correct outputs for visualization tasks like colormap generation and keypoint overlay.
- **Integration tests:** 4 tests simulate full inference requests using mocked model managers and in-memory images. This ensures that routing, input parsing, and response formatting work as expected without requiring heavy model files.
- **API route tests:** 8 tests verify endpoint behavior for valid and invalid input cases. These include image uploads, model selection, and checking current model status, with assertions on status codes and response structure.
- **Configuration tests:** A small set of tests confirm that paths, model specs, and required dependencies are correctly defined and importable. This helps prevent environment-related failures before runtime.
- **Manual testing:** Additional HTTP request templates are used for manual validation with tools like REST Client. These help confirm inference behavior with real data during development.

## 7.3 Gradio App

I developed a lightweight **Gradio** interface to be able to debug and test during model integration. The interface allows for interaction with both depth estimation and feature matching models through a simple web GUI, without launching the full React and FastAPI stack.

The app shares all logic and core dependecies with the backend with the exception of the Gradio ui related package, ensuring consistency between the Gradio app and the main API.

The interface includes two tabs, one for each type of model. The **Tiny RoMa** tab allows users to choose the fine-tuned model and the top K matches to display after inference. The **Depth-Anything-V2** tab allows users to choose the fine-tuned model and select the colormap in which to display the depth map. The outputs are rendered using **Matplotlib** and **PIL**.

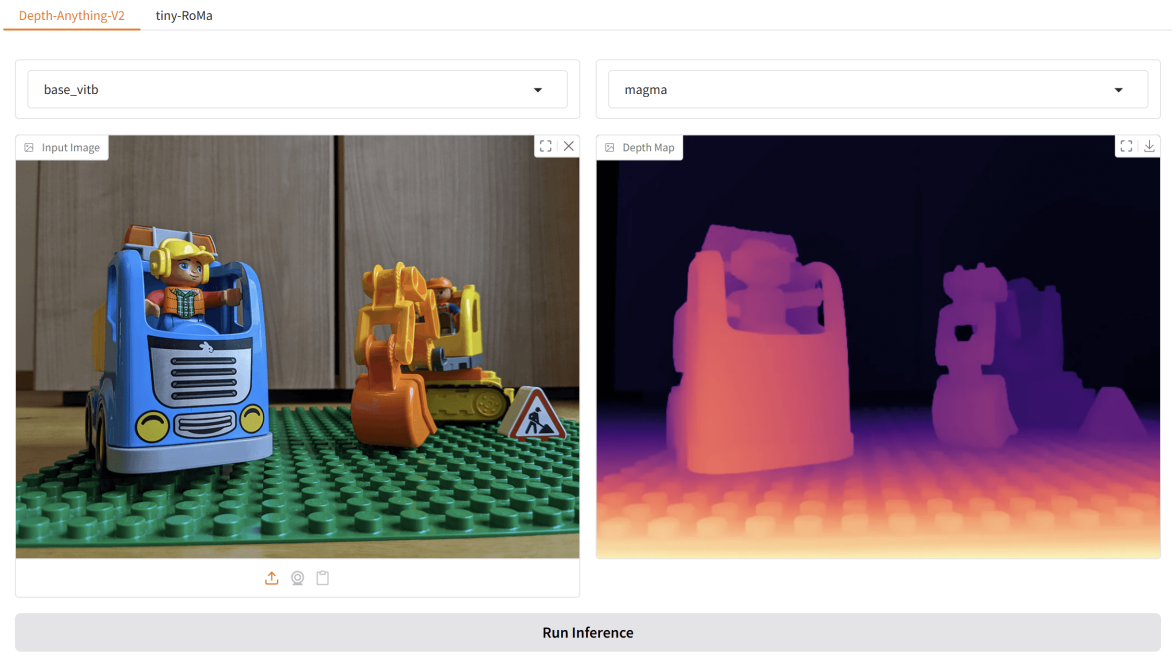


Figure 7.3: Gradio interface for the Depth-Anything-V2 Models.

## 7.4 React: Frontend

The frontend uses the **React + Vite** stack with **TypeScript** and **SWC** for fast, type-safe development and build performance. The UI is built with **Material UI (MUI)**, offering a modern and responsive design. Navigation is handled via **React Router DOM** and backend communication via **Axios**.

The interface supports two core workflows: monocular depth estimation and stereo feature matching, each accessible through a separate tab. After the output is rendered it can be easily downloaded with the click of a button.

The **Tiny RoMa** tab allows users to upload two input images, select the desired fine-tuned model, and then get the match data from the backend. After the match data is retrieved the visualization is rendered on the frontend using the user setting **Top K matches**. If the **Top K matches** slider is changed the visualization is rerendered on the frontend with no extra API calls.

The **Depth-Anything-V2** tab allows users to upload an input image, select the desired fine-tuned model, and then get the corresponding depth data. After the depth data is retrieved the depth map is rendered on the frontend using the selected **colormap**. Similarly, if the **colormap** is changed, the depth map is rerendered on the frontend with no extra API calls.

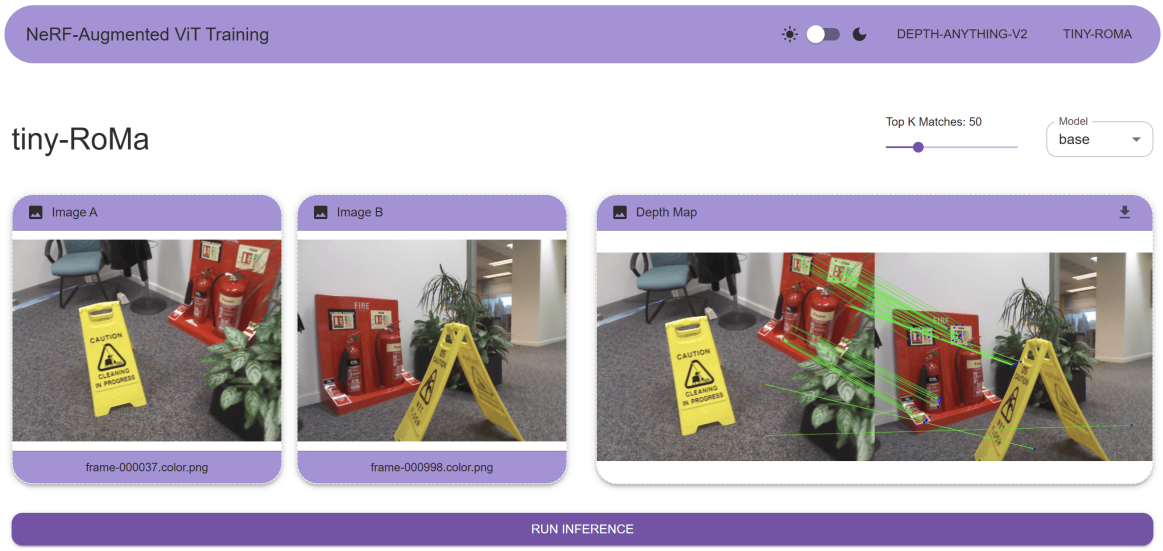


Figure 7.4: Tiny RoMa tab of the React frontend. The tab allows uploading input images, selecting fine-tuned models, and visualizing keypoint matches. Interaction elements like top-$K$ slider update the visualization without additional API calls.

The frontend applies common **React** design patterns to maintain clarity and modularity. Presentational components like **ImageInput** and **ImageOutput** are separated from logic-handling views, following the **container-presentational pattern**. State and events are passed via props to support unidirectional data flow.

### 7.4.1 Component Overview

The interface is built primarily with MUI components for layout, styling, and responsiveness. Most structure, forms, and controls are composed using MUI's built-in elements.

Three custom components were added for key functionality. **Navbar** handles navigation between views and includes a dark mode toggle. **ImageInput** manages file selection and drag-and-drop image upload, with live preview support. **ImageOutput** displays prediction results and includes loading states and a download button.

### 7.4.2 Testing the Frontend

The frontend is tested using **Vitest** and **Testing Library**, with over 20 automated tests organized:

- **UI component tests:** About 15 tests verify the behavior of custom components like **NavBar**, **ImageInput**, and **ImageOutput**. These check user interaction, layout rendering, and props such as theme mode toggling, file uploads, previews, and download actions.
- **API interaction tests:** A set of tests validate integration with the backend API. These simulate requests for model inference and check that the frontend handles loading, errors, and display logic correctly based on mocked responses.
- **Rendering utility tests:** Several unit tests verify that visual outputs like depth maps and keypoint matches are correctly processed and rendered from backend data.

# Chapter 8

# Discussion and Conclusion

## 8.1 Research Summary

This thesis explored the feasibility of using synthetic data generated via NeRF to train Transformer-based computer vision models for two key tasks: **dense feature matching** and **monocular depth estimation**. The research focused on fine-tuning two state-of-the-art architectures: **Tiny RoMa** and **Depth-Anything-V2**.

By training models on partially and fully synthetic data, the study aimed to evaluate how NeRF-generated RGB-D content impacts model performance, generalization, and trianing behavior. Additionally, a full-stack application was developed to demonstrate real-time inference using the trained models and provide an interactive interface for visual inspection of results.

## 8.2 Experimental Insights

The experiments results revealed nuanced effects of NeRF-generated data on model performance. When used as a partial augmentation, synthetic data had a neutral or slightly positive impact. In the case of **Tiny RoMa**, augmenting the **Fire** scene with 15% NeRF-generated frames led to marginal improvements in coarse keypoint matching accuracy without degrading fine-grained performance. This suggests that NeRF data can complement real datasets in feature matching tasks.

In contrast, **Depth-Anything-V2** showed a decrease in accuracy when trained with the same augmented data, indicating that even small proportions of synthetic frames can introduce inconsistencies in monocular depth estimation models.

Furthermore, fully synthetic datasets, such as the **0080** and **0097** scenes, failed to provide sufficient diversity or realism for effective training on their own. The **ViT-B** model showed partial learning, while **ViT-L** exhibited convergence issues and unstable loss behavior.

Overall, the findings suggest that NeRF-generated data is better suited for data augmentation than for standalone training, particularly when model size and task sensitivity to visual artifacts are taken into account.

### 8.2.1 Contributions and Limitations

This thesis contributed a practical evaluation of NeRF-based synthetic data as a training resource for Transformer-based vision models. The main contributions include:

- Creation of partially and fully synthetic datasets combining real and NeRF-rendered RGB-D frames from the **7-Scenes** and **NeRF-Stereo** datasets.
- Fine-tuning and evaluation of **Tiny RoMa** and **Depth-Anything-V2** on both partially and fully synthetic datasets.
- Implementation of a full-stack application with **FastAPI** and **React** for interactive model inference and visualization.

Despite these contributions, the study faced several limitations. NeRF artifacts, low-resolution training data, and limited scene diversity constrained the generalizability of the results. Larger models such as ViT-L proved unstable when trained on fully synthetic data, highlighting sensitivity to data quality. Additionally, due to computational constraints, experiments were limited to a small number of scenes and configurations.

## 8.3 Future Work

Future work should evaluate a broader range of model architectures and sizes to better understand how different capacities respond to NeRF-based supervision. Testing varying percentages of NeRF augmentation would help identify optimal balance points. It is also worth investigating whether adding more NeRF-generated frames can continue to improve performance or introduces diminishing returns.

Experiments on datasets with higher resolution and greater visual diversity could provide more reliable insights into NeRF's effectiveness. Finally, incorporating domain adaptation techniques may help bridge the gap between synthetic and real data performance.

## 8.4 Conclusion

This thesis investigated the effectiveness of NeRF-generated synthetic data for training Transformer-based models in computer vision tasks. The results showed that NeRF data can be a useful augmentation tool for tasks like dense feature matching but is insufficient on its own, particularly for more complex models and monocular depth estimation tasks.

While synthetic data offers clear advantages in scalability and flexibility, its integration into training pipelines must be done carefully, with attention to data quality, model capacity, and task sensitivity. Overall, NeRF-based augmentation shows promise, but hybrid training approaches and further refinement are essential for maximizing its potential.

# Bibliography


[CLY+21] Jieneng Chen, Yongyi Lu, Qihang Yu, Xiangde Luo, Ehsan Adeli, Yan Wang, Le Lu, Alan L Yuille, and Yuyin Zhou. Transunet: Transformers make strong encoders for medical image segmentation. *arXiv preprint arXiv:2102.04306*, 2021.

[DBK+20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[Elg05] Ahmed Elgammal. Cs 534: Computer vision camera calibration, 2005.

[ESB+24] Johan Edstedt, Qiyu Sun, Georg Bökman, Mårten Wadenbäck, and Michael Felsberg. Roma: Robust dense feature matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19790–19800, 2024.

[Hug23] Hugging Face. Monocular depth estimation, 2023. Accessed: 2025-05-27.

[HYP+24] Kewei Hu, Wei Ying, Yaoqiang Pan, Hanwen Kang, and Chao Chen. High-fidelity 3d reconstruction of plants using neural radiance fields. *Computers and Electronics in Agriculture*, 220:108848, 2024.

[Ken21] Mikhail Kennerley. A comparison of sift, surf and orb on opencv. `https://mikhail-kennerley.medium.com/a-comparison-of-sift-surf-and-orb-on-opencv-59119b9ec3d0`, 2021. Accessed: 2025-05-27.

[KHM17] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. *Advances in neural information processing systems*, 30, 2017.

[Kum24] Ankit Kumar. Vision transformers part 4 – understanding different vit architectures. Medium article, 2024.

[LLC+21] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[MESK22] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022.

[MST+21] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[NRK+21] Muhammad Muzammal Naseer, Kanchana Ranasinghe, Salman H Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Intriguing properties of vision transformers. *Advances in Neural Information Processing Systems*, 34:23296–23308, 2021.

[ODM+23] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[Ori11] Emanuele Oriani. qpsnr: A quick psnr/ssim analyzer for linux. *SSIM analyzer for Linux*, 2011.

[PCA+24] Guilherme Potje, Felipe Cadar, André Araujo, Renato Martins, and Erickson R Nascimento. Xfeat: Accelerated features for lightweight image matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2682–2691, 2024.

[Pro19] Prof. Purble. The fundamentals of SfM: For beginners, 2019. Accessed: 2025-05-27.

[Sch20] Johannes L. Schönberger. COLMAP tutorial, 2020. Accessed: 2025-05-27.

[SGZ+13] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the*

*IEEE conference on computer vision and pattern recognition*, pages 2930–2937, 2013.

[SHT15] Christopher Sweeney, Tobias Hollerer, and Matthew Turk. Theia: A fast and scalable structure-from-motion library. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 693–696, 2015.

[Tan22a] Jiaxiang Tang. nerf-template: A simple template for practicing NeRF, 2022. Accessed: 2025-05-27.

[Tan22b] Jiaxiang Tang. Torch-ngp: A pytorch implementation of instant-NGP, 2022. Accessed: 2025-05-27.

[TCD+21] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[TTDGP23a] Fabio Tosi, Alessio Tonioni, Daniele De Gregorio, and Matteo Poggi. The nerf-stereo dataset, 2023.

[TTDGP23b] Fabio Tosi, Alessio Tonioni, Daniele De Gregorio, and Matteo Poggi. Nerf-supervised deep stereo. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 855–866, 2023.

[Wal20] Walsvid. Awesome-mvs: Awesome list of multi-view stereo papers. `https://github.com/walsvid/Awesome-MVS`, 2020. Accessed: 2025-05-27.

[WBSS04] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[XTS+22] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. In *Computer Graphics Forum*, volume 41, pages 641–676. Wiley Online Library, 2022.

[YKH+24] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. *Advances in Neural Information Processing Systems*, 37:21875–21911, 2024.

# List of Abbreviations

**Abs Rel** Absolute Relative Error. 39, 42, 46

**API** Application Programming Interface. 51, 53, 55, 57

**AR** Augmented Reality. 1, 29

**AUC** Area Under the Curve. 30, 33, 34

**CNN** Convolutional Neural Network. 1, 11–13, 15, 16, 29

**DeiT** Data-efficient Image Transformers. 12

**DPT** Dense Prediction Transformer. 14, 38

**Instant-NGP** Instant Neural Graphics Primitives. 1, 10, 18, 21

**mAA** Mean Average Accuracy. 30, 33, 34, 37

**mAP** Mean Average Precision. 30

**MUI** Material UI. 55, 56

**MVS** Multi-View Stereo. 7, 9

**NeRF** Neural Radiance Fields. 1–3, 10, 17–27, 31–34, 36, 37, 40–45, 47, 58–60

**NS** NeRF-Supervised. 18

**ORB** Oriented FAST and Rotated BRIEF. 5, 6, 15

**PCK** Percentage of Correct Keypoints. 30, 33, 37

**PSNR** Peak Signal-to-Noise Ratio. 21, 25, 26, 28

**REST** Representational State Transfer. 49, 53

**RGB** Red-Green-Blue. 13, 23, 26, 29, 38, 40, 43, 48

**RGB-D** Red-Green-Blue-Depth. 1–3, 20, 23, 26, 27, 31, 43, 58, 59

**RMSE** Root Mean Squared Error. 39, 42, 46

**RoMa** Robust Matcher. 2, 3, 15, 16, 29–31, 34, 36, 38, 40, 42, 43, 48, 49, 51, 53–56, 58, 59

**SDFs** Signed Distance Functions. 10

**SfM** Structure-from-Motion. 7–9, 13

**SIFT** Scale-Invariant Feature Transform. 5, 6, 9, 15

**SILog** Scale-Invariant Logarithmic Error. 39, 42, 46

**SLAM** simultaneous localization and mapping. 16, 29

**Sq Rel** Squared Relative Error. 39, 42, 46

**SSIM** Structural Similarity Index Measure. 18, 21, 25, 26, 28

**SURF** Speeded-Up Robust Features. 5, 15

**ViT** Vision Transformer. 1, 11–14, 44, 47