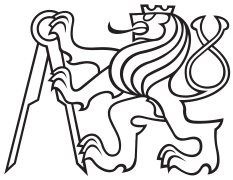
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

In vivo Varroa destructor detection by computer vision

Jafarov Gurban

Supervisor: Ing. Jiří Ulrich
Field of study: Open Informatics
Study Programme: Software Engineering
May 2026

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Jafarov Gurban** Personal ID number: **530773**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Software Engineering**

II. Master’s thesis details

Master’s thesis title in English:

In vivo Varroa destructor detection by computer vision

Master’s thesis title in Czech:

Detekce Kleštíka zhoubného in vivo s využitím počítačového vidění

Name and workplace of master’s thesis supervisor:

Ing. Jiří Ulrich Department of Computer Science FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **06.02.2026** Deadline for master's thesis submission: _____

Assignment valid until: **by the end of summer semester 2026/2027**

Head of department’s signature

Vice-dean´s signature on behalf of the Dean

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work.
The student must produce his thesis without the assistance of others, with the exception of provided consultations.
Within the master’s thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Jafarov Gurban

Student’s signature

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Jafarov Gurban** Personal ID number: **530773**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Software Engineering**

II. Master’s thesis details

Master’s thesis title in English:

In vivo Varroa destructor detection by computer vision

Master’s thesis title in Czech:

Detekce Kleštíka zhoubného in vivo s využitím počítačového vidění

Guidelines:

Research the current methods, software modules, and datasets for Varroa detection and monitoring
Get to know the current state of the art of the robotic beehives and the software systems for their control and data management
Investigate the underlying principles of image-based methods relevant for varroa detection
Propose and formally define a suitable testing benchmark for Varroa detection in living honeybee colonies.
Implement a software module for Varroa detection benchmarking.
Design, implement and evaluate a modular and reproducible software pipeline for Varroa detection in living honeybee colonies. Integrate the pipeline into software framework of a a state-of-the-art robotic beehive.
Investigate the feasibility of exploiting temporal continuity of the images for Varroa detection.
Evaluate the performance of the detection pipeline across selected datasets and discuss the used software design choices.

Bibliography / sources:

- Ulrich, Jiří, et al. "Autonomous tracking of honey bee behaviors over long-term periods with cooperating robots." Science Robotics 9.95 (2024): eadn6848.
- Bjerge, Kim, et al. "A computer vision system to monitor the infestation level of Varroa destructor in a honeybee colony." Computers and Electronics in Agriculture 164 (2019): 104898.
- Bakonyi, Tamás, et al. "Detection of acute bee paralysis virus by RT-PCR in honey bee and Varroa destructor field samples: rapid screening of representative Hungarian apiaries." Apidologie 33.1 (2002): 63-74.
- Kriouile, Yassine, et al. "Nested object detection using mask R-CNN: application to bee and varroa detection." Neural Computing and Applications 36.35 (2024): 22587-22609.
- Giovannesi, Luca, et al. "Vit4V: a Video Classification Method for the Detection of Varroa Destructor from Honeybees." Proceedings of the Computer Vision and Pattern Recognition Conference. 2025.

Acknowledgements

I would like to express my gratitude to my supervisor for guidance and support throughout this work. I also thank the team at the University of Graz for providing access to the AROBA robotic beehive system and the dataset used in this thesis. Finally, I thank the Czech Technical University in Prague for providing the academic environment and resources necessary for this research.

Declaration

I, Jafarov Gurban, hereby declare that the Master's Thesis title "In vivo Varroa destructor detection by computer vision" is my own independent work and that I have declared all sources of information used in accordance with the Methodological Guideline for adhering to ethical principles when elaborating an academic final thesis and the Framework Rules for the use of artificial intelligence at CTU for study and pedagogical purposes in Bachelor and continuing Masters studies. I declare that I have used artificial intelligence tools during the preparation and writing of the final thesis. I've verified the generated content. I confirm that I am aware that I am fully responsible for the content of the final thesis

In Prague, 22. May 2026

Abstract

Varroa destructor mites represent one of the most significant parasitic threats to honeybee colonies worldwide, contributing substantially to colony collapse disorder. Traditional detection methods rely on manual inspection techniques such as sticky boards, sugar rolls, and visual examination of capped brood, which are labor-intensive, subjective, and often detect infestations only after substantial colony decline has occurred.

This thesis presents the design, implementation, and evaluation of a modular and reproducible software pipeline for automated Varroa mite detection in living honeybee colonies. A key innovation of this work is the use of near-infrared (NIR) illumination (750 nm) in combination with grayscale cameras, which eliminates visual disturbance to honeybees whose vision does not extend beyond 700 nm. Furthermore, the proposed system operates entirely within the observation hive without removing combs, thereby minimizing stress to the colony and enabling seamless integration with future automated treatment mechanisms.

The thesis provides a systematic survey of current varroa detection methods, progressing from foundational Convolutional Neural Networks through the evolution of region-based detectors to single-stage architectures, culminating in the selection of YOLOv11 as the detection backbone. A formal benchmarking framework for Varroa detection is proposed and implemented, enabling standardized evaluation across datasets.

A custom YOLOv11-based architecture was trained on a dataset of 4,066 annotated NIR grayscale images collected at the University of Graz (80% training, 10% validation, 10% testing), achieving an mAP@50 of 0.793 and F1-Score of 0.824. The pipeline is integrated into the software framework of the AROBA robotic beehive system. A web-based discovery and annotation service has also been created. The goal was to automate annotations and varroa mites discoveries for the bee keepers community. This service will help speed up their work and improve the YOLO model by learning from new data.

The feasibility of exploiting temporal continuity of image sequences for improved detection is investigated, and the performance of the pipeline is evaluated across selected datasets. The results demonstrate that autonomous, non-invasive monitoring of Varroa infestations is viable and can serve as a foundation for future targeted treatment.

Keywords: Varroa destructor, object detection, web-interface, YOLO, honeybee monitoring, near-infrared imaging

Supervisor: Ing. Jiří Ulrich

Abstrakt

Roztoči Varroa destructor představují jednu z nejvýznamnějších parazitických hrozeb pro včelí kolonie na celém světě a zásadním způsobem přispívají ke kolapsu včelstev (Colony Collapse Disorder). Tradiční metody detekce se opírají o manuální kontrolní techniky, jako jsou lepicí podložky, cukerné testy a vizuální vyšetření zavíčkovaného plodu, které jsou pracné, náročné, subjektivní a často odhalí napadení až poté, co došlo k výraznému úpadku kolonie.

Tato diplomová práce představuje návrh, implementaci a vyhodnocení modulárního a reprodukovatelného softwarového řešení pro automatizovanou detekci roztočů *Varroa* ve včelích koloniích. Klíčovou inovací této práce je využití infračerveného (NIR) osvětlení (750 nm) v kombinaci s kamerami snímajícími ve stupních šedi, což eliminuje rušení včel, jejichž viditelné spektrum nepřesahuje vlnovou délku 700 nm. Navrhovaný systém navíc pracuje zcela uvnitř pozorovacího úlu bez nutnosti vyjímání plástů, čímž minimalizuje stres kolonie a umožňuje bezproblémovou integraci s budoucími automatizovanými ošetřovacími mechanismy.

Práce poskytuje systematický přehled metod detekce objektů, postupující od základních konvolučních neuronových sítí (CNN) přes vývoj detektorů regionů (R-CNN, Fast R-CNN, Faster R-CNN) až po jednostupňové architektury, které vrcholí výběrem YOLOv11 jako detekčního základu. Je navržen a implementován formální testovací rámec pro detekci roztoče *Varroa destructor*, který umožňuje standardizované vyhodnocení napříč datovými sadami.

Vlastní architektura založená na YOLOv11 byla natrénována na datové sadě 4 066 anotovaných NIR snímků ve stupních šedi, pořízených na Univerzitě ve Štýrském Hradci (80 % trénovací, 10 % validační, 10 % testovací data), přičemž bylo dosaženo mAP@50 hodnoty 0,793 a F1-skóre 0,824. Řešení je integrováno do softwaru robotického včelího systému AROBA.

Je zkoumána proveditelnost využití časové kontinuity obrazových sekvencí pro zlepšení detekce a výkon řešení je vyhodnocen napříč vybranými datovými sadami. Výsledky ukazují, že autonomní, neinvazivní monitorování napadení roztoči *Varroa* je realizovatelné a může sloužit jako základ pro budoucí cílené ošetřovací mechanismy.

Klíčová slova: Varroa destructor, detekce objektů, webové rozhraní, YOLO, monitorování včel, infračervené zobrazování

Překlad názvu: Detekce Kleštíka zhoubného in vivo s využitím počítačového vidění

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

Varroa destructor mites (see Fig. 1.1) are obligate ectoparasites of the Western honeybee (*Apis mellifera*) and are now one of the most damaging threats to managed apiaries worldwide. The mite feeds on the hemolymph and fat body of adult bees and developing brood and acts as a vector for several bee viruses; in heavily infested colonies, the most visible consequence is Deformed Wing Virus (DWV), whose typical symptoms include malformed wings and a shortened abdomen [1]. Without active control, infestation rates of more than 50% are common in many regions, and untreated colonies usually collapse within two to three years [1].

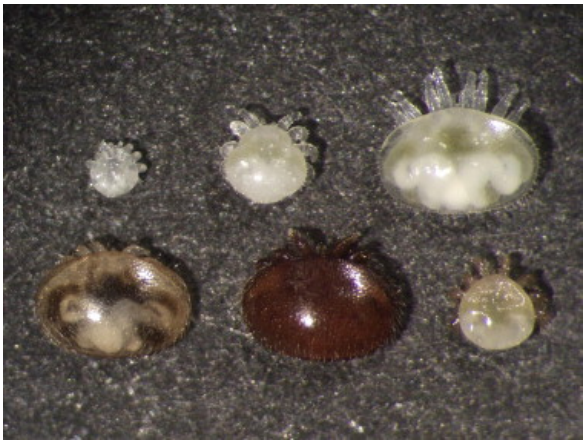


Figure 1.1: Varroa mites (taken from [1]).

The significance of this problem extends well beyond apiculture. Honeybees pollinate roughly three-quarters of the world's leading food crops, so the ongoing decline of managed colonies has direct food-security implications. High annual colony mortality rates in parts of Europe and North America have been reported alongside the rise of Varroa-driven decline [2]. Early and reliable detection of Varroa infestation is therefore not only an entomological problem but an agricultural one.

Traditional detection and monitoring of Varroa infestations rely on manual

inspection techniques, including sticky boards placed beneath hives to capture fallen mites, sugar roll or alcohol wash methods that dislodge mites from adult bees, and direct visual examination of capped brood cells [3]. These methods are inherently limited: they are time-consuming, requiring significant labor per hive; subjective, depending on the experience and judgment of the individual beekeeper; and they provide only indirect infestation metrics that reflect past rather than current mite levels. Most critically, beekeepers often detect severe infestations only after substantial colony decline has already occurred, severely limiting intervention opportunities.

1.2 Problem Statement

To automate or simplify the solution to this problem, there are already Varroa detection methods on the market that can be used. Existing computer vision systems for automated Varroa detection have primarily employed RGB (color) imaging. However, RGB cameras capture visible light wavelengths (400–700 nm) that fall within the honeybee visual spectrum and can cause visual discomfort or behavioral disturbance during extended monitoring periods [4]. The honeybee eye is sensitive to ultraviolet (300–400 nm), blue (400–500 nm), and green (500–600 nm) wavelengths, with sensitivity tapering off above 650 nm and effectively ceasing beyond 700 nm, and continuous exposure to artificial illumination in these ranges may stress the colony or disrupt natural foraging and reproductive behaviors. Furthermore, most existing systems require the removal of combs from the hive for studio-based imaging, which causes substantial stress to the colony, disrupts brood temperature regulation, and interrupts normal bee activities.

The central challenge addressed in this thesis is the development of a non-invasive, real-time detection system for Varroa mites that operates within the hive without disturbing the colony, using imaging technology that is invisible to honeybees.

1.3 Related Work Overview

Early automated work focused on analysing sticky-board photographs and brood patterns. More recent systems detect mites directly on bee hosts using deep learning and advanced feature extraction pipelines. Bilik et al. [5] compared YOLO and SSD detectors on a custom dataset, while Le et al. [6] benchmarked six YOLO variants on the VarroaDataset, reporting a best mAP@50 of 0.906 for YOLOv12x under controlled laboratory imaging. To bridge the gap between classification accuracy and complex background noise, Duma et al. [7] implemented a cooperative segmentation network coupled with an attention-enhanced YOLOx detector, and Noriega-Escamilla et al. [8] explored hand-crafted multi-channel Legendre–Fourier moments to robustly isolate mite features across different color spaces. For continuous field deployment, Bjerge et al. [9] built a portable monitor placed at the hive

entrance. Most of these approaches either require comb extraction for studio imaging or only see forager bees as they enter and exit the hive. Chapter 2 reviews this prior work in detail.

1.4 Goals of the Thesis

The primary objectives of this thesis, aligned with the assignment guidelines, are as follows:

1. Research the current methods, software modules, and datasets for Varroa detection and monitoring.
2. Get to know the current state of the art of the robotic beehives and the software systems for their control and data management.
3. Investigate the underlying principles of image-based methods relevant for Varroa detection.
4. Propose and formally define a suitable testing benchmark for Varroa detection in living honeybee colonies.
5. Implement a software module for Varroa detection benchmarking.
6. Design, implement and evaluate a modular and reproducible software pipeline for Varroa detection in living honeybee colonies. Integrate the pipeline into the software framework of a state-of-the-art robotic beehive.
7. Investigate the feasibility of exploiting temporal continuity of the images for Varroa detection.
8. Evaluate the performance of the detection pipeline across selected datasets and discuss the used software design choices.

1.5 Structure of the Thesis

The rest of the thesis is organized as follows. Chapter 2 covers the theoretical background on object detection and reviews prior work on Varroa detection and robotic beehive monitoring. Chapter 3 describes our approach, including the NIR imaging setup, the model, and how the pieces fit together. Chapter 4 presents the web-based detection and collaborative annotation service built around the model (see Figure 4.1): the system architecture, the technology stack, the annotation workflow, and the load tests used to evaluate it. Chapter 5 reports the main experimental results on detection quality and discusses the findings. Chapter 6 concludes with a summary of contributions and directions for future work.

Chapter 2

Background and State-of-the-Art

2.1 Theoretical Background

This section traces the evolution from CNNs to modern object detectors and motivates the choice of YOLOv11 as the detection backbone.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) introduced fundamental architectural principles for processing spatial data through learnable hierarchical feature representations [10]. Unlike traditional image processing approaches that rely on hand-crafted features (e.g., Haar cascades, HOG descriptors, SIFT/SURF keypoints), CNNs automatically learn relevant patterns from data through end-to-end training.

A CNN consists of three primary types of layers:

- **Convolutional layers** apply a set of learnable filters (kernels) to the input, producing feature maps that capture local spatial patterns. Each filter slides across the input image, computing a dot product between the filter weights and the local input region. This operation exploits two key properties: local connectivity (each neuron connects to only a small region of the input, reducing parameters) and weight sharing (the same filter is applied across the entire spatial extent, enforcing translation equivariance).
- **Pooling layers** progressively reduce the spatial dimensions of feature maps while retaining the most salient information. Max-pooling, the most common variant, selects the maximum value within each pooling window, providing a degree of translation invariance and reducing computational cost.
- **Fully connected layers** at the network's end flatten the spatial feature maps into a one-dimensional vector and perform classification based on the learned feature representations.

A critical property of CNNs is their ability to learn a hierarchy of increasingly abstract features: early layers learn low-level features such as edges

and textures, middle layers combine these into shapes and object parts, and deep layers capture high-level semantic features. This hierarchical feature extraction enables CNNs to learn rich, discriminative representations without manual feature engineering.

2.1.2 Why CNNs Alone Are Not Sufficient for Object Detection

While CNNs revolutionized image classification - assigning a single label to an entire image - they are fundamentally limited when applied to object detection tasks. Object detection requires not only recognizing *what* objects are present in an image but also determining *where* each object is located, typically expressed as bounding box coordinates.

A standard CNN classifier has several critical limitations for detection:

1. **Fixed input size.** Classification CNNs require fixed-size of objects (e.g., 224×224 pixels), but objects in natural images appear at varying scales and aspect ratios.
2. **Single output.** A classifier produces one label per image, but real-world scenes contain multiple objects of different classes at different locations.
3. **No localization.** Standard classification architectures have no mechanism to output bounding box coordinates. The fully connected layers discard spatial information during the flattening operation.
4. **Sliding window inefficiency.** A naive approach to detection using CNNs is to slide a classification window across the image at multiple scales, classifying each patch independently. This exhaustive search is computationally prohibitive.

These limitations motivated the development of specialized object detection architectures that build upon CNN feature extraction while adding mechanisms for efficient region proposal and bounding box regression.

2.1.3 R-CNN: Region-Based Convolutional Neural Networks

R-CNN, proposed by Girshick et al. [11], pioneered the region-based approach to object detection. The method employs Selective Search[1] to identify approximately 2,000 region candidates, followed by feature extraction where each region is resized and passed through a pre-trained CNN to extract feature vectors, which are then classified by Support Vector Machines (SVMs). While R-CNN achieved breakthrough accuracy (53.3% mAP on PASCAL VOC), its computational cost made it impractical for real-time applications,

[1]Selective Search is a region proposal algorithm that segments the image into small regions and iteratively merges similar neighboring regions to generate candidate object locations.

as processing 2,000 regions per image with full CNN forward passes requires excessive computational time.

2.1.4 Fast R-CNN

Fast R-CNN [12] addressed R-CNN's computational bottleneck by introducing a unified architecture that processes the entire image once through convolutional layers. The method applies Region of Interest (RoI) pooling to extract fixed-size feature maps for each region proposal directly from shared convolutional feature maps. This eliminated redundant CNN computations and unified classification and bounding box regression into a single multi-task training process, achieving 9× faster training and 213× faster inference compared to R-CNN. Despite these improvements, Fast R-CNN still relied on Selective Search for region proposals (∼2 seconds per image), preventing real-time operation.

2.1.5 Faster R-CNN

Faster R-CNN [13] addressed the region proposal bottleneck by introducing the Region Proposal Network (RPN), a small convolutional network that slides over the shared feature map, predicting at each spatial location whether an object is present and estimating bounding box coordinates using anchor boxes. The RPN shares its convolutional feature maps with the detection network, enabling nearly cost-free region proposals. Faster R-CNN achieved 5–17 FPS with state-of-the-art accuracy (73.2% mAP on PASCAL VOC 2007), but retained the two-stage detection paradigm, inherently limiting inference speed.

2.1.6 YOLO: You Only Look Once

YOLO, proposed by Redmon et al. [14], revolutionized object detection by reframing it as a single-stage regression problem. The architecture processes the entire image through a single neural network that simultaneously predicts bounding box coordinates, objectness scores, and class probabilities in one forward pass. The input image is divided into an $S \times S$ grid, where each cell predicts B bounding boxes and C class probabilities, producing an output tensor of size $S \times S \times (B \cdot 5 + C)$, where the factor of 5 accounts for the four bounding box coordinates (x, y, width, height) plus one confidence score per box. Key advantages of YOLO include real-time speeds , fewer background errors through global image reasoning, highly generalizable features, and a simple single-network design.

2.1.7 YOLOv11 Architecture

While the original YOLO [14] established the single-stage paradigm, subsequent versions have substantially redesigned the backbone, neck, and detection head. YOLOv11 [15] is the version adopted in this thesis. It retains the

overall feature-pyramid structure of its predecessors but introduces three architectural changes that are particularly relevant for small-object detection such as Varroa destructor mites. Chapter 3.2 describes the architecture in details.

2.1.8 Comparative Analysis and Model Selection

The family of object detectors described above offers two distinct design philosophies: two-stage detectors (R-CNN, Fast R-CNN, Faster R-CNN) optimise accuracy at the cost of inference latency, while single-stage detectors (YOLO and its successors) optimise inference latency while remaining competitive in accuracy. This subsection summarises empirical evidence from five recent studies in which YOLO variants are directly benchmarked against other detectors, in order to justify the choice of YOLOv11 for in-hive Varroa detection.

Table 2.1 gathers the principal results of these benchmarks together with the dataset used in each study.

Table 2.1: Reported comparisons of YOLO variants against other detectors. Datasets and metrics are taken directly from each cited study.

Study	Dataset	Headline result
Gholinavaz et al. [?]	DAWN (adverse-weather driving images: fog, rain, snow, sand)	YOLOv8m reaches mAP@0.5 = 0.753 on the rain subset; YOLOv10n runs at ∼6.7 ms/image (>150 FPS) with only 2.27 M parameters and a 5.8 MB model, while Faster R-CNN is the slowest and obtains the lowest mAP among the compared models.
Yinkfu et al. [16]	Kigali motorbike dataset, 198 annotated images	YOLOv5: mAP@0.5 = 0.8511, 24.7 FPS; Faster R-CNN: mAP@0.5 = 0.7592, 9.27 FPS; SSD: mAP@0.5 = 0.4200, 49.67 FPS; RetinaNet: mAP@0.5 = 0.8020, 8.79 FPS.
Kumar [17]	Container damage dataset, 278 annotated images	YOLOv12: mAP@0.5 = 0.8190; YOLOv11: mAP@0.5 = 0.8190; RF-DETR: mAP@0.5 = 0.7770.
Ghahremani et al. [18]	Solar-panel defect dataset	YOLOv11 reaches mAP@0.5 = 94.1%, while Faster R-CNN reaches only 62.7% on the same data.

Three consistent patterns emerge from Table 2.1. First, YOLO variants achieve substantially faster inference speeds than two-stage detectors, often by a factor of two to three, with lightweight variants reaching over 100 FPS. Second, modern YOLO models are compact-typically 2–6 MB for parameter-efficient variants-enabling deployment on embedded hardware. Third, the accuracy gap between single-stage and two-stage detectors has narrowed significantly, with recent YOLO versions matching or exceeding Faster R-CNN performance on several benchmarks while maintaining real-time throughput.

In the specific context of bee colonies, several prior Varroa-detection works have already converged on YOLO-based architectures: Bilik et al. [5] bench-

mark YOLOv5 against SSD and Faster R-CNN on ex-hive bee imagery, and Duma et al. [7] demonstrate detection on hyperspectral data. The convergence of recent work on YOLO detectors, combined with their favorable speed-accuracy-size trade-off, supports their selection for this thesis.

YOLOv11 was selected as the detection backbone for in-hive Varroa detection for five reasons: real-time inference speeds, compact model size suitable for embedded deployment, enhanced small-object detection via the C2PSA spatial-attention block, single-stage architectural simplicity, and mature ecosystem with prior applications to Varroa detection.

2.2 Previous Results and Related Work

This section reviews six representative prior works on automated Varroa destructor detection. They cover classical object detection, hyperspectral imaging, YOLO architecture benchmarking, entrance-based video monitoring, edge/IoT pipelines, and moment-based classification. For each one the methodology, dataset, and reported performance are summarised, together with how the approach relates to the presented work.

2.2.1 Object Detection Approaches for Varroa Detection

Bilik et al. [5] were among the first to apply modern single-stage object detectors directly to Varroa mites on honeybee hosts. The study compared two detector families - YOLO and SSD - against a Deep SVDD anomaly-detection baseline, and serves as a useful starting point for end-to-end detection pipelines in apiculture.

Methodology. The study framed Varroa detection as a multi-class object detection problem with six target categories: healthy bee, pollen-carrying bee, drone, queen, infected bee, and Varroa mite (V-mite). These six classes were subsequently reorganized into three experimental subsets to evaluate the detectors under varying levels of task complexity: (1) full six-class detection, (2) a reduced V-mites only task focusing exclusively on mite localization, and (3) a binary healthy versus infected classification task. For YOLO-based detection, both the small and extra-large variants were trained. For SSD, two backbone architectures were compared: VGG-16 and MobileNetV2. Additionally, a Deep SVDD anomaly detector was evaluated under the hypothesis that infested bees might constitute anomalies relative to healthy bee representations in a learned latent space.

All networks were trained on 640×640 pixel input resolution. Both YOLO and SSD used default pretrained weights as initialization. The Deep SVDD model was trained using a custom autoencoder.

Dataset. The dataset comprised 803 RGB images compiled from two sources: 500 images from diverse online and apiary sources captured under general environmental conditions (variable backgrounds, lighting, and bee

orientations), and 303 images from the publicly available VarroaDataset by Schurischuster and Kampel [19], which features individual bees photographed against artificial white backgrounds under controlled studio conditions. The combined dataset was split into 561 training, 127 validation, and 115 test images. Data augmentation was applied, expanding the effective training set to 24,684 images. While this augmentation substantially increased training sample diversity, it could not fully compensate for the relatively small number of unique source images. The heterogeneity of the source images-combining field photographs with studio images - introduces a domain gap within the training set itself, which may confuse the detector with inconsistent background statistics and illumination profiles. Furthermore, the relatively small test set of 115 images limits the statistical significance of the reported performance metrics.

Performance. Table 2.2 summarizes the key detection metrics reported by Bilik et al. across the three experimental subsets.

Table 2.2: Detection performance reported by Bilik et al. [5] across experimental subsets.

Subset	Model	Recall	F1	mAP@50
V-mites only	YOLOv5x	0.696	0.666	0.752
V-mites only	YOLOv5s	0.746	0.656	0.777
V-mites only	SSD-MobileNetV2	0.593	0.714	0.519
V-mites only	SSD-VGG16	0.322	0.242[2]	0.355
Healthy vs Infected	YOLOv5x	0.848	0.874	0.908
Healthy vs Infected	YOLOv5s	0.852	0.838	0.917

YOLOv5 achieved the best balance on the mite localization task, with F1-scores around 0.66, mAP@50 around 0.76, and recall up to 0.75. SSD variants showed poor recall, missing a substantial fraction of mites. Performance improved substantially on the coarser healthy-versus-infected classification task, demonstrating that whole-image classification is considerably easier than precise localization. Figure 2.1 illustrates representative detection results, highlighting the visual similarity between Varroa mites and bee anatomical features such as compound eyes.

Strengths. Bilik et al. demonstrated end-to-end YOLO/SSD detection on heterogeneous (uncontrolled) Varroa imagery and reported inference times on both desktop and embedded hardware, providing the first deployment-oriented benchmarks for this task.

Limitations. Two limitations are specific to Bilik et al.'s study. First, the original dataset of 803 images, while augmented extensively, provides

[2]The F1-score 0.242 is reproduced verbatim from Bilik et al. [5]. This value is inconsistent with the reported precision (0.980) and recall (0.322), which would yield F1 $\approx$ 0.485. We retain the published number for fidelity to the source.

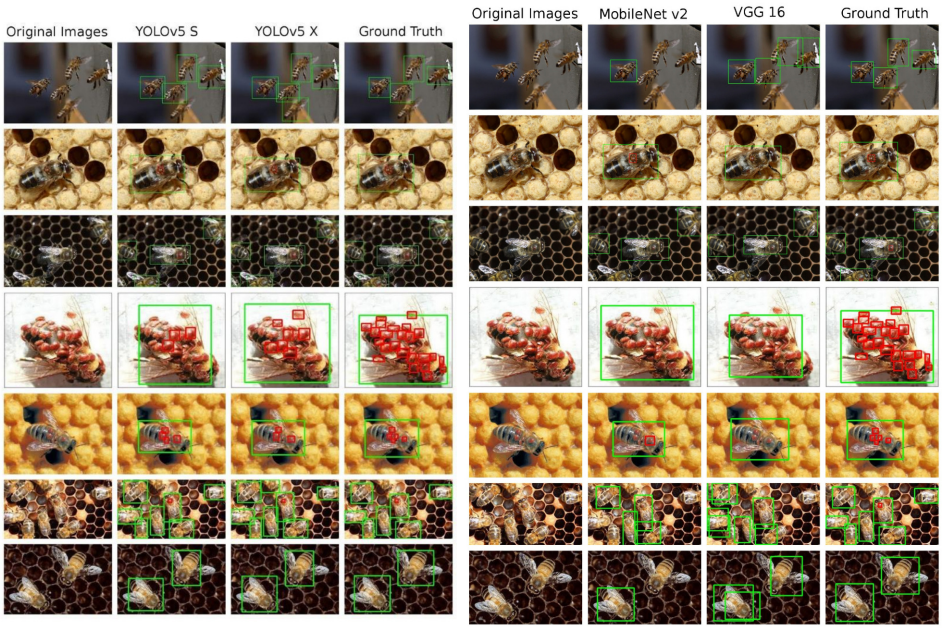


Figure 2.1: Sample detection results from Bilik et al.[5] showing bounding box predictions on honeybee images. The detector must distinguish Varroa mites from visually similar structures such as compound eyes and wing bases.

limited diversity in terms of mite appearance, bee subspecies, and imaging conditions; the heavy reliance on augmentation may lead to models that overfit to the specific noise and transformation patterns introduced during training rather than learning genuinely invariant mite features. Second, Bilik et al. themselves identify visual confusion between Varroa mites and bee compound eyes as a recurring source of false positives under visible-light RGB illumination-a confound that the higher spatial resolution of the presented work's Full HD imaging system (1920×1080 vs. 640×640) further helps to separate. Limitations that Bilik et al. share with most of the related work are discussed jointly in Section 2.2.7.

2.2.2 Hyperspectral Imaging Approaches

Duma et al. [7] took a fundamentally different approach and applied hyperspectral imaging (HSI), which captures hundreds of narrow wavelength bands rather than the three broad channels of a standard RGB camera. Their results show that Varroa mite cuticle and bee body parts have distinct spectral signatures that allow reliable discrimination even from very small datasets, provided the specimens are stationary.

Methodology. The imaging system used a Specim IQ portable hyperspectral camera - a push-broom scanner that records 204 spectral bands across the 400–1,000 nm range at 7 nm spectral resolution and 512×512 pixel spatial resolution. Dead bees and Varroa mites collected from hive floor detritus were

arranged on Petri dishes and white calibration panels, and the hyperspectral data cubes were acquired through line-by-line scanning.

Two classification methods were tested. The first was unsupervised K-means++ clustering, applied to spectra. The second was the supervised Kernel Flows Partial Least Squares (KF-PLS) method, a non-linear regression model using a Matérn 5/2 kernel, trained for 150 iterations with 6 latent variables on 300 labeled pixels per class (Varroa mite, bee wing, bee body, and background) from the calibration set.

A key contribution of the study was identifying the smallest set of wavelengths sufficient for discrimination. Two variable selection algorithms were compared: the COVPROC method and the R^2-based forward selection. The COVPROC method reduced the full 204-band set down to just four wavelengths: three in the blue-green range (492.97 nm, 498.80 nm, 507.56 nm) capturing differences between mite chitin and bee cuticle, and one in the near-infrared (796.74 nm) reflecting differences in surface scattering. The R^2 method identified 12 wavelengths for the unsupervised pipeline, also including a near-infrared band around 800 nm.

Dataset. The dataset consists of six hyperspectral images of dead bees and Varroa mites from hive detritus . Each image is a $512 \times 512 \times 204$ data cube. The six images were divided into a calibration set (specimens separated on Petri dishes) and a held-out test set (specimens in dense, clustered arrangements on a white reference panel), intentionally using different backgrounds to test generalization. The dataset is publicly available [20]. The small size of six images, collected in a single season and region, limits the assessment of generalization across diverse colony and seasonal conditions.

Performance. With K-means++ clustering on PCA-reconstructed spectra, at least four clusters were needed to separate mite pixels from bee pixels (Figure 2.2). Under this configuration, no mite pixels were misclassified on the test image – mites were correctly identified even when sitting on top of bees. The KF-PLS classifier also successfully located all mites. These results confirm that the spectral signature of Varroa chitin is highly distinctive at sufficient spectral resolution.

Strengths. Reducing the discriminative information to just four wavelengths opens a practical path to simplified hardware: a purpose-built sensor with four narrow-bandpass filters could potentially match the performance at a fraction of the cost of a full hyperspectral camera. Also, the unsupervised Kmeans++ approach requires no labeled data, which is a significant advantage given how scarce annotated Varroa datasets are

Limitations. All experiments used dead specimens collected from hive floor detritus; the spectral properties of dead mites may differ from live ones due to desiccation and chemical changes in the chitin surface, and the method has not been tested on live bees in a colony. Also, the push-broom scanner requires the subject to be stationary during acquisition, which is

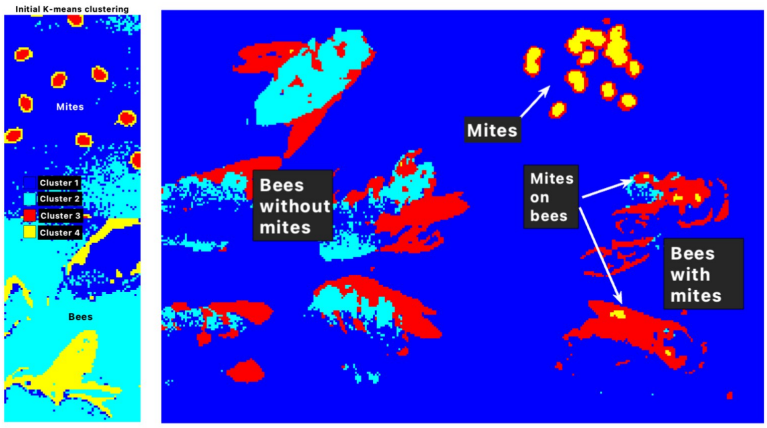


Figure 2.2: Spectral reconstruction and PCA-based cluster visualization from Duma et al.[7]. Principal component projections reveal distinct spectral clusters corresponding to Varroa mites, bee body parts, and background material.

fundamentally incompatible with real-time monitoring of moving bees inside a hive.

Processing 204-band hyperspectral cubes carries substantially higher computational cost than single-channel grayscale images and would be difficult to deploy on the embedded hardware.

2.2.3 YOLO Architecture Benchmarking on Controlled Datasets

Le et al. [6] conducted a systematic benchmarking of six YOLO generations: v5, v8, v9, v10, v11, and v12 - for Varroa mite detection, comparing successive architectural innovations on the same task under identical training conditions. A Faster R-CNN baseline with a ResNet-50-FPNv2 backbone was also included for reference.

Methodology. All six YOLO variants used the extra-large (x) model size. Training hyperparameters were identical across all models: input resolution 640×640, so that any performance differences reflect architectural changes.

Dataset. Two datasets were used. Dataset1[3] derives from the publicly available VarroaDataset [19], containing 13,509 images of individual bees recorded under laboratory conditions, with bounding box annotations for visible mites. Only infected images were used, giving 2,762 training, 592 validation, and 592 test images (4,626 mite instances). Dataset2 is a custom set of 1,820 images from online videos of infested bees, manually annotated and validated by apiculture experts, split into 1,274 training, 273 validation, and 273 test images (3,359 mite instances). Dataset1 contains isolated single-bee images under controlled conditions; Dataset2 contains multi-bee natural scenes.

[3]The dataset names “Dataset1” and “Dataset2” are internal labels used by Le et al. [6]

Performance. The substantially lower AP@50:95 on Dataset1 (0.33–0.35) compared to Dataset2 (0.81) is partly attributed to annotation quality issues: many ground-truth bounding boxes in Dataset1 were larger than the actual mite size, causing models to learn irrelevant background regions and reducing IoU between ground-truth and predicted boxes. Faster R-CNN achieved the highest recall on Dataset1 but the lowest AP@50:95 and the slowest inference.YOLOv12x leads in accuracy and localisation quality on both datasets, while YOLOv10x/v11x are the fastest at inference/

Strengths. Identical training conditions across six YOLO generations isolate the effect of architecture from dataset or recipe differences. The inclusion of Dataset2 with real-world multi-bee scenes quantifies the gap between lab and natural conditions.

Limitations. Dataset1 (VarroaDataset) requires extracting combs from the hive and photographing individual bees in isolation, which precludes continuous monitoring and cannot be reused for in-hive deployment. Second, Le et al. themselves report annotation quality issues in Dataset1 that limit the reliability of the AP@50:95 scores.

2.2.4 Entrance-Based Monitoring Systems

Bjerge et al. [9] built a portable computer vision system that estimates Varroa infestation levels from video of bees at the hive entrance. It is one of the earlier attempts to move detection out of the lab and toward something a beekeeper could deploy in the field.

Methodology. The system employed a custom-designed portable video monitoring unit with multispectral illumination, placed in front of the beehive entrance (Figure 2.3). The monitoring procedure required bees from a selected frame to be shaken off into the entrance area, where they were recorded on video as they walked back into the hive. The video processing pipeline, termed the Infestation Level Estimator (ILE), operated in three stages: (1) background subtraction to segment individual bees from the entrance platform, (2) a CNN-based classifier to detect the presence of Varroa mites on segmented bee regions, and (3) a counting module that aggregated bee and mite counts across the video sequence to compute the colony infestation percentage.

Dataset. The evaluation was based on a video sequence containing 1,775 bees and 98 visible Varroa mites, captured at a single apiary. The ground truth was established through manual frame-by-frame annotation of the video.

Performance. At the individual detection level, the system achieved an F1-score of 0.97 for counting bees and an F1-score of 0.91 for detecting mites on individual bees. These metrics indicate strong detection performance, particularly for the bee-counting component, which benefits from the relatively simple entrance-platform background.

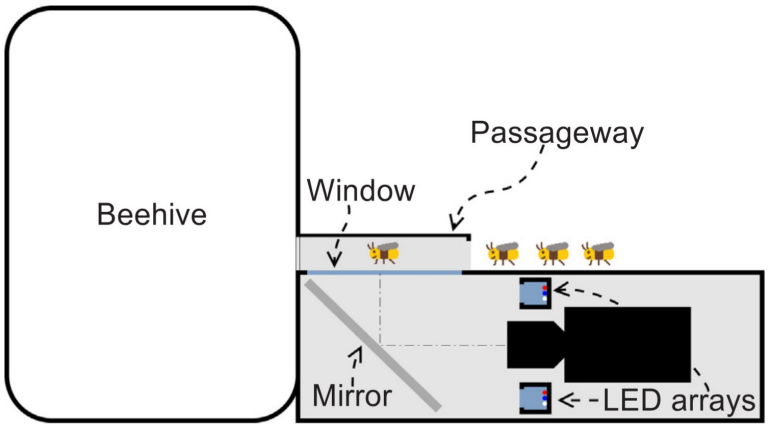


Figure 2.3: The portable video monitoring hardware setup from Bjerge et al. [9], positioned at the hive entrance for recording bees as they return to the colony.

Strengths. The entrance-mounted hardware is external to the hive and replicable across many colonies, supporting large-scale surveillance without opening the brood chamber. The uniform entrance platform simplifies background subtraction, and the resulting infestation percentage is a directly actionable metric for treatment decisions. Multispectral illumination further improves mite/bee contrast compared to broadband visible light.

Limitations. While the video format provided a large number of individual bee observations, it represented only a single colony at a single time point, limiting the assessment of generalization across colonies, seasons, and environmental conditions. The principal limitation is the entrance-only observation paradigm. The system observes only forager bees entering and exiting the hive, which represent a biased subsample of the colony population; forager bees have different Varroa prevalence rates than nurse bees and house bees due to their age and behavioral profile. Nurse bees, which remain within the hive and tend to brood cells where Varroa mites preferentially reproduce during their phoretic phase, are never captured by entrance monitoring. Consequently, mite infestations concentrated within the brood area - where they cause the most damage through parasitism of developing pupae - may remain undetected until the infestation has propagated to the forager population, potentially resulting in delayed intervention. In addition, the initial setup requires bees to be shaken from frames onto the entrance platform, which constitutes a non-trivial colony disturbance and requires a beekeeper to be present, and the system fundamentally cannot serve as a foundation for automated in-hive treatment, since mites must be detected within the hive to enable targeted intervention on specific bees or brood cells. Generic limitations are discussed in Section 2.2.7.

2.2.5 Edge Computing and IoT-Enabled Monitoring

Wachowicz et al. [21] combined embedded inference with cloud-side IoT infrastructure: detection runs on a small edge device, results are pushed to the cloud, and notifications are dispatched to the beekeeper. The work is interesting because it covers the whole pipeline - from acquisition to notification - rather than only the detector itself.

Methodology. The detection pipeline employed a two-stage architecture. In the first stage, individual bees were localised in camera frames. In the second stage, each detected bee region was classified as healthy or Varroa-infested using a re-trained GoogLeNet CNN. The overall edge-to-cloud architecture of the system is illustrated in Figure 2.4.

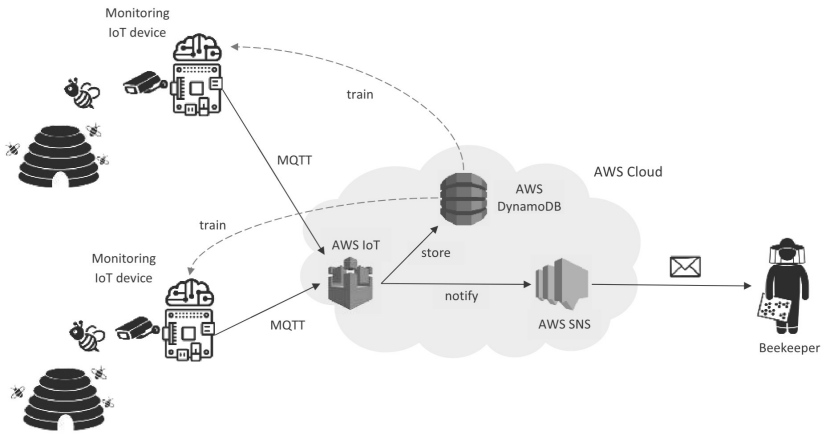


Figure 2.4: IoT architecture for edge-based Varroa monitoring from Wachowicz et al. [21]. The system comprises an NVIDIA Jetson Nano for local inference, MQTT-based communication with AWS IoT Core, DynamoDB for event storage, and SNS for beekeeper notifications.

Dataset. The mite classification dataset comprised 4,800 augmented images (half healthy, half infected bees), generated by applying augmentation techniques to a smaller base set of manually annotated images. From this dataset, 50 images (25 negative and 25 positive) were assigned to the test set; the remainder formed the training set.

Performance. On the Jetson Nano, the GoogLeNet-based mite classifier achieved an accuracy of 80%, with precision 0.89 and recall 0.68 on 50 test images.

Strengths. The principal contribution is the holistic system-level design: Wachowicz et al. cover the complete pipeline from image acquisition through edge inference and MQTT/AWS transmission to SNS-based beekeeper notifications.By performing inference locally and transmitting only compact detection events instead of raw images, the system reduces network and power usage, and the authors show that a CNN-based bee detector clearly

outperforms a traditional adaptive background subtraction method on the same hardware.

Limitations. Two limitations are specific to Wachowicz et al.'s implementation. First, the reported recall of only 68% means that approximately one in three infested bees is missed; the authors themselves note they are primarily concerned with the falsenegative rate, acknowledging that detection of even a single case of varroosis is sufficient and that false positives are of less concern. Second, real-time operation at FullHD resolution was not achievable for either algorithm, requiring either resolution reduction or frame skipping; the IoT pipeline also depends on an external wireless access point to reach the cloud, which restricts deployment in remote apiaries.

2.2.6 Mathematical Moment-Based Classification

Noriega-Escamilla et al. [8] took a non-deep-learning route. They computed Multichannel Legendre–Fourier Moments (MLFMs), a family of orthogonal image moments, and fed the resulting feature vectors into a classical Support Vector Machine. On a controlled dataset, these hand-crafted descriptors competed with - and outperformed - CNN baselines, which is a useful counterweight to the assumption that deep learning is always the right choice.

Methodology. Legendre–Fourier moments are numerical features that describe an object's shape inside a circle. They use one index to describe how the shape changes with distance from the center and another index to describe how it changes around the circle. These moments have three properties that are particularly relevant for insect image classification: they are rotation invariant, scale invariant, and relatively robust to noise. The authors note that scale invariance holds only if the images are cropped to contain only the bee. By computing moments up to order five, the method produces 75 descriptors per image (25 per color channel), encoding shape and color information. The authors also note that experiments with higher-order moments did not improve results, while increasing computation time and introducing numerical instability.

Three color spaces were evaluated: RGB, HSV (Hue, Saturation, Value/Brightness), and YCbCr (luminance and chrominance).

Classification employed a Cubic SVM (Support Vector Machine with a cubic kernel). A ten-fold cross-validation strategy was used to validate the model, and an independent test set from the VarroaDataset was used for final evaluation.

A two-stage classification pipeline was implemented: in the first stage, bee images were classified as dorsal or ventral view; in the second stage, each orientation category was further classified as healthy or Varroa-infested. The motivation for this subdivision was that classifying by orientation allows the algorithm to process additional information about the color and shape of the bee's legs, wings, eyes, and mouth, and to better distinguish the mite from similarly colored bee body parts such as the tongue and eye.

Dataset. The experiments used the VarroaDataset [19], a public database consisting of 13,509 images of healthy bees and bees infested with the Varroa destructor mite at 160 × 280 pixel resolution, taken in a laboratory in a controlled environment and manually labelled. The dataset is pre-split into train, test, and validation subsets. For the two-stage classification experiments, a balanced subset of 2,000 images (500 per class: healthy dorsal, healthy ventral, infested dorsal, infested ventral) was created through manual classification. The full dataset was used for the initial binary (healthy vs. infested) classification experiment.

Performance. On the balanced VarroaDataset subset, the moment-based classifier with a cubic SVM in the YCbCr colour space outperformed both alternative colour spaces (RGB, HSV) and deep learning baselines (YOLOv5 and DeepLabV3). The authors attribute the advantage of YCbCr to the Cr channel, which emphasises the characteristic red coloration of Varroa mites.

Strengths. The rotation and scale invariance of the moment descriptors permits classification on small databases without any image preprocessing or augmentation, and low-order moments yield compact feature vectors that encode shape and colour simultaneously. The two-stage dorsal/ventral subdivision allows orientation-specific features (legs, wings, eyes, mouthparts) to be exploited and helps separate mites from similarly coloured bee anatomy.

Limitations. The authors themselves acknowledge a key limitation: with more extensive databases and transfer learning, deep neural networks can overcome the advantages of the moment-based approach, as they can more easily handle the constant mobility of bees that causes Varroa to be confused with body parts. Furthermore, the method performs classification rather than detection and localization, and the paper does not address bounding box prediction. Additionally, scale invariance is conditional: it only holds if images are properly cropped to contain a single bee, which is not guaranteed in unconstrained hive imaging scenarios.

2.2.7 Summary and Positioning of the Presented Work

The six surveyed approaches demonstrate that automated Varroa detection is feasible across a wide range of imaging modalities and detection paradigms, yet each method addresses only part of what a practical, long-term, autonomous monitoring system requires. The field has progressed from custom CNNs and classical moment descriptors to modern YOLO variants, from RGB to hyperspectral imaging and IoT-integrated pipelines, and toward greater attention to deployment constraints - but the combination of bee-invisible illumination, in-hive operation that captures every bee caste, and real-time detection integrated with treatment infrastructure remains unaddressed.

A consistent trade-off between accuracy and ecological validity runs through the surveyed works: the highest reported numbers (Le et al.'s mAP@50 = 0.906 on isolated single-bee images; Noriega-Escamilla et al.'s 96.7% classification

accuracy on a balanced studio subset; Bjerge et al.'s F1 = 0.91 at the hive entrance) are obtained either under controlled studio conditions, on isolated specimens, or on well-constrained entrance scenes. Systems that operate in more ecologically realistic settings report harder numbers on harder data. None of the surveyed systems simultaneously achieve bee-invisible illumination, in-hive imaging that covers all bee castes and brood, and real-time detection. Datasets also remain modest in scope: the largest available public collection, the VarroaDataset (13,509 images), consists of low-resolution single-bee studio shots at 160×280 px. The 4,066 Full HD in-hive images used in the presented work constitute, to the best of our knowledge, the largest in-hive annotated collection available. Temporal information from video is barely exploited - only Bjerge et al. [9] address it, without modern multi-object tracking - and no surveyed system supports community-driven dataset growth or active learning.

Recurring limitations across the surveyed works. Three limitations recur across the surveyed studies. (i) *Bee-visible illumination.* Bilik et al. [5], Le et al. [6], Wachowicz et al. [21] and Noriega-Escamilla et al. [8] all operate under standard RGB illumination in the 400–700 nm range, which lies fully within the honeybee visual spectrum [22] and may stress the colony under prolonged exposure; even Duma et al. [7] include three bee-visible wavelengths among their four discriminative bands (493, 499, 508 nm). The presented work resolves this by using NIR illumination at 750 nm, which lies entirely outside the bee-visible spectrum. (ii) *Limited dataset size and ecological validity.* Most of the surveyed datasets are small (Bilik et al.: 803 images; Duma et al.: 6 hyperspectral cubes; Wachowicz et al.: 50 test images; Noriega-Escamilla et al.: 2,000 images) or are recorded under controlled studio conditions on isolated single bees (Le et al.'s Dataset1 of 13,509 images at 160×280 px). The present work uses 4,066 Full HD in-hive frames, which is, to the best of our knowledge, the largest annotated in-hive collection currently available, and which captures all bee castes and brood in their natural environment. (iii) *Limited real-time and treatment readiness.* Bilik et al. report approximately 1 FPS on a Jetson Nano, Wachowicz et al. require resolution reduction or frame skipping to achieve real-time inference, Duma et al.'s push-broom hyperspectral acquisition is intrinsically non-real-time, and Bjerge et al. process video offline. None of the surveyed systems are designed to interface with a treatment actuator.

Methodologically, the closest relatives are Bilik et al. [5] (YOLO-style direct mite localization) and Wachowicz et al. [21] (system-level deployment thinking), but the presented work differs from both in imaging modality (NIR grayscale instead of RGB) and monitoring location (in-hive instead of ex-hive or entrance). The benchmarking protocol of Le et al. [6] across multiple YOLO generations provides empirical justification for the architectural choice of YOLOv11. The spectral findings of Duma et al. [7] - in particular the discriminative near-infrared wavelength near 797 nm - provide indirect support for NIR-based detection. The moment-based approach of Noriega-Escamilla et al. [8] and the entrance monitoring of Bjerge et al. [9] are complementary to

rather than competing with the presented work. To the best of our knowledge, this is the first system that combines bee-invisible NIR illumination, in-hive deployment on live bees, real-time deep learning-based mite detection, and integration with a robotic beehive platform designed to support automated treatment. The next chapter describes how this approach is realised in practice.

Chapter 3

Overview of Our Approach

This chapter presents the complete methodology of the proposed Varroa detection system, covering the imaging technology, data acquisition pipeline, dataset composition, model architecture, training procedure, benchmarking framework, and integration with the AROBA robotic beehive system. Additionally, we describe a web-based detection and collaborative annotation service that enables community-driven dataset expansion and model improvement.

3.1 NIR Grayscale Imaging and In-Hive Monitoring

The fundamental design principle of the proposed system is that the monitoring process must be invisible to honeybees. Honeybee photoreceptors are sensitive to ultraviolet (300–400 nm), blue (400–500 nm), and green (500–600 nm) wavelengths, with sensitivity tapering off above 650 nm and effectively ceasing beyond 700 nm [4, 22]. By employing near-infrared (NIR) illumination in the 750 nm range, the imaging system operates entirely outside the bee-visible spectrum. This ensures that continuous, long-term monitoring introduces no visual disturbance, behavioral artifacts, or stress to the colony - a critical requirement for ecologically valid observations and for compatibility with natural colony dynamics.

Grayscale cameras optimized for the NIR spectrum are used instead of conventional RGB cameras. This choice provides three practical advantages. First, NIR-sensitive sensors with back-side illumination (BSI) architecture achieve higher quantum efficiency in the 750 nm range compared to standard RGB sensors, yielding improved signal-to-noise ratios in the low-light conditions typical of hive interiors. Second, grayscale imaging eliminates the Bayer filter mosaic present in color cameras, utilizing the full spatial resolution of the sensor without demosaicing artifacts. Third, single-channel grayscale images reduce data throughput and computational load by a factor of three compared to three-channel RGB images, enabling faster processing at equivalent spatial resolution.

A second critical design principle is that the system operates entirely within

the observation hive without removing combs from the colony. Unlike studio-based imaging approaches that extract combs for controlled photography [6], the proposed system images bees in their natural environment directly above the frame surface. While in-hive imaging presents greater challenges - variable bee density, partial occlusion, less controlled illumination geometry - it preserves brood temperature regulation, avoids disrupting normal bee activities, and enables observation of all bee castes (foragers, nurse bees, the queen, and developing brood). Most importantly, in-hive monitoring is a prerequisite for future integration with automated treatment mechanisms, which require detecting and targeting mites on bees within the colony.

3.2 Model Architecture

Building upon the YOLOv5 and YOLOv8 baselines discussed in Chapter 2, YOLOv11 introduces three specific structural innovations-the C3k2, SPPF, and C2PSA blocks-designed to enhance feature extraction and spatial attention.The detection pipeline employs a custom implementation of YOLOv11, developed from first principles using the PyTorch deep learning framework [23]. The architecture consists of three principal components: a DarkNet-based backbone [24] for multi-scale feature extraction, a Spatial Pyramid Pooling – Fast (SPPF) module for contextual enrichment, and a single-stage detection head for bounding box regression and classification.

C3k2 block. The C2f block used in YOLOv8 is replaced by the C3k2 (CSP with kernel size 2) block [15]. CSP (Cross Stage Partial) is designed to get a richer mix of gradients while doing less computation. It does this by splitting the base-layer feature map into two parts, sending only one part through the heavy layers, and then recombining the two branches later in a cross-stage structure [25]. C3k2 employs smaller convolutional kernels within a CSP-style structure to improve computational efficiency and feature extraction while reducing parameter overhead. C3k2 blocks are reused throughout the backbone, neck, and detection head (see Fig. 3.1).

SPPF module. The Spatial Pyramid Pooling – Fast (SPPF) module is retained from earlier YOLO versions and is placed near the end of the backbone [15]. SPPF aggregates multi-scale contextual information through cascaded max-pooling operations and provides enriched feature maps to the neck. This is particularly important for in-hive imagery, where Varroa mites may appear at different apparent scales depending on bee orientation and camera distance (see Fig. 3.1).

C2PSA block. One of the main architectural additions in YOLOv11 is the C2PSA (Convolutional block with Parallel Spatial Attention) module, inserted after the SPPF stage [15]. By incorporating spatial attention mechanisms, C2PSA improves the network's ability to focus on informative spatial regions and enhances feature representation for small or partially occluded objects.

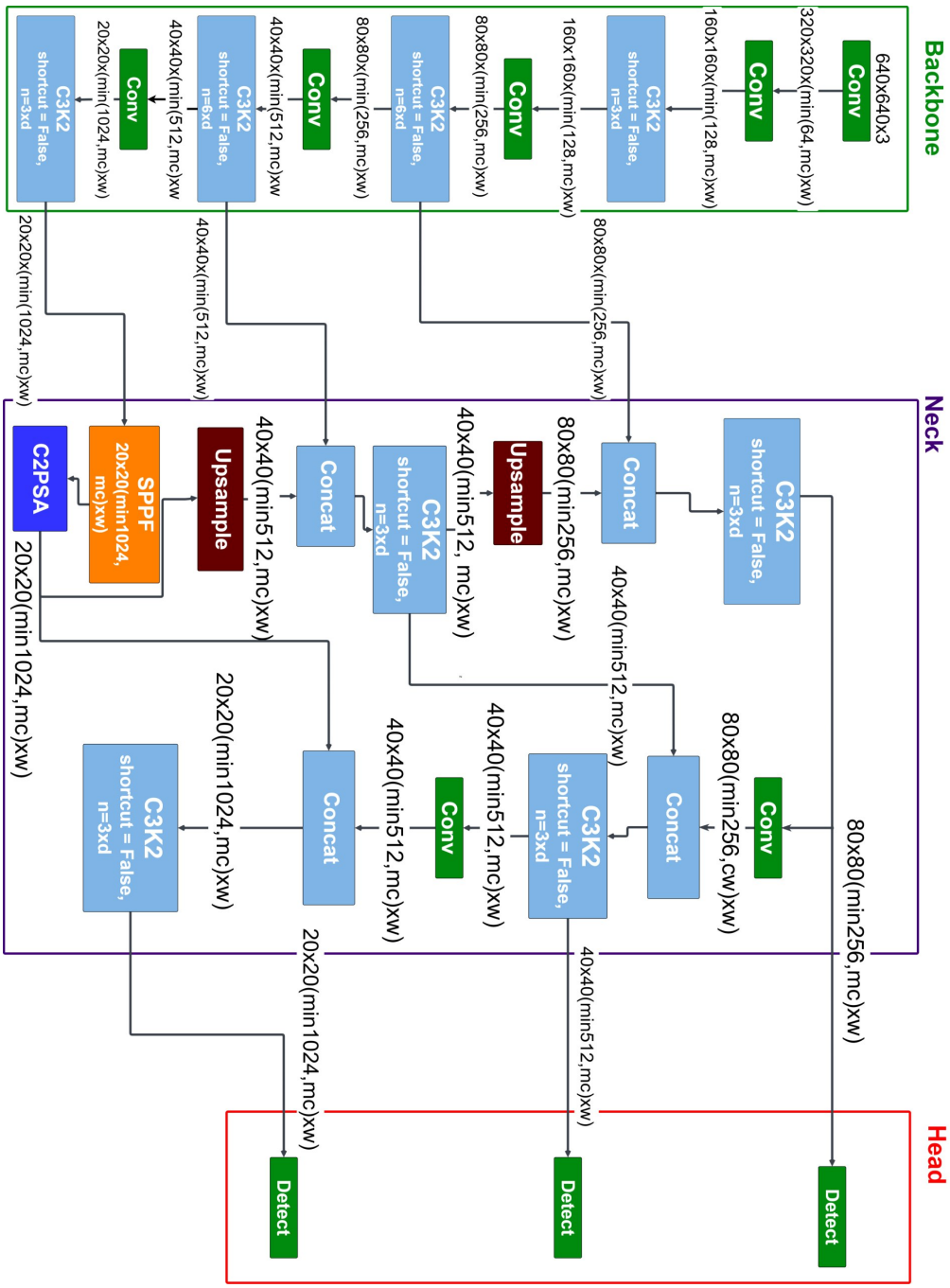


Figure 3.1: Overall YOLOv11 architecture used in this work, consisting of a DarkNet-style backbone with CSP blocks, a Spatial Pyramid Pooling – Fast (SPPF) module, and a single-stage detection head.

This is especially relevant for Varroa mite detection, where mites are small and frequently partially occluded by bee bodies or hive structures (see Fig. 3.1).

The detection head produces three feature maps: 80×80, 40×40, and 20×20, which are used to predict small, medium, and large objects, respectively. In

the presented work, the finest prediction scale at 80×80 resolution is the most relevant, because the dataset consists of FullHD images in which Varroa mites typically occupy approximately 40×40 pixels. Although the images have 1920×1080 resolutions, all input images are resized to 640×640 pixels before being passed to the network. This fixed input size ensures compatibility with the detection pipeline, keeps the feature-map dimensions consistent during training and inference, and reduces computational cost.

3.2.1 Backbone: DarkNet with Cross-Stage Partial Connections

The backbone extracts hierarchical feature representations through sequential downsampling stages. In YOLOv11, the backbone is based on a DarkNet-style design and uses Cross-Stage Partial (CSP) principles [25] to improve feature extraction efficiency and gradient flow (see Figure 3.2).

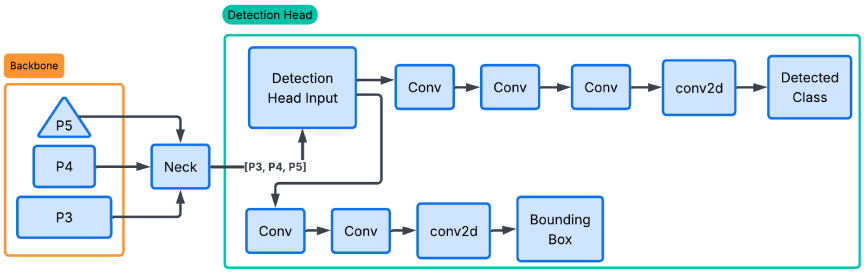


Figure 3.2: DarkNet-style backbone with CSP-based feature extraction and multi-scale pyramid stages (P3–P5).

- **Stem.** Initial convolution layers reduce the spatial resolution of the input and extract low-level visual features such as edges, textures, and contrast patterns.
- **P3 stage (8× downsampling).** The first feature pyramid stage produces feature maps at 1/8 of the input resolution. In YOLOv11, C3k2 blocks [15] are used as efficient CSP-style feature extraction modules (Figure 3.3). These blocks split feature channels into multiple paths, process part of the representation through bottleneck transformations, and then fuse the outputs, improving computational efficiency while preserving representational capacity.
- **P4 stage (16× downsampling).** The next stage produces feature maps at 1/16 of the input resolution, capturing intermediate-level structures such as local shape configurations and object parts.
- **P5 stage (32× downsampling).** The deepest backbone stage produces feature maps at 1/32 of the input resolution, encoding higher-level semantic information and broader contextual relationships.

The resulting multi-scale feature hierarchy (P3–P5) is important for Varroa detection because mites may appear at different apparent sizes depending

on camera distance, image scale, and their position on adult bees or brood cells. The finer-resolution features are more suitable for detecting small mite instances, whereas the coarser feature maps provide contextual information that can help distinguish mites from visually similar bee structures.

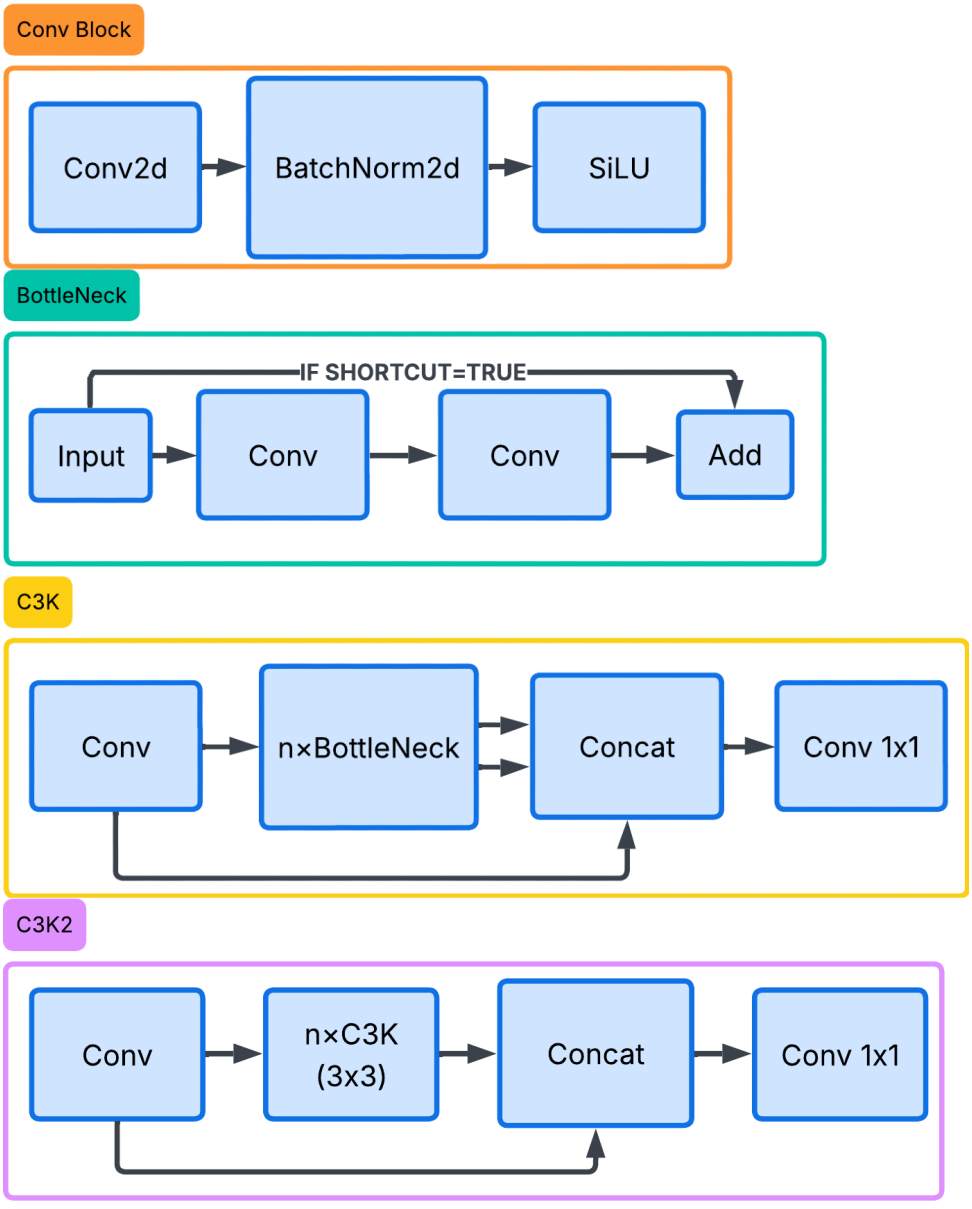


Figure 3.3: Modular building blocks used in the YOLOv11 backbone and neck, including C3k/C3k2 units for core feature extraction.

3.2.2 Spatial Pyramid Pooling and Attention

At the deepest feature level (Figure 3.1), YOLOv11 retains the Spatial Pyramid Pooling – Fast (SPPF) (Figure 3.4) module [15, 26], which applies repeated max-pooling operations to enrich deep features with broader con-

textual information. The pooled features are concatenated with the original representation, allowing each spatial location to incorporate information from a larger receptive field.

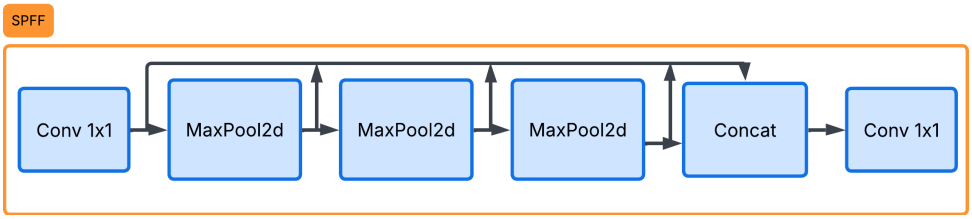


Figure 3.4: SPFF collects features from different parts of the image at multiple scales

After SPPF, YOLOv11 applies the C2PSA block [15], which introduces attention-based feature refinement to emphasize informative spatial regions. In the context of Varroa detection, the combination of SPPF and C2PSA is useful because it helps preserve local detail while also incorporating wider spatial context, which may improve detection of small or partially occluded mites.

3.2.3 Detection Head

The detection head receives the feature maps produced by the backbone and neck and converts them into grid-based detection predictions. Each grid cell predicts a fixed output vector consisting of an objectness score, bounding-box values, and class logits [14].

- **Bounding box offsets** (b_x, b_y, b_w, b_h): four values encoding the predicted center position and dimensions of the bounding box. The center coordinates b_x and b_y are represented relative to the assigned grid cell, while b_w and b_h represent the predicted width and height in grid-cell units. These values are optimized as a regression task using Mean Squared Error (MSE) loss.
- **Objectness score**: a scalar prediction indicating whether a grid cell contains the center of a Varroa mite. This is formulated as a binary classification task, where cells containing an object are assigned a target value of 1 and background cells are assigned a target value of 0. The objectness branch is trained using Binary Cross-Entropy (BCE) loss.
- **Class logits**: class predictions associated with the object assigned to a grid cell. Since the current task contains only one object class, Varroa mite, this branch performs single-class classification. It is also trained using BCE loss. In this single-class configuration, the class prediction is partly redundant with the objectness score, but it keeps the detection head compatible with possible future multi-class extensions.

The predicted objectness and class scores are used to estimate the confidence of each detection. Non-Maximum Suppression (NMS) [27] is then applied as a

post-processing step to remove duplicate detections of the same mite. When multiple overlapping boxes exceed the confidence threshold, NMS retains the highest-confidence prediction and suppresses the remaining overlapping boxes.

3.3 Training Configuration

The model was trained with the following configuration:

- **Optimizer:** AdamW [28] with an initial learning rate of 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight decay of 10^{-2}. AdamW decouples weight decay from the gradient update, providing more stable training dynamics compared to standard Adam with L2 regularization.
- **Batch size:** 4 images per batch, constrained by GPU memory at Full HD input resolution.
- **Training epochs:** 300, with validation loss monitored for convergence assessment.
- **Loss function:** A composite loss combining three terms:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{box}}\mathcal{L}_{\text{box}} + \lambda_{\text{obj}}\mathcal{L}_{\text{obj}} + \lambda_{\text{cls}}\mathcal{L}_{\text{cls}} \tag{3.1}$$

  where $\mathcal{L}_{\text{box}}$ is the Mean Squared Error (MSE) loss on bounding box coordinates for grid cells containing objects, $\mathcal{L}_{\text{obj}}$ is the Binary Cross-Entropy loss for objectness prediction across all grid cells, and $\mathcal{L}_{\text{cls}}$ is the BCE loss for class prediction in object-containing cells. The weighting coefficients $\lambda_{\text{box}} = 5.0$, $\lambda_{\text{obj}} = 1.0$, and $\lambda_{\text{cls}} = 1.0$ are fixed hyperparameters set prior to training. The higher weight assigned to λ_{box} reflects the greater importance in the detection [14]. The coefficients λ_{obj} and λ_{cls} are set to 1.0 as the unit baseline. Since only a single class is present, $\mathcal{L}_{\text{cls}}$ contributes minimally to the total objective and serves primarily to maintain compatibility with future multi-class extensions.

Data augmentation techniques [29, 30] applied during training include:

- Horizontal and vertical flips with 50% probability each, increasing the effective variety of mite positions and orientations.
- Brightness and contrast adjustments, simulating variation in NIR LED illumination intensity across different hive regions.
- Blur with a small kernel (blur_limit = 3) applied with 10% probability, providing regularization against high-frequency noise patterns in the sensor output.
- Affine transformations: rotation ($\pm 15°$), scale (0.9–1.1×), and translation up to 6.25% of the image size, increasing robustness to variations in camera-to-frame distance and gantry positioning.

Training was conducted on an NVIDIA DGX A100 Station [31] equipped with 40 GB of GPU memory per accelerator, enabling training at full input resolution without tiling or downsampling.

3.4 Temporal Continuity Investigation

The AROBA system captures continuous video at 30 FPS, meaning that the same Varroa mite typically remains visible across multiple consecutive frames as it passes through the camera field of view attached to its bee host. Without temporal association, a naive per-frame detector would count the same mite repeatedly, producing a severe overestimate of the true infestation level. Temporal continuity therefore enables accurate mite counting - by ensuring that each individual mite is counted exactly once regardless of how many frames it appears in (see Figure 3.5).

Figure 3.5: Example of BoTSORT tracking applied to a video frame from the AROBA system. The first three frames are consecutive, which is why the track identity remains the same (track #1). The last frame was taken from a different part of the video, so the mite appears as a new detection (track #2).

This thesis addresses this problem through detection-level tracking using a simplified BoT-SORT algorithm [32]. BoT-SORT, short for Bag of Tricks

Simple Online and Realtime Tracking, is a multi-object tracking framework proposed by Aharon et al. [32] that extends the classical SORT tracker with an improved Kalman filter state vector and camera-motion compensation. Its full form additionally incorporates a visual re-identification (ReID) module, which increases robustness at the cost of significant computational overhead. Since the deployment target is a lightweight web service rather than a dedicated GPU workstation, a simplified variant is used that relies solely on motion-based association.

For every pair of an existing track i and a new detection j, a combined association score is computed,

$$\text{score}(i,j) = w_{\text{iou}} \cdot \text{IoU}(i,j) + w_d \cdot \left(1 - \frac{d_{\text{centre}}(i,j)}{d_{\text{limit}}(i,j)}\right), \tag{3.2}$$

where $\text{IoU}(i,j)$ is the intersection-over-union overlap between the two boxes, $d_{\text{centre}}(i,j)$ is the Euclidean distance in pixels between their centre points, $d_{\text{limit}}(i,j)$ is an adaptive distance threshold derived from the size of the larger box, and $w_d = 0.65$, $w_{\text{iou}} = 0.35$ were chosen empirically. Candidate pairs are ranked by score and matched greedily in descending order (see Figure 3.5). Each confirmed track is assigned a unique identity, allowing the system to count the number of distinct mites observed. A track is confirmed as soon as it receives its first successful detection match, while tracks that fail to be re-matched within a fixed number of consecutive frames are discarded as likely false positives.

All images submitted within a single request are treated as a consecutive sequence and processed by a shared tracker instance. Images uploaded in separate requests are tracked independently with no shared state between them.

The tracking approach is motivated by the observation that Varroa mites, being physically attached to their bee hosts, exhibit smooth and predictable motion trajectories that mirror the movement of the underlying bee. False positive detections, by contrast, tend to be spatially and temporally inconsistent - appearing in one frame but not in adjacent frames, or shifting location discontinuously. By requiring temporal persistence before confirming a detection, the system exploits this behavioural prior to improve effective precision without sacrificing recall. The expected benefit is greatest when the single-frame confidence threshold is set conservatively low to maximise recall: track-based filtering then recovers high precision by suppressing the resulting false positives [32].

Chapter 4

Web-Based Detection and Collaborative Annotation Service

The trained model alone targets a narrow audience - biologists and researchers operating within the AROBA experimental setup. A web-based service (see Fig 4.1 this reach to beekeepers and other non-technical users who lack access to a CUDA-capable workstation. A user opens a web page, uploads a few hive images, and receives the same images with bounding boxes drawn around detected Varroa mites. Mites are also tracked, this will help for automatic counting of varroa in hive. Any obvious mistakes can be corrected by hand and saved, building a collection of reviewed annotations that can be used to improve future model versions. This chapter describes how that service is put together.

4.1 Motivation and Design Principles

Varroa detection on NIR images is a task where a human still has the upper hand in tricky cases. Mites can sit between bee setae, hide in shadows, or look almost identical to compound eyes, wing joints, or debris. A network trained on one dataset cannot capture all that variety, but the people who look at hundreds of bees a week can. The web service is the place where their corrections are collected and turned into new training data.

Four practical goals shaped the design. First, it should run in any browser, with no Python, no CUDA, no setup. Second, predictions should appear right away and the boxes should be editable directly on the picture. Third, going from a correction to a retrained model should not require manual steps. And fourth, the code should follow the usual layered structure of a small web application so that the detection, the business logic, and the database stay separated and testable on their own.

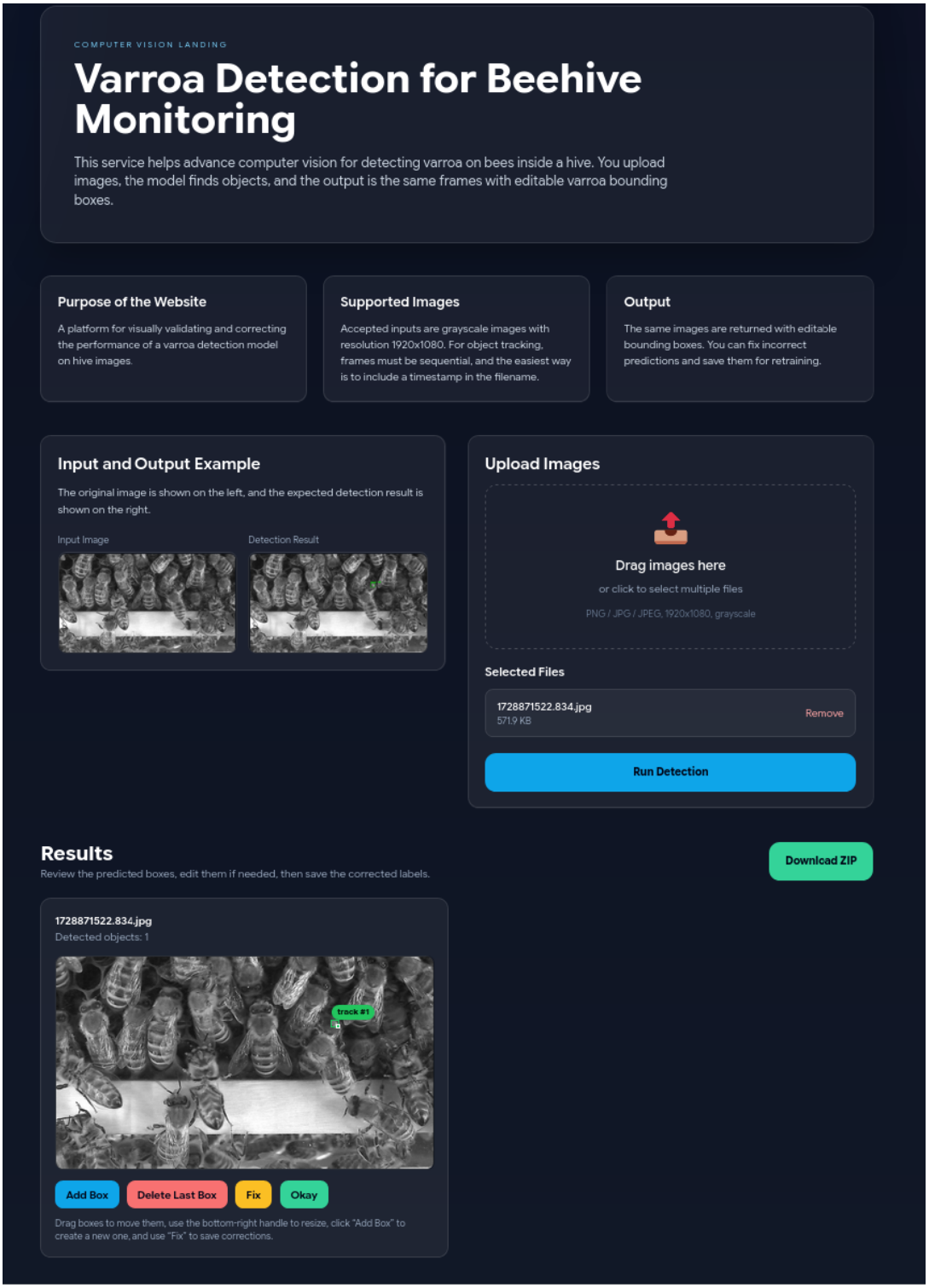


Figure 4.1: Web-based interface for automatic detection and annotation of Varroa mites

4.2 System Architecture

The backend is built on FastAPI [33] and follows a standard layered architecture [34, 35]. Each layer has one responsibility and only talks to the layer directly below it:

- **API router** (`app/api/routes.py`) declares the three HTTP endpoints and the OpenAPI metadata that Swagger UI [36, 37] renders.
- **Controllers** are thin adapters. They call a service and translate domain exceptions into HTTP status codes (`ValueError` → 400, `FileNotFoundError` → 404).
- **Services** hold the business logic: validating uploads, running inference and tracking, drawing previews, packaging the ZIP, saving corrections, and starting retraining.
- **Repositories** (the DAO layer [34, 38]) hide every database call behind small intent-revealing methods. The rest of the code never sees raw SQL.
- **Entities** (`app/models/entities.py`) are the SQLModel [39] classes that describe the database tables (`Job`, `ImageRecord`, `Annotation`, `TrainingRun`).
- **DTOs** (`app/schemas/dto.py`) are Pydantic [40] models that describe the request and response payloads. They are kept separate from the entities so that the database can change without breaking the public API.

The class diagram in Figure 4.2 shows how these pieces fit together. The diagram source is kept under `docs/class-diagram.mmd` as a Mermaid [41] file, so it stays next to the code in Git and can be regenerated whenever the structure changes.

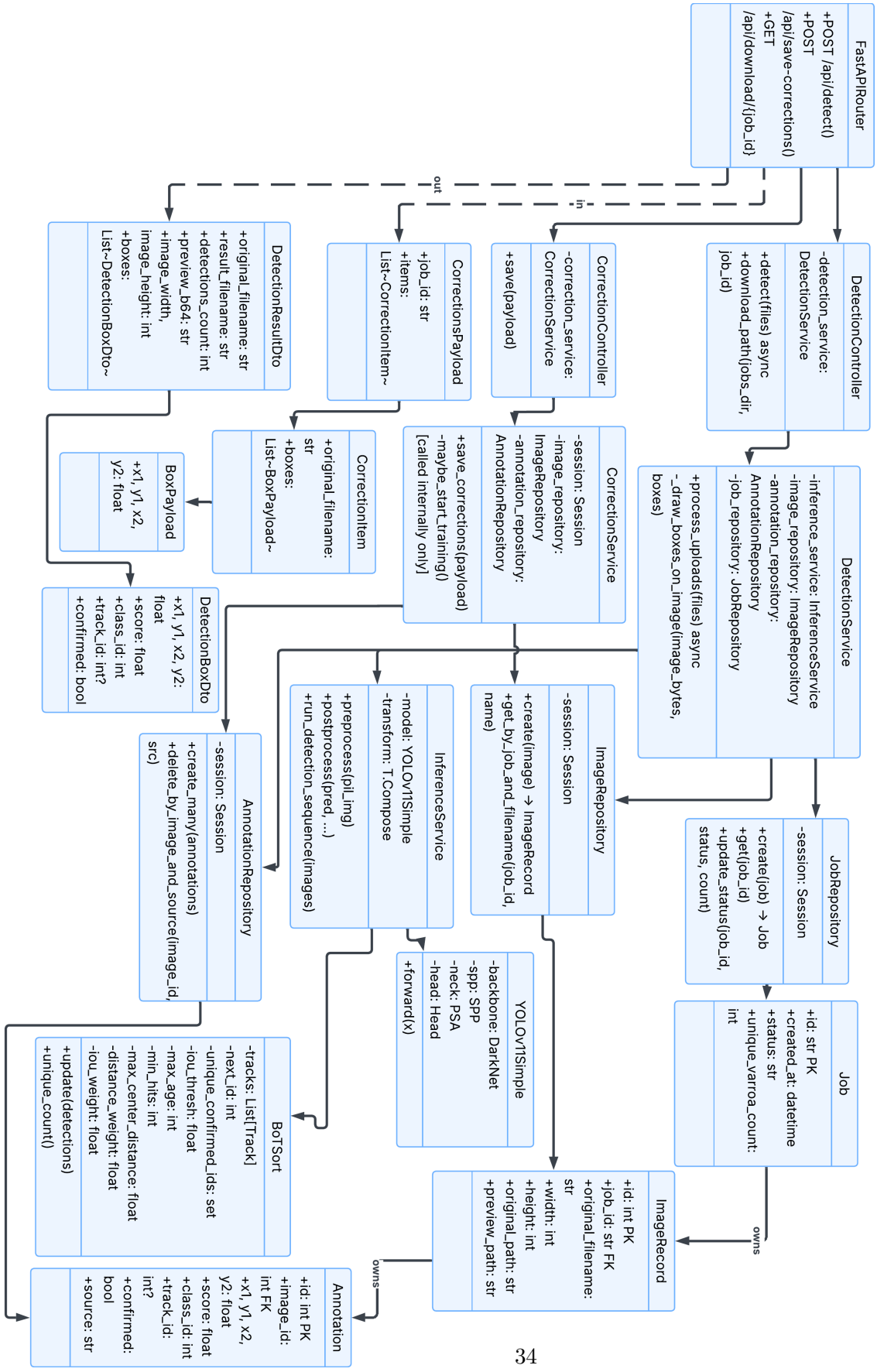


Figure 4.2: Class diagram of the web service.

The service exposes three endpoints, all documented automatically at `/docs`:

1. `POST /api/detect` - accepts one or more uploaded images, runs YOLOv11 inference, applies BoT-SORT (see Section 3.4) tracking across the batch, and returns boxes, previews, and a unique job id.
2. `POST /api/save-corrections` - accepts manually fixed boxes for a finished job, replaces the previous corrections in the database, and writes the updated YOLO label files.
3. `GET /api/download/{job_id}` - returns a ZIP archive with the annotated images and label files for one job.

4.3 Detection Pipeline

When images arrive at `/api/detect`, the detection service runs the same pipeline for each one: validate the upload, convert the image to grayscale, resize to 640×640, run a single YOLOv11 forward pass, decode the grid cells into pixel-space boxes, drop low-confidence boxes, run NMS, and rescale the survivors back to 1920×1080. The input is resized to 640×640 because it keeps inference fast, resizing actually enlarges Varroa mites, making small objects easier for the detector to localise. Also it is the established standard input resolution across the YOLO family of detectors.

The most fragile part of this pipeline turned out to be the conversion to grayscale. The trained backbone declares its first convolution as `Conv(1, …)`, so the input must have exactly one channel; an RGB tensor crashes the very first layer. The conversion is therefore explicit in `InferenceService.preprocess` and pinned by a regression test:

```
def preprocess(self, pil_img):
    if pil_img.mode != "L":
        pil_img = pil_img.convert("L")
    tensor = self.transform(pil_img)        # Resize 640x640, ToTensor
    return tensor.unsqueeze(0), pil_img.size
```

Listing 4.1: Single-channel preprocessing in `InferenceService`.

Once detection is finished, BoT-SORT is fed the boxes from each frame to associate them across the batch and report a deduplicated count of unique mites; this is described in the next section.

4.4 Interactive Annotation Correction

The frontend draws every predicted box as an HTML element on top of the returned preview. The user can drag a box to reposition it, drag its corners to resize it, click an empty area to add a missing one, or press a key to delete a false positive (see Figure 4.3). When the user is done, the corrected boxes are

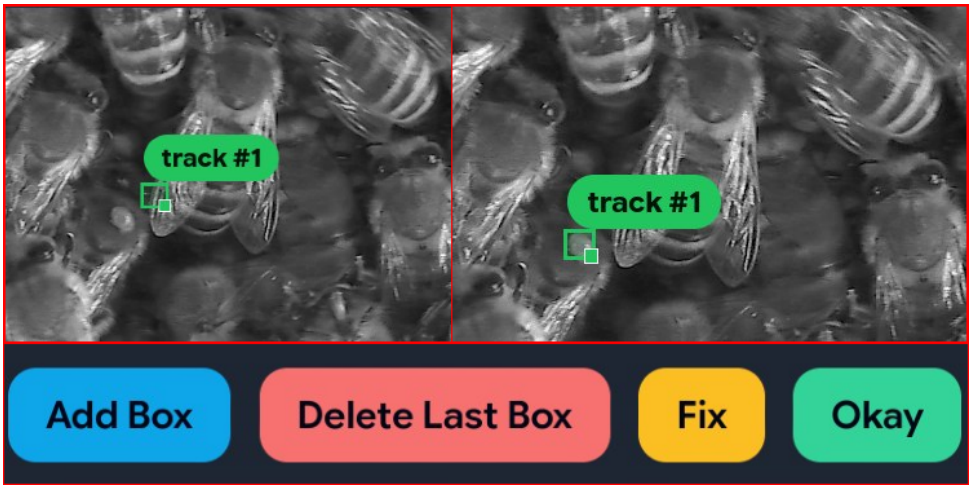


Figure 4.3: The annotation correction interface after repositioning a predicted box. The user can also add missing boxes or delete false positives directly on the preview. Pressing the *Fix* button saves the corrected boxes alongside the original image to a dedicated folder for use in future model retraining.

sent to `/api/save-corrections` as JSON. On the server side, the correction service does four things in order (see Figure 4.2). It clamps every box to the image bounds and drops anything smaller than one pixel. It converts pixel coordinates to the normalised YOLO format, where each bounding box is represented by five values (`class_id cx cy w h`). Here, `class_id` is the object class index; `cx` and `cy` are the box centre coordinates; and `w` and `h` are its width and height, with all spatial values scaled to $[0, 1]$ relative to the image dimensions. It then writes a fresh `.txt` label file. It copies the original image to the corrections directory next to the new label. Finally, atomically per image, it deletes any previous `source = "correction"` rows from the `Annotation` table and inserts the new ones. That last step is what makes the operation idempotent: the same image can be corrected repeatedly without the predictions or older corrections piling up.

4.5 Automatic Retraining Trigger

After every successful correction, the service counts how many label files are currently sitting in the corrections directory. If the count crosses the configured threshold (`TRAIN_TRIGGER_COUNT`, default 500) and no `train.lock` file is present, the service writes the lock file and starts `train.py` as a detached subprocess; the HTTP response returns immediately. When training finishes, the lock file is removed and the new `model.pt` is picked up by the next `InferenceService` that loads it.

```
if len(label_files) < TRAIN_TRIGGER_COUNT \
   or train_lock.exists() \
   or not TRAIN_SCRIPT.exists():
    return False, len(label_files)
```

```
5
6  train_lock.write_text("running", encoding="utf-8")
7  subprocess.Popen([
8      "python3", "-u", str(TRAIN_SCRIPT),
9      "--corrections_images", str(CORRECTIONS_IMAGES_DIR),
10     "--corrections_labels", str(CORRECTIONS_LABELS_DIR),
11     "--weights",            str(MODEL_PATH),
12     "--epochs", "50", "--lr", "1e-4",
13     "--lock_file", str(train_lock),
14 ])
```

Listing 4.2: Retraining is fired only when the threshold is crossed and no run is active.

The lock file prevents two retraining runs from starting at the same time, and detaching the subprocess prevents the HTTP request from blocking on a multi-hour training job. Together these two details turn the server into a closed feedback loop: predictions → corrections → retraining → better predictions.

4.6 Technology Stack

The backend is intentionally built from a small set of well-known Python tools so that it stays easy to read and to deploy.

- **Python 3.10+** as the runtime, shared with the offline training script.
- **FastAPI** [33] as the web framework. Its type-annotated route signatures double as runtime validation and as the source of the OpenAPI document.
- **Uvicorn** [42] as the ASGI server. The `--workers N` flag forks N independent processes that each run a copy of the app. How many workers make sense on a given machine is the topic of Section 4.8.
- **Swagger UI / OpenAPI** [36, 37] for live, browser-based API documentation.
- **Pydantic** [40] for DTO validation. Malformed JSON is rejected with HTTP 422 before any business code runs.
- **SQLAlchemy** and **SQLModel** [43, 39] as the ORM.
- **PostgreSQL** [44] as the supported backend database.
- **PyTorch** [23] for the model, **torchvision** for NMS, **OpenCV** [45] for drawing boxes, **Pillow** for image I/O, and **NumPy** for the array work inside the tracker.
- **Locust** [46] for the load test of Section 4.10.
- **Docker** [47] to run PostgreSQL from the `postgres:12.3-alpine` image and to package the app for deployment.

- **Pytest** [48] for the layered test suite of Section 4.9.

`app/core/config.py` centralises all detection and tracking parameters: the confidence threshold (0.25), NMS IoU threshold (0.45), maximum detections per image (300), and tracking weights ($w_d = 0.65$, $w_{\text{iou}} = 0.35$, iou_thresh = 0.05, max_age = 8, min_hits = 1). Any of these can be tuned without modifying the rest of the application logic.

4.7 Running the Service Locally

The backend is a plain Python project with no compile step. It requires a running PostgreSQL instance, which can be started via Docker:

```
python3 -m venv venv && source venv/bin/activate
pip install -r requirements.txt

# Start PostgreSQL via Docker
docker run --rm -d --name varroa-pg \
    -e POSTGRES_HOST_AUTH_METHOD=trust \
    -e POSTGRES_USER=root -e POSTGRES_DB=varroa \
    -p 5432:5432 postgres:12.3-alpine

uvicorn main:app --host 0.0.0.0 --port 8000 --workers 1
xdg-open http://localhost:8000/docs
```

Listing 4.3: Local launch procedure for the backend.

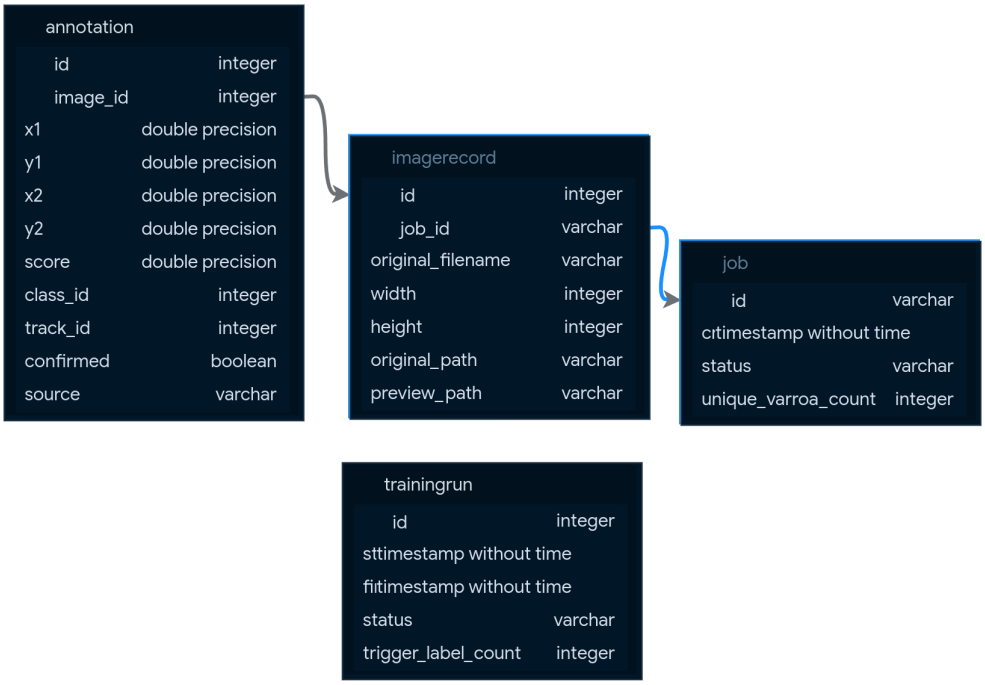


Figure 4.4: PostgreSQL database schema of the backend service.

PostgreSQL stores all metadata across four tables - `job`, `imagerecord`, `annotation`, and `trainingrun` - whose structure is shown in Figure 4.4. Beyond simply persisting paths and coordinates, the database enables meaningful operational queries: the `job` table records the unique varroa count per job, making it possible to track infestation levels over time and build hive statistics; the job and image statuses allow the system to detect and report failed or incomplete detections; and the `trainingrun` table provides a full history of how many retraining cycles have been triggered and whether they completed successfully. The image files themselves and the label files are stored directly on disk; the database holds only the paths and metadata. PostgreSQL was chosen for its native support for concurrent connections - since each Uvicorn worker runs as an independent process, multiple workers can read and write simultaneously without blocking each other, which would be slow with a SQLite [49].

4.8 Process Workers

A Uvicorn worker is a separate operating-system process that runs its own copy of the FastAPI app, its own Python interpreter, its own SQLAlchemy pool, and its own loaded YOLOv11 weights. Workers do not share Python objects, so the parallelism is real multi-core parallelism that bypasses the GIL (Global Interpreter Lock - CPython's mechanism that prevents more than one thread from executing Python bytecode simultaneously).

The realistic upper bound on the number of workers is the minimum of three limits:

1. **CPU cores.** Most of `/api/detect` outside the forward pass is plain CPU work (image decoding, drawing, ZIP). For CPU-bound services one worker per physical core is a safe.

2. **Memory.** Each worker loads the model into RAM, so the count is bounded by ⌊available RAM/worker RSS⌋. (Resident Set Size (RSS) - mount of physical RAM that a process is occupying).

3. **GPU contention.** On a single GPU, extra workers do not raise inference throughput - they only smooth out the CPU work around inference.

In practice four to six workers saturate a typical workstation without overcommitting any of these resources. The load test in Section 4.10 confirms this on the development laptop.

4.9 Automated Test Suite

The backend ships with a Pytest suite under `tests/` that exercises every layer in isolation and runs in under five seconds without GPU, model weights, or network access. This is achieved by stubbing `torchvision`, `model`, and

`bot_sort` with in-memory fakes inside `conftest.py` and by using an in-memory SQLite engine for every database test.

- `test_repositories.py` - the DAO methods, including the source-scoped delete that makes saving corrections idempotent.
- `test_services.py` - detection and correction services against a fake inference backend, including the wrong-size-image and degenerate-box paths.
- `test_controllers.py` - exception-to-status-code translation in the controllers.
- `test_api.py` - end-to-end through FastAPI's `TestClient`, with the session and inference dependencies overridden.
- `test_schemas.py` - DTO validation for missing fields and lenient numeric strings.
- `test_inference_preprocess.py` - a focused regression test that pins the grayscale-conversion bug observed during the very first Locust run.
- `locustfile.py` and `run_load_test.sh` - the load profile and shell wrapper used in Section 4.10.

4.10 Load testing and scalability analysis

4.10.1 Test methodology

A staged Locust profile drives `POST /api/detect` with a staircase of virtual users, each step held for sixty seconds (1, 10, 100, 200, 400, and 800 concurrent users). The same profile was replayed three times against a local deployment of the backend (`uvicorn main:app --workers N`) backed by PostgreSQL, with $N \in \{1, 6, 12\}$.

The runs reported in Tables 4.1–4.3 were executed on the *development laptop*: AMD Ryzen 7 8845HS (8 cores / 16 threads) with Radeon 780M integrated graphics, an NVIDIA GeForce RTX 3050 Laptop GPU with 6 GiB of VRAM, and 13.4 GiB of usable RAM plus 16 GiB of swap. Each virtual user repeatedly posts an image to `/api/detect` and waits for the annotated response, whose body averages roughly 4 MB. The numbers below should therefore be read as a worst-case profile on commodity hardware; behaviour on a proper server is reported separately in Section 4.10.4.

4.10.2 Aggregate results on the development laptop

Tables 4.1–4.3 summarise the three runs. The values are taken directly from the Locust HTML reports: *Request Statistics* (one row per run), *Response Time Statistics* (per-request latency percentiles), and *Failure Statistics*.

Table 4.1: Aggregate Locust *Request Statistics* for `POST /api/detect` on the development laptop, one row per worker count. *# Req.* the total number of completed requests; *# Fails* the number of failed requests. *Avg.*, *Min*, and *Max* are the average, minimum, and maximum response times in milliseconds. *Avg. size* is the average response payload size in bytes. *RPS* (requests per second) reports the achieved throughput, and *Fail/s* is the average number of failed requests per second.

Workers	# Req.	# Fails	Avg.	Min	Max	Avg. size	RPS	Fail/s
1	1,407	1	19,134.44	233	177,961	4,109,141	3.90	0.00
6	5,748	8	11,191.50	222	80,874	4,106,340	15.90	0.02
12	7,161	105	17,545.76	221	163,198	4,051,769	15.52	0.23

Table 4.2: Locust *Response Time Statistics* on the development laptop: end-to-end latency percentiles, in milliseconds.

Workers	p_{50}	p_{60}	p_{70}	p_{80}	p_{90}	p_{95}	p_{99}	p_{100}
1	16,000	20,000	22,000	24,000	26,000	47,000	152,000	178,000
6	8,900	10,000	13,000	17,000	25,000	27,000	58,000	81,000
12	7,800	10,000	14,000	17,000	35,000	98,000	157,000	160,000

Table 4.3: Locust *Failure Statistics* on the development laptop, grouped by error class. The single- and six-worker runs only see client-side `ConnectionResets`; with twelve workers the server starts to return truncated responses (`ChunkedEncodingError` / `IncompleteRead`) and aborts mid-stream (`RemoteDisconnected`).

Workers	# Failures	Failure classes (counts)
1	1	`ConnectionResetError` (1)
6	8	`ConnectionResetError` (8)
12	105	`ChunkedEncodingError / IncompleteRead` (33), `RemoteDisconnected` (36), `ConnectionResetError` (36)

4.10.3 Discussion of the laptop runs

From one to six workers - almost-linear speed-up. Adding workers up to six lifts sustained throughput from 3.90 to 15.90 req/s - roughly a 4.1× gain - and pulls every latency percentile down at the same time (p_{50}: 16 s → 8.9 s; p_{95}: 47 s → 27 s; p_{99}: 152 s → 58 s). At this point neither the GPU nor the CPU is fully busy, so extra workers can run in parallel without fighting for any one resource. Failures stay at a handful of stray client-side `ConnectionResets`.

From six to twelve workers - throughput plateau and quality collapse. The twelve-worker run does not increase overall throughput at all: it reaches 15.52 req/s, which is essentially the same as 15.90 req/s, and it makes the latency tail dramatically worse: p_{95} jumps from 27 s to 98 s and the failure count explodes from 8 to 105. The character

of the failures also changes - previously they were just clients giving up (`ConnectionResetError`); now the server itself is delivering truncated responses (`ChunkedEncodingError` / `IncompleteRead`) and tearing connections down mid-stream (`RemoteDisconnected`).

Worker count is bounded by the number of physical cores. The number of physical cores is the primary ceiling for worker count, but several additional factors must be considered when tuning for production:

1. **CPU load.** Most of `/api/detect` outside the forward pass is CPU-bound work - image decoding, drawing boxes and labels, and ZIP packaging. These are all computationally heavy operations, with ZIP compression being particularly expensive (see Table 4.5), which guarantees that a single core under high request rates will be driven to full utilisation. Once the number of workers exceeds the number of physical cores, processes compete for the same execution units, context switching grows, and every individual request stretches longer. This is also confirmed empirically in Table 4.1, where increasing the worker count beyond the core count yields no significant growth in throughput.

2. **Memory pressure.** Each worker maintains its own Python interpreter, model weights, the FastAPI/SQLAlchemy stack, and an output buffer. The exact resident set depends on model size, image resolution, batch size, and OS-level shared mapping behaviour. The practical consequence is that even on a machine with a capable CPU, insufficient RAM will exhaust physical memory and, if swap space is configured, force the OS to begin paging memory to disk - page faults hitting swap are several orders of magnitude slower than RAM accesses. On systems without swap, the OS will simply begin terminating processes instead.

3. **GPU serialisation.** All workers share the same physical device through separate CUDA contexts. The GPU does not parallelise forward passes across workers - it queues them and executes one at a time. Under high traffic this becomes a significant bottleneck: every incoming request waits in line regardless of how many workers are active. Adding more workers past a certain point only lengthens the queue, it does not increase throughput. Each worker occupies approximately 354 MiB of VRAM under load, and while the 6 144 MiB capacity is not exhausted, the single SM scheduler remains the hard ceiling on inference throughput.

4. **Disk I/O.** Disk write latency itself is not a limiting factor, as confirmed by the low `decode` times in Table 4.5, which include saving originals to disk.

5. **Database overhead.** As shown in Table 4.5, the database phase consistently completes in tens of milliseconds regardless of batch size. The hot path consists of a small number of `INSERT` statements and a single `UPDATE` per request; with response payloads in the megabyte range

and end-to-end latency in the seconds, the database contribution to total latency is negligible.

Table 4.4: Timing phases recorded by the `perf` logger.

Phase	Description
`decode`	Reading uploaded file bytes, decoding, validating dimensions, and saving originals to disk.
`inf`	Full YOLO inference pipeline: pre-processing, GPU forward pass, detection post-processing (NMS, box decoding), and BoT-SORT tracking.
`zip`	Drawing bounding boxes and labels onto result images, writing YOLO label `.txt` files, and packaging everything into a `ZIP` archive.
`db`	Creating `Job`, `ImageRecord`, and `Annotation` rows in PostgreSQL.

Table 4.5: Per-request timing breakdown recorded by the `perf` logger on the local machine.

Images	`decode` (s)	`inf` (s)	`zip` (s)	`db` (s)	`total` (s)
1	0.008	0.075	0.107	0.018	0.208
12	0.063	0.888	1.250	0.021	2.222

4.10.4 Validation on a remote server

The load tests conducted on the laptop provided an initial proof of concept, but its limited CPU core count and GPU VRAM made it difficult to analyse scaling behaviour and isolate true architectural bottlenecks. To obtain more representative results, the same staged Locust profile was replayed on a remote server fitted with an AMD Ryzen 9 3950X (16 cores / 32 threads), an NVIDIA GeForce RTX 3080 Ti (12 GiB of VRAM), 125 GiB of system RAM, and 63 GiB of swap space, running the same backend code and PostgreSQL configuration. Three worker counts were evaluated: 1, 8, and 16. Each run lasted 6 minutes with the same ramp-up profile reaching 800 concurrent users at the plateau. (see Tables 4.6 and 4.7).

Table 4.6: Locust *Request Statistics* for `POST /api/detect` on the remote server, one row per worker count. Latencies are in milliseconds; *RPS* is requests per second; *Fail/s* is the average number of failed requests per second.

Workers	# Req.	# Fails	Avg.	Min	Max	Avg. size (B)	RPS	Fail/s
1	3,160	0	13,251	92	159,148	14,621	8.79	0.00
8	16,842	0	4,729	98	58,706	14,621	46.73	0.00
16	19,684	10	4,337	95	39,325	14,614	54.58	0.03

Table 4.7: Locust *Response Time Statistics* on the remote server: end-to-end latency percentiles in milliseconds.

Workers	p_{50}	p_{60}	p_{70}	p_{80}	p_{90}	p_{95}	p_{99}	p_{100}
1	11,000	12,000	15,000	16,000	19,000	25,000	118,000	159,000
8	3,400	5,600	5,900	6,600	9,000	9,700	26,000	59,000
16	3,200	4,700	5,300	6,600	9,600	10,000	19,000	39,000

The jump from 1 to 8 workers delivers a substantial improvement across all metrics. Scaling further to 16 workers yields only a marginal 16.8% RPS gain and slightly better latency. It is not entirely obvious what exactly is a bottleneck, so further actions have been taken to identify it.

Bottleneck analysis. To identify the true limiting resource, RAM, GPU, network, and CPU were each inspected in turn during an active 16-worker load test.

RAM was ruled out immediately: of the 125 GiB available, only ∼15 GiB was in use and swap remained untouched throughout the run.

GPU utilisation was measured with `nvidia-smi dmon -s u` (Figure 4.5). The GPU was therefore idle roughly 70% of the time, not because of any hardware limit, but because it was simply not being fed work fast enough.

TCP connections were checked with `ss -s` (Figure 4.6). At peak load: 861 established connections, zero orphaned, zero in `timewait`, meaning no network issues appeared.

CPU utilisation, monitored via `htop` (Figure 4.6), showed all 32 logical threads at 85–97% with a one-minute load average of 48.82 - three times the 16-core capacity of the chip. This confirmed CPU as the primary bottleneck.

Flame graph profiling with `py-spy` [50] identified the exact source of CPU pressure (Figure 4.7). The `process_uploads` call tree breaks into three visible blocks by width, which is proportional to CPU time. The largest block is the inference pipeline (`run_detection_sequence`), which contains the GPU forward pass, BoT-SORT tracking, and NMS post-processing, followed by `_draw_boxes_on_image` (bounding box and label rendering with OpenCV), and ZIP compression. Notably, a significant share of CPU time is consumed by purely CPU-bound operations, confirming that CPU-side is the primary bottleneck under high concurrency.

# gpu	sm	mem	enc	dec	jpg	ofa
# Idx	%	%	%	%	%	%
0	26	13	0	0	0	0
0	25	12	0	0	0	0
0	28	13	0	0	0	0
0	26	12	0	0	0	0
0	20	9	0	0	0	0
0	27	13	0	0	0	0
0	23	11	0	0	0	0
0	25	12	0	0	0	0
0	32	15	0	0	0	0
0	27	13	0	0	0	0
0	28	13	0	0	0	0
0	28	13	0	0	0	0
0	25	12	0	0	0	0
0	26	13	0	0	0	0
0	23	11	0	0	0	0
0	30	14	0	0	0	0

Figure 4.5: GPU utilisation sampled with `nvidia-smi dmon` during the 16-worker load test. Each row is one one-second sample for GPU 0. **sm** (streaming multiprocessor) - shader core utilisation (%); **mem** (memory controller) - memory bus utilisation (%); The primary metric of interest is **sm** (streaming multiprocessor utilisation), which directly reflects how heavily the GPU is engaged during YOLO inference. Values in the 20–32% range indicate moderate but stable utilisation, confirming that the GPU is not the bottleneck under the current load.

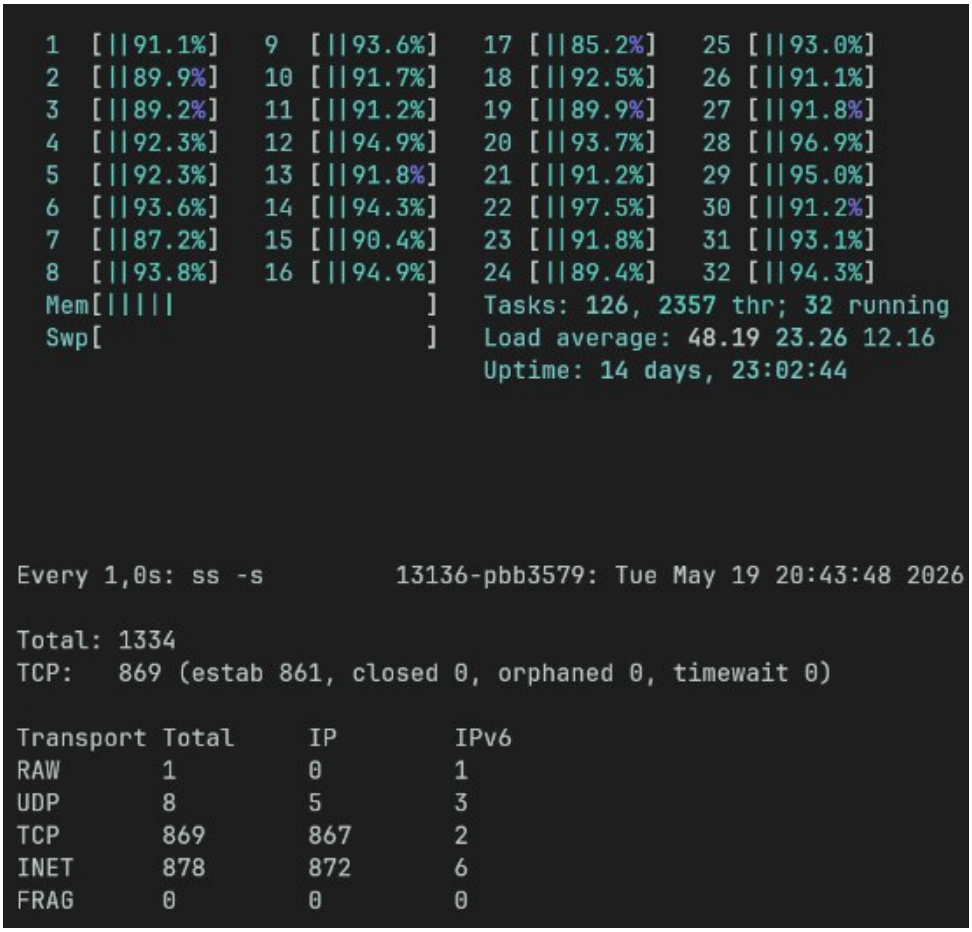


Figure 4.6: System resource utilisation. `htop` shows per-core CPU utilisation and memory consumption across all 32 threads; `ss` (bottom) reports the number of active TCP connections established by the Locust clients.

This is also consistent with the `perf` logger results in Table 4.5, where `zip` time equals or exceeds `inf` time even in an unloaded single-request scenario. Under concurrent load with 16 workers all compressing simultaneously, the effect compounds and saturates all available cores. The GPU stalls at $\sim$25% `sm%` not due to any GPU-side constraint, but because CPU post-processing starves it of work. Pushing beyond 16 workers would drive cores to 100% and increase `ConnectionResetError` failures further, as the OS would begin stalling `write()` calls on response sockets. The 8-worker configuration avoids this regime entirely, which is why it delivers better tail latencies and zero failures despite slightly lower peak RPS.

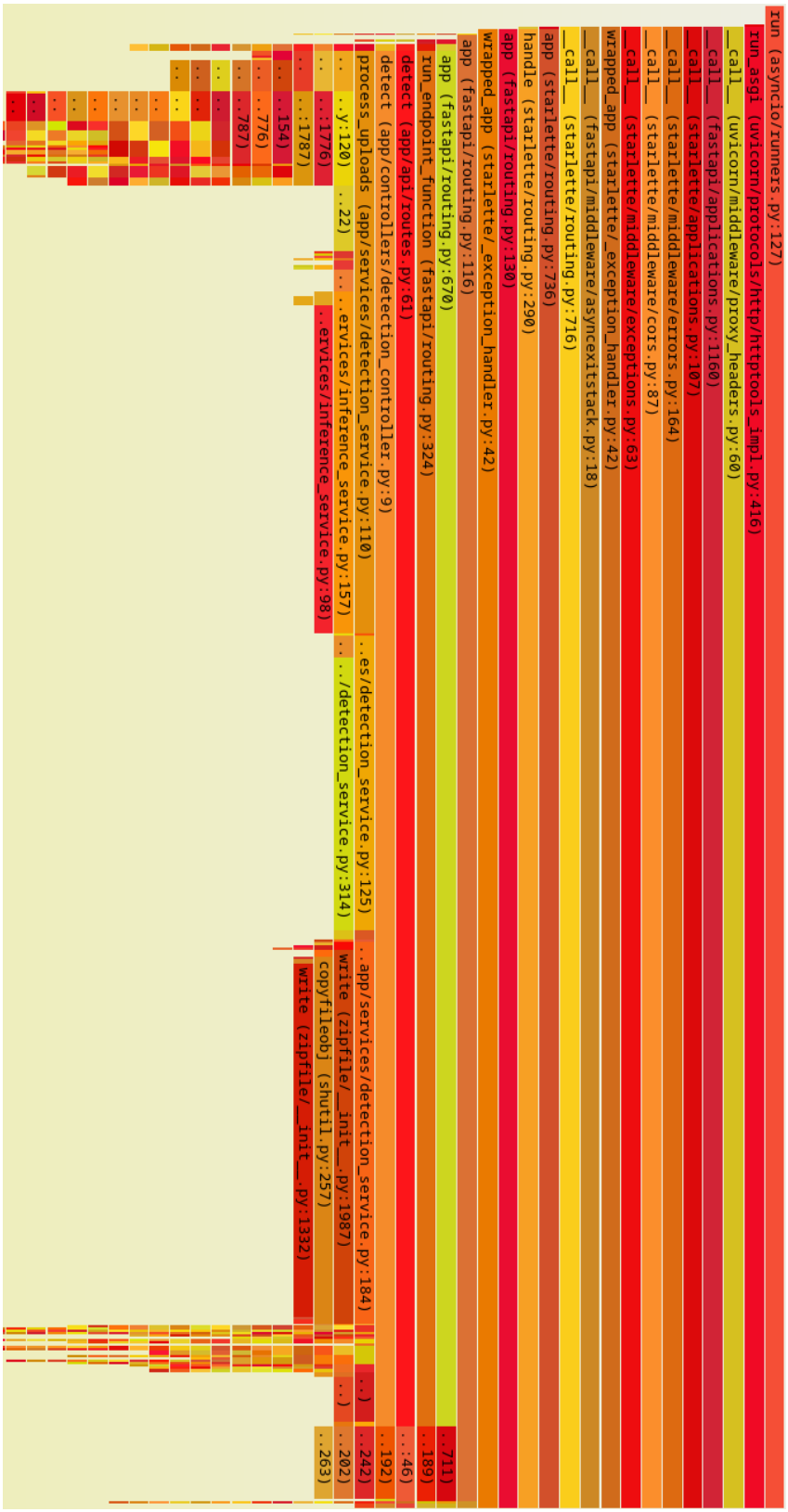


Figure 4.7: Call stack profile captured with `py-spy` during the 16-worker load test. Each row represents a Python function sampled at regular intervals, with the percentage indicating the proportion of time the process spent executing that function.

4.11 AROBA Integration via Web-Based Interface

Several possible ways of connecting the detector to the AROBA robotic beehive platform were considered: either running the pipeline directly on the AROBA hardware, or delegating inference to a web-based server. In either case, integration implies the ability to detect Varroa mites in real time. A server-based deployment is the more convenient option for two reasons. First, it simplifies integration - server can be updated, retrained, and maintained independently of the AROBA hardware. Second, a dedicated server can be equipped with a substantially more powerful GPU than the embedded compute available on the robotic platform. Since AROBA does not consume the full output of the web service - it does not need the ZIP archive, the rendered preview images, the BoT-SORT tracking, the database records, or any of the beekeeper-facing extras described in Section 4.3 - the integration narrows down to a single question: how fast can the pipeline produce raw bounding boxes?

4.11.1 Detection benchmark

The AROBA camera system captures frames at 30 images per second. To support direct integration, the detector has to keep up with at least that rate on a per-frame basis. To check whether this is the case, a micro-benchmark was run in which a single frame was sent to the running backend, and the timing was measured with the `perf` logger introduced in Section 4.10. The relevant log lines for this run are reproduced verbatim in Listing 4.8.

The inference service reports that the detector initialized sequence tracking for one source image, preprocessed that image, executed model inference on a `cuda` device, and finalized tracking with one unique target tracked. The total post-processing runtime for this single frame was measured as 0.000849 seconds. This value covers the detection pipeline alone - the YOLOv11 forward pass and the NMS / box-decoding stage that produces the final bounding boxes.

The corresponding `perf` timing line records the surrounding web service overhead as `decode` = 0.010 s, `inf` = 0.437 s, `zip` = 0.159 s, `db` = 0.020 s, and an end-to-end response `total` = 0.647 s. These phases account for the parts of the web service that AROBA does not require: reading the upload, rendering preview images, packaging the ZIP archive, and writing the database rows. The detection-only component remains well below the camera frame interval even for a single frame, confirming that the core detector is fast enough for real-time use when operated on individual frames.

4.11.2 Throughput and consequences for AROBA

At 30 frames per second, AROBA provides a per-frame time budget of approximately $1/30 \approx 0.0333$ seconds. The single-frame benchmark shows

Table 4.8: Timing results for a single NIR frame on a CUDA device. Only the Detection (post-processing) row is relevant to the AROBA integration; the remaining phases describe the surrounding web service overhead and are listed for reference only.

Detection (post-processing) [s]	**0.000849**
For reference — not part of detection	
`decode` [s]	0.010
`inf` (full `perf` stage) [s]	0.437
`zip` [s]	0.159
`db` [s]	0.020
End-to-end response `total` [s]	0.647

that the detection pipeline uses only 0.000849 seconds of this budget for the post-processing step, leaving over 0.032 seconds for communication and any remaining integration overhead per frame. To quantify the communication overhead, ICMP round-trip times (RTT, i.e. the time for a packet to travel from AROBA to the server and back) were measured for both local and remote configurations; the corresponding ping traces are shown in Figure 4.8 (local) and Figure 4.9 (remote). On the loopback interface, the measured RTT values were on the order of 0.03–0.06 ms, while for the remote server they varied between approximately 7 ms and more than 160 ms, with an average of about 45 ms and occasional packet loss. These measurements indicate that, in a local deployment the network round-trip time remains well below the available ≈ 0.0333 second per-frame budget when combined with the 0.000849 second detection time, so real-time detection is feasible. In contrast, when the detector is hosted on a remote server reached over the public network, the higher and more variable RTT (with spikes into the hundreds of milliseconds) can consume a large fraction of, or even exceed, the per-frame budget, which makes strict real-time integration difficult. For this reason, a local or edge deployment is recommended for AROBA, while using a remote server is not advisable for real-time operation.

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.052 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.050 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.053 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.059 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.049 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.059 ms
64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.053 ms
64 bytes from 127.0.0.1: icmp_seq=9 ttl=64 time=0.038 ms
64 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=0.055 ms
64 bytes from 127.0.0.1: icmp_seq=11 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=12 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=13 ttl=64 time=0.058 ms
64 bytes from 127.0.0.1: icmp_seq=14 ttl=64 time=0.040 ms
64 bytes from 127.0.0.1: icmp_seq=15 ttl=64 time=0.059 ms
64 bytes from 127.0.0.1: icmp_seq=16 ttl=64 time=0.062 ms
64 bytes from 127.0.0.1: icmp_seq=17 ttl=64 time=0.037 ms
64 bytes from 127.0.0.1: icmp_seq=18 ttl=64 time=0.060 ms
64 bytes from 127.0.0.1: icmp_seq=19 ttl=64 time=0.075 ms
64 bytes from 127.0.0.1: icmp_seq=20 ttl=64 time=0.058 ms
64 bytes from 127.0.0.1: icmp_seq=21 ttl=64 time=0.028 ms
64 bytes from 127.0.0.1: icmp_seq=22 ttl=64 time=0.061 ms
64 bytes from 127.0.0.1: icmp_seq=23 ttl=64 time=0.059 ms
64 bytes from 127.0.0.1: icmp_seq=24 ttl=64 time=0.061 ms
^C
--- 127.0.0.1 ping statistics ---
24 packets transmitted, 24 received, 0% packet loss, time 23569ms
rtt min/avg/max/mdev = 0.028/0.053/0.075/0.009 ms
```

Figure 4.8: ICMP round-trip time (RTT) measurement on the local loopback interface. The RTT values are on the order of 0.03–0.06 ms.

```
PING 147.32.83.173 (147.32.83.173) 56(84) bytes of data.
64 bytes from 147.32.83.173: icmp_seq=1 ttl=62 time=53.6 ms
64 bytes from 147.32.83.173: icmp_seq=2 ttl=62 time=32.8 ms
64 bytes from 147.32.83.173: icmp_seq=3 ttl=62 time=31.9 ms
64 bytes from 147.32.83.173: icmp_seq=5 ttl=62 time=33.3 ms
64 bytes from 147.32.83.173: icmp_seq=6 ttl=62 time=24.4 ms
64 bytes from 147.32.83.173: icmp_seq=7 ttl=62 time=28.9 ms
64 bytes from 147.32.83.173: icmp_seq=8 ttl=62 time=38.7 ms
64 bytes from 147.32.83.173: icmp_seq=9 ttl=62 time=40.6 ms
64 bytes from 147.32.83.173: icmp_seq=10 ttl=62 time=30.8 ms
64 bytes from 147.32.83.173: icmp_seq=11 ttl=62 time=7.42 ms
64 bytes from 147.32.83.173: icmp_seq=12 ttl=62 time=15.4 ms
64 bytes from 147.32.83.173: icmp_seq=13 ttl=62 time=42.6 ms
64 bytes from 147.32.83.173: icmp_seq=14 ttl=62 time=164 ms
64 bytes from 147.32.83.173: icmp_seq=15 ttl=62 time=31.3 ms
64 bytes from 147.32.83.173: icmp_seq=16 ttl=62 time=14.7 ms
64 bytes from 147.32.83.173: icmp_seq=17 ttl=62 time=21.4 ms
64 bytes from 147.32.83.173: icmp_seq=18 ttl=62 time=41.3 ms
64 bytes from 147.32.83.173: icmp_seq=19 ttl=62 time=43.5 ms
64 bytes from 147.32.83.173: icmp_seq=20 ttl=62 time=115 ms
64 bytes from 147.32.83.173: icmp_seq=21 ttl=62 time=33.7 ms
64 bytes from 147.32.83.173: icmp_seq=22 ttl=62 time=83.7 ms
64 bytes from 147.32.83.173: icmp_seq=23 ttl=62 time=55.2 ms
64 bytes from 147.32.83.173: icmp_seq=24 ttl=62 time=49.5 ms
^C
--- 147.32.83.173 ping statistics ---
24 packets transmitted, 23 received, 4.16667% packet loss, time 23034ms
rtt min/avg/max/mdev = 7.424/44.929/164.111/33.762 ms
```

Figure 4.9: ICMP round-trip time (RTT) measurement to the remote server over the public network. The RTT values vary between approximately 7 ms and more than 160 ms, with an average of about 45 ms and occasional packet loss.

Chapter 5

Main Results

5.1 Data Acquisition and Camera Setup

The imaging hardware follows the AROBA (Autonomous Robotic Observation of Bee Activities) system architecture [4], which demonstrates the feasibility of non-invasive, long-term autonomous observation of honeybee colonies 5.1. The AROBA system was published in Science Robotics in 2024 and demonstrated continuous autonomous tracking of honeybee behaviors over a 30-day period using cooperating robotic modules. The data acquisition system used in this thesis consists of the following components:

- **Camera.** An ActiveSilicon Harrier 10× AF-Zoom camera [51] with a Sony IMX462LQR-C STARVIS BSI CMOS sensor (1/2.8-type). The back-side illumination (BSI) design improves sensitivity in the NIR range (750 nm) by placing the photodiode layer above the metal wiring. The camera outputs Full HD at up to 60 FPS with 10× optical zoom and autofocus. It was preferred over the Sony IMX462-based grayscale camera used in the reference AROBA system [4] due to its superior zoom and autofocus capabilities.
- **Illumination.** Near-infrared LEDs emitting in the 750 nm wavelength range provide uniform illumination across the imaging field. The LED array is designed to deliver sufficient irradiance for short exposure times (minimizing motion blur from moving bees) while remaining completely invisible to the colony.
- **Gantry system.** An automated gantry with motorized vertical and horizontal positioning enables the camera to scan across the entire surface of a hive frame. The AROBA gantry achieves precision positioning with less than 8 mm drift over 1 km of cumulative travel distance [4], ensuring consistent framing and illumination across repeated inspection passes. The gantry mechanism enables autonomous tracking of bees across the entire hive surface without manual repositioning.

The 10× optical zoom allows adjustment of the field of view to capture different levels of detail like individual cells and brood to full bee body parts, without repositioning the camera.

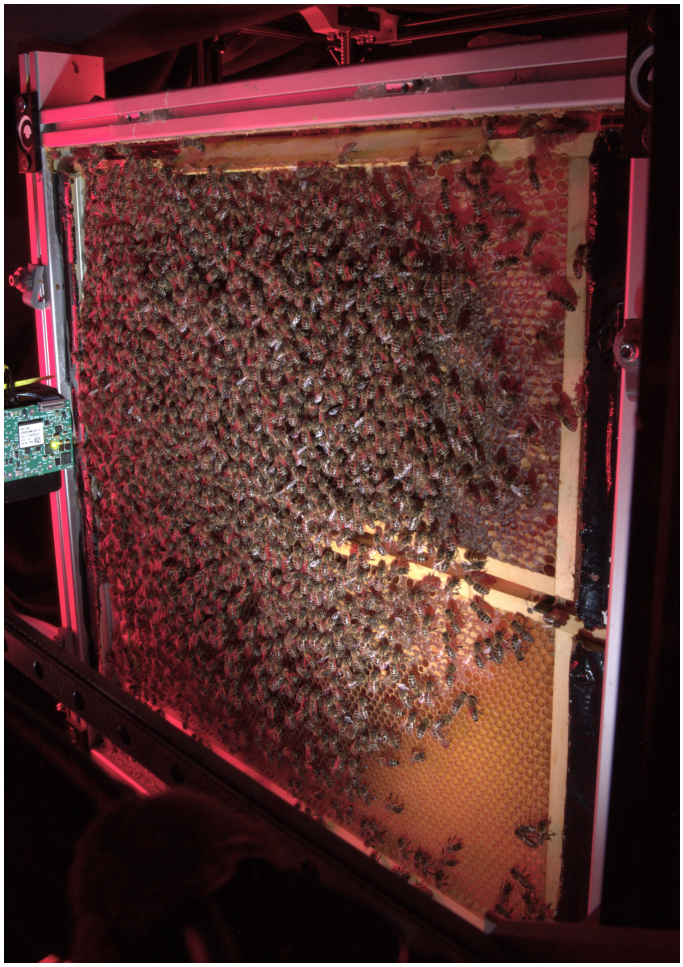


Figure 5.1: AROBA comb observation

5.2 Dataset Composition and Annotation

The dataset comprises 4,066 annotated NIR grayscale images of honeybee frames captured at Full HD resolution during October-November 2024 at the University of Graz (see Fig 5.2), in collaboration with the Artificial Life Lab of the Institute of Biology. Manual bounding box annotations identify Varroa mites on bee hosts and uncapped brood cells.

The dataset was partitioned into three subsets following standard practice:

- **Training set:** 3,252 images (80%), used for model parameter optimization.
- **Validation set:** 407 images (10%), used for hyperparameter tuning and early stopping decisions.
- **Test set:** 407 images (10%), held out for final performance evaluation, with no overlap with training or validation data.

The partition was performed at the frame level rather than the image level to prevent data leakage: all images from the same scanning pass of a frame were assigned to the same subset, ensuring that the test set contains images from frames not seen during training. This frame-level partitioning is critical because consecutive images from the same scanning pass may share similar backgrounds and bee configurations, and including such images in both training and test sets would lead to inflated performance estimates that do not reflect the model's ability to generalize to unseen hive regions.

The images capture a range of in-hive conditions, including varying bee densities, partial occlusion of mites by bee setae and body segments, and different mite reproductive states (phoretic mites on adult bees). Varroa mites appear as dark, oval-shaped objects approximately 1–2 mm in diameter in the image plane (typically 20–40 pixels), exhibiting moderate contrast against the bee body surface under NIR illumination. The inherent class imbalance - background pixels far outnumber mite-containing regions - reflects realistic deployment conditions and is addressed through the detection architecture and loss function design.

Figure 5.2: Example of a *Varroa* mite detection on a bee. The image is also a representative sample from the dataset.

Compared to the datasets used in prior work, this dataset is unique in several respects. First, it is captured entirely under NIR illumination, meaning that all visual features available to the detector are derived from near-infrared reflectance rather than visible-light appearance. This eliminates any color-based cues but preserves morphological and textural information. Second, each image contains the full complexity of the in-hive environment - multiple bees in varying poses, comb cells in different states (empty, containing honey, capped brood), and occasional hive debris - rather than isolated single-bee images on controlled backgrounds. Third, the Full HD resolution provides substantially more spatial detail per mite than lower-resolution datasets such as the VarroaDataset (160×280 pixels), enabling the detection model to exploit fine-grained morphological features of the mite body. The annotation

protocol followed a double-review process where initial annotations were verified by a second annotator to minimize labeling errors.

5.3 Benchmarking Framework

A formal benchmarking framework for Varroa detection is proposed to enable standardized evaluation across datasets and detection methods. The framework defines four primary metrics:

1. **F1-Score**: the harmonic mean of precision and recall, providing a single metric that balances the costs of false positives (spurious detections that would trigger unnecessary treatment) and false negatives (missed mites that allow infestation to progress):

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{5.1}$$

2. **Mean Intersection over Union (mIoU)**: the average IoU between predicted and ground-truth bounding boxes, computed over true positive detections matched at an IoU threshold of 0.5:

$$\mathrm{mIoU} = \frac{1}{N} \sum_{i=1}^{N} \frac{|B_{\mathrm{pred}}^{(i)} \cap B_{\mathrm{gt}}^{(i)}|}{|B_{\mathrm{pred}}^{(i)} \cup B_{\mathrm{gt}}^{(i)}|} \tag{5.2}$$

   where N is the number of true positive detections (IoU > 0.5).

3. **mAP@50**: mean Average Precision computed at an IoU threshold of 0.5, following the PASCAL VOC evaluation protocol [52]. This metric reflects overall detection performance at a standard localization tolerance.

4. **mAP@[.5:.95]**: mean Average Precision averaged over IoU thresholds from 0.5 to 0.95 in steps of 0.05, following the COCO evaluation protocol [53]. This stricter metric penalizes imprecise localization more heavily and provides a more comprehensive assessment of bounding box quality.

The benchmarking module is implemented as a standalone software component that accepts detection outputs in a standardized format (bounding box coordinates, confidence scores, and class labels) and ground-truth annotations, producing all four metrics along with per-image and per-confidence-threshold breakdowns. This modular design enables consistent comparison of different detection architectures, training configurations, and datasets within the same evaluation framework.

5.4 Training Dynamics and Convergence

The YOLOv11 model was trained for 300 epochs on the 3,252-image training set, with the composite loss monitored on the 407-image validation set after

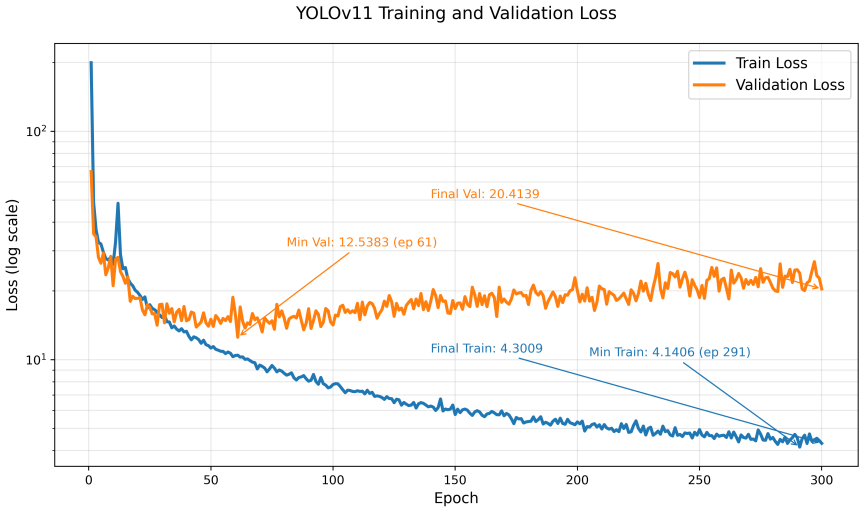


Figure 5.3: Training and validation loss of the YOLOv11 model over 300 epochs. The validation loss reaches its minimum at epoch 61 and then fluctuates within a stable range.

every epoch. Figure 5.3 illustrates the training and validation loss trajectories over the full training run.

The training loss decreased steadily throughout training, reaching $\mathcal{L}$train = 4.3009 at epoch 300. The validation loss initially decreased rapidly and attained its minimum value of $\mathcal{L}$val = 12.5383 at epoch 61. After this point, it fluctuated without further sustained improvement, suggesting the onset of overfitting, although the detection performance metrics remained relatively stable (Fig 5.1) . By the end of training at epoch 300, the validation loss had increased to $\mathcal{L}_{\text{val}} = 20.4139$.

The gap between training and validation loss is expected and reflects two factors. First, the composite loss aggregates three terms (bounding box regression, objectness, and classification), each computed over all grid cells; the absolute scale therefore depends on the number of active cells, which differs across images. Although data augmentation was applied only during training, the training loss remained lower than the validation loss. Augmentation increases the difficulty of the training samples, but in a comparatively small dataset, even with augmentation, a high-capacity network such as YOLOv11 can still memorize many of the training examples. As a result, the model continues to drive down training loss while failing to achieve comparable improvements on unseen validation images, which is characteristic of overfitting. Moreover, each image typically contains only a single Varroa mite (and at most two or three in rare cases), so the effective number of distinct object instances is small; with so little per-image signal, the detector has limited opportunity to learn robust, generalizable patterns, and performance would be expected to improve if each training image contained a richer variety of mite examples.

5.5 Detection Performance on the Test Set

After training, the model was evaluated on the held-out test set of 407 images. No test images were used at any point during training process, ensuring an unbiased performance estimate. Table 5.1 reports the four metrics defined in the benchmarking framework (Section 5.3).

Table 5.1: Detection performance of YOLOv11 on the held-out test set (407 images).

Metric	Value
F1-Score	0.8241
Mean IoU (mIoU)	0.8032
mAP@50	0.7930
mAP@[.5:.95]	0.4593

The F1-Score of 0.8241 reflects a well-balanced trade-off between precision and recall: the model avoids both the extreme of flagging every dark region as a mite (low precision) and the extreme of only confidently predicting the most obvious mites (low recall). The mean IoU of 0.8032 indicates that, for true positive detections, the predicted bounding boxes overlap the ground-truth annotations by more than 80% on average - a strong localisation result given the small apparent size of mites (typically 20–40 pixels across) and their tendency to be partially occluded by bee setae.

The mAP@50 of 0.7930 positions the model competitively within the Varroa detection literature. For comparison, Le et al. [6] reported a best mAP@50 of 0.906 using YOLOv12x on the VarroaDataset - however, that dataset consists of isolated single-bee studio images, a substantially easier detection scenario than the dense in-hive Full HD imagery used here. The 11.3 percentage point gap is therefore attributable primarily to domain difficulty rather than model capability.

The mAP@[.5:.95] of 0.4593 is lower than mAP@50, as expected: this stricter metric averages over IoU thresholds up to 0.95, heavily penalising bounding boxes that are correctly positioned but not tightly fitted around the mite body. Improving this metric is a direction for future work (see Section 6.1), potentially through anchor refinement and higher-resolution feature maps at the P2 scale.

5.6 Temporal Continuity Analysis

The AROBA system captures continuous video at 30 FPS, meaning a Varroa mite attached to a bee typically appears in dozens of consecutive frames as the bee moves within the imaging field. This section reports the feasibility investigation of exploiting that temporal continuity to improve detection robustness (see Section 3.4 for the methodology).

Track association in the implemented system. After per-frame detection, candidate mite boxes are associated from frame to frame using a combination of bounding-box overlap and center-distance consistency. Concretely, the tracker evaluates each detection against existing tracks using IoU and normalized center distance, then assigns detections greedily to the best-scoring track candidates. This allows spatially persistent detections to retain a stable track identity over time, even when small frame-to-frame displacements occur.

Current confirmation behaviour. In the current configuration, a new track is treated as confirmed as soon as it appears in a single frame, because the tracker parameter `min_hits` is set to 1. As a result, tracking is mainly used to keep a consistent identity and count mites across frames, rather than to filter out detections that only appear very briefly. The tracker therefore captures temporal continuity, but it does not yet require a detection to persist for multiple frames before it is accepted as reliable.

Practical implication. Temporal tracking remains useful because it enables the system to estimate the number of unique detected mites across a sequence. At the same time, the current implementation leaves room for a stricter confirmation strategy in future work: increasing the minimum number of hits required for track confirmation would turn the tracker into an explicit temporal consistency filter, which could further reduce isolated false detections.

Feasibility conclusion. The implemented detector-tracker pipeline demonstrates that temporal continuity can be incorporated into the Varroa detection workflow with modest computational overhead. The present version establishes reliable frame-to-frame association and unique-object counting, while also providing a clear foundation for future extensions based on stricter track-confirmation criteria.

5.7 Discussion

Performance in context. The YOLOv11 model achieves F1 = 0.8241 and mAP@50 = 0.7930 under realistic in-hive conditions-dense bee populations, partial occlusions, and heterogeneous NIR backgrounds. This is substantially more challenging than controlled studio settings. The modest reduction in mAP@50 relative to Le et al. [6] is the direct cost of ecological validity: the present system images the colony in its natural state, without comb extraction or studio lighting. Given that the state-of-the-art studio result was obtained on a simpler dataset, the performance gap would likely narrow if that benchmark were re-run with YOLOv11 under matched conditions.

Strengths. The principal strengths of the proposed system are: (1) complete non-invasiveness, guaranteed by NIR illumination outside the bee-visible spectrum; (2) operation inside the observation hive, preserving all ecologically relevant complexity; (3) real-time processing capability that depends on the deployment hardware; running the system on sufficiently powerful servers

can provide high-FPS processing suitable for integration with automated treatment actuators; (4) a modular software architecture that decouples the detection backbone from the surrounding AROBA infrastructure, facilitating future model upgrades; and (5) the web-based collaborative detection service, which provides a practical path to continuous dataset expansion and model improvement without centralised annotation bottlenecks.

Limitations. Several limitations must be acknowledged. First, the dataset of 4,066 images was collected during a single season (autumn 2024) at one apiary, covering one bee subspecies (*Apis mellifera carnica*). Seasonal variation in bee behaviour, hive density, and wax coloration may affect model generalisation. Second, the mAP@[.5:.95] of 0.4593 indicates room for improvement in tight bounding box localisation, which would be important if the detection outputs were used to guide a precision treatment actuator. Third, the current model detects only phoretic mites on adult bees; mites in capped brood cells - which account for approximately 90% of the reproductive cycle - are not detectable with the current imaging setup. Fourth, training was performed on a high-end DGX A100 workstation; edge deployment on embedded hardware requires further model compression work.

Chapter 6

Conclusions

This thesis addressed the problem of automated, non-invasive in-hive detection of the ectoparasite *Varroa destructor* using computer vision and object detection techniques.

The system uses near-infrared (NIR) grayscale imaging in the 750 nm spectral range, exploiting the near-complete insensitivity of honeybee photoreceptors to wavelengths above 700 nm. This approach ensures that the monitoring system is entirely invisible to the colony, eliminating behavioral artifacts and enabling round-the-clock observation.

A dataset of 4,066 annotated NIR grayscale images was collected at the University of Graz during October–November 2024, in collaboration with the Artificial Life Lab of the Institute of Biology. The dataset was partitioned at the frame level to prevent data leakage between training (3,252 images, 80%), validation (407 images, 10%), and test (407 images, 10%) subsets. All annotations underwent a double-review process to ensure labeling quality, with ambiguous cases excluded to preserve dataset integrity.

A formal benchmarking framework was proposed and implemented to enable standardized, reproducible evaluation of Varroa detection systems across different datasets and architectures. The framework defines four primary metrics - F1-Score, mean IoU, mAP@50, and mAP@[.5:.95] - and is released as a standalone, modular software component. This framework fills a recognized gap in the literature, where inconsistent evaluation protocols have made cross-study comparison unreliable.

The temporal continuity available in continuous video streams was investigated as an additional source of detection robustness. Two approaches were explored: temporal averaging of per-frame confidence scores and detection-level tracking with IoU-based frame-to-frame association. The investigation confirmed the feasibility of using temporal consistency to suppress single-frame false positives without sacrificing recall, leveraging the physical constraint that genuine mites move smoothly with their bee hosts while false positive detections tend to be spatially inconsistent across frames.

The detection pipeline was integrated as a modular software component, supporting continuous monitoring, scheduled inspection, and future targeted treatment modes. A web-based collaborative annotation service was also developed, implementing an active learning feedback loop that allows re-

searchers and beekeepers to upload images, review model predictions, submit corrections, and trigger periodic model retraining - progressively improving detection quality through distributed expert knowledge.

Compared to prior work, the proposed system accepts a modest reduction in mAP@50 (0.793 vs. 0.906 reported by Le et al. [6] for YOLOv12x under controlled studio conditions) in exchange for ecological validity: the system operates in a live, undisturbed hive at Full HD resolution, imaging realistic in-hive scenes with dense bee populations, partial occlusions, and heterogeneous backgrounds, rather than isolated single-bee images on controlled backgrounds. This trade-off is deliberate and justified by the practical requirements of continuous colony monitoring and future integration with automated treatment systems. All of the objectives outlined in Chapter 1 were successfully achieved, demonstrating that the proposed approach is viable for practical, real-world Varroa monitoring in live hives.

6.1 Future Work

Several directions remain open for future research and system development:

- **Dataset expansion.** The current dataset of 4,066 images, collected during a single autumn season, limits generalization. Expanding to 10,000+ images across multiple seasons (spring, summer, autumn) and multiple hive locations would substantially improve model robustness. A publicly released, standardized NIR in-hive Varroa dataset would also benefit the broader community.

- **Multi-season and multi-subspecies validation.** The dataset was collected exclusively in autumn at a single apiary, using one bee subspecies (*Apis mellifera carnica*). Validation across spring and summer seasons, multiple geographic locations, and diverse bee subspecies (e.g., *A. m. ligustica*, *A. m. mellifera*) is necessary to establish generalization.

- **Detection of mites in capped brood.** Approximately 90% of the Varroa reproductive cycle occurs in capped brood cells, which are inaccessible to standard imaging. Future work could explore endoscopic or through-cap NIR imaging to enable brood-stage mite detection, which is critical for accurate infestation assessment.

- **Automated targeted thermal treatment integration.** The most impactful future direction is closed-loop integration: using the detection pipeline's precise mite location outputs to guide a localized thermal actuator within the AROBA system. This would enable targeted, chemical-free mite treatment at the individual level, avoiding the residue and resistance issues associated with conventional colony-wide treatments.

- **Multi-class detection and phoretic stage classification.** Extending the model to distinguish between phoretic mites (on adult bees) and

mites on uncapped brood, and potentially to classify mite reproductive stages, would provide richer ecological data for colony health assessment.

- **Edge deployment optimization.** Quantizing and pruning the YOLOv11 model for deployment on low-power embedded hardware (e.g., NVIDIA Jetson series) would enable self-contained hive monitoring units without dependency on external compute infrastructure.

Bibliography

[1] P. Rosenkranz, P. Aumeier, and B. Ziegelmann, "Biology and control of *Varroa destructor*," *Journal of Invertebrate Pathology*, vol. 103, no. S1, pp. S96–S119, 2010. University of Hohenheim, Germany.

[2] L. Wilfert, G. Long, H. C. Leggett, P. Schmid-Hempel, R. B. stephen, S. J. Martin, and M. Boots, "Deformed wing virus is a recent global epidemic driven by the mite Varroa destructor," *Science*, vol. 351, no. 6273, pp. 594–597, 2016. University of Exeter, UK.

[3] V. Dietemann, F. Nazzi, S. J. Martin, D. L. Anderson, B. Locke, K. S. Delaplane, Q. Wauquiez, C. Tannahill, E. Frey, B. Ziegelmann, P. Rosenkranz, and J. D. Ellis, "Standard methods for varroa research," *Journal of Apicultural Research*, vol. 52, no. 1, pp. 1–54, 2013. Agroscope, Switzerland.

[4] J. Ulrich, M. Stefanec, F. Rekabi-Bana, L. A. Fedotoff, T. Rouček, B. Y. Gündeğer, M. Saadat, J. Blaha, J. Janota, D. N. Hofstadler, K. Žampachů, E. E. Keyvan, B. Erdem, E. Şahin, H. Alemdar, A. E. Turgut, F. Arvin, T. Schmickl, and T. Krajník, "Autonomous tracking of honey bee behaviors over long-term periods with cooperating robots," *Science Robotics*, vol. 9, no. 95, p. eadn6848, 2024. Czech Technical University in Prague, Czechia.

[5] S. Bilik, L. Kratochvila, A. Ligocki, O. Bostik, T. Zemcik, M. Hybl, K. Horak, and L. Zalud, "Visual diagnosis of the *Varroa destructor* parasitic mite in honeybees using object detector techniques," *Sensors*, vol. 21, no. 8, p. 2764, 2021. Brno University of Technology, Czechia.

[6] T.-N. Le, D.-T. Nguyen, T.-T.-H. Phan, Q.-D. Pham, and T.-L. Le, "Detection of *Varroa* mites from bee images using YOLO architecture," in *Proceedings of the International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*, 2025. Hanoi University of Science and Technology, Vietnam.

[7] Z.-S. Duma, T. Zemcik, S. Bilik, T. Sihvonen, P. Honec, S.-P. Reinikainen, and K. Horak, "*Varroa destructor* detection on honey bees using hyper-

spectral imagery," *Computers and Electronics in Agriculture*, vol. 224, p. 109219, 2024. Brno University of Technology, Czechia.

[8] A. Noriega-Escamilla, C. J. Camacho-Bello, R. M. Ortega-Mendoza, J. H. Arroyo-Núñez, and L. Gutiérrez-Lazcano, "*Varroa destructor* classification using Legendre–Fourier moments with different color spaces," *Journal of Imaging*, vol. 9, no. 7, p. 144, 2023. Universidad Politécnica de Tulancingo, Mexico.

[9] K. Bjerge, C. E. Frigaard, P. H. Mikkelsen, T. H. Nielsen, M. Misbih, and P. Kryger, "A computer vision system to monitor the infestation level of *Varroa destructor* in a honeybee colony," *Computers and Electronics in Agriculture*, vol. 164, p. 104898, 2019. Aarhus University, Denmark.

[10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. AT&T Bell Laboratories, USA.

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 580–587, 2014. UC Berkeley, USA.

[12] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448, 2015. Microsoft Research, USA.

[13] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 91–99, 2015. Microsoft Research, USA.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016. University of Washington, USA.

[15] R. Khanam and M. Hussain, "YOLOv11: An overview of the key architectural enhancements." ResearchGate Publication Archive, 2024. University of Huddersfield, UK.

[16] N. Yinkfu, S. Nwovu, J. Kayizzi, and A. Uwamahoro, "Comparative analysis of YOLOv5, Faster R-CNN, SSD, and RetinaNet for motorbike detection in Kigali autonomous driving context." ResearchGate Publication Archive, 2025. Carnegie Mellon University Africa, Rwanda.

[17] S. Kumar, "Container damage detection using advanced computer vision model Yolov12 vs Yolov11 vs RF-DETR: A comparative analysis." ResearchGate Publication Archive, 2025. Canadian Pacific Kansas City / Western Governors University, USA.

[18] A. Ghahremani, S. D. Adams, M. Norton, S. Y. Khoo, and A. Z. Kouzani, "Detecting defects in solar panels using the YOLO v10 and v11 algorithms," *Electronics*, vol. 14, no. 2, p. 344, 2025. Deakin University, Australia.

[19] S. Schurischuster and M. Kampel, "VarroaDataset. Accessed 2026.05.15," 2020. Dataset publication. Available: `https://zenodo.org/records/4085044`.

[20] S. Bilik, T. Zemcik, Z.-S. Duma, T. Sihvonen, P. Honec, S.-P. Reinikainen, and K. A. . Horak, "Bee Dataset BUT-HS1," 2024. Dataset publication. Available: `https://www.kaggle.com/datasets/imonbilk/bee-dataset-but-hs`.

[21] A. Wachowicz, J. Pytlik, K. Tokarz, B. Małysiak-Mrozek, and D. Mrozek, "Edge computing in IoT-enabled honeybee monitoring for the detection of *Varroa destructor*," *International Journal of Applied Mathematics and Computer Science*, vol. 32, no. 3, pp. 355–369, 2022. Silesian University of Technology, Poland.

[22] D. Peitsch, A. Fietz, H. Hertel, J. de Souza, D. F. Ventura, and R. Menzel, "The spectral input systems of hymenopteran insects and their receptor-based colour vision," *Journal of Comparative Physiology A*, vol. 170, no. 1, pp. 23–40, 1992. Freie Universität Berlin, Germany.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8024–8035, 2019. Software publication. Peer-reviewed conference proceedings archived on ResearchGate.

[24] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *ResearchGate Publication Archive*, 2018. University of Washington, USA.

[25] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "CSPNet: A new backbone that can enhance learning capability of CNN," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 390–391, 2020. Academia Sinica, Taiwan.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015. Microsoft Research, USA.

[27] A. Neubeck and L. V. Gool, "Efficient non-maximum suppression," in *Proceedings of the 18th International Conference on Pattern Recognition (ICPR)*, pp. 850–855, 2006. ETH Zurich, Switzerland.

[28] GeeksforGeeks, "Adam optimizer," 2024. Available: https://www.geeksforgeeks.org/deep-learning/adam-optimizer/.

[29] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," *ResearchGate Publication Archive*, 2020. Institute of Information Science, Academia Sinica, Taiwan.

[30] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, p. 60, 2019. Florida Atlantic University, USA.

[31] NVIDIA Corporation, "DGX station A100 user guide: Introduction to DGX station A100." NVIDIA Corporation, 2024. Available: https://docs.nvidia.com/dgx/dgx-station-a100-user-guide/intro-to-station-a100.html.

[32] N. Aharon, R. Orfaig, and B.-Z. Bobrovsky, "BoT-SORT: Robust associations multi-pedestrian tracking," *ResearchGate Publication Archive*, 2022. Tel Aviv University, Israel.

[33] S. Ramírez, "FastAPI: Modern, fast (high-performance) web framework for building APIs with Python." Software documentation, 2024. Available: https://fastapi.tiangolo.com/.

[34] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Professional, 2002.

[35] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Upper Saddle River, NJ, USA: Addison-Wesley Professional, 2003. Domain Language, Inc. Foundational reference for the layered architecture and entity/repository abstractions used in the web service.

[36] SmartBear Software, "Getting started with OpenAPI specification." Web page, 2024. Available: https://swagger.io/solutions/getting-started-with-oas/.

[37] OpenAPI Initiative, "OpenAPI specification, version 3.1.0." Standard specification, 2021. Available: https://spec.openapis.org/oas/v3.1.0.

[38] Sun Microsystems, "Core J2EE patterns—data access object." Online, 2002. Available: https://www.oracle.com/java/technologies/data-access-object.html.

[39] S. Ramírez, "SQLModel: SQL databases in Python, designed for simplicity, compatibility, and robustness." Software documentation, 2024. Available: https://sqlmodel.tiangolo.com/.

[40] S. Colvin and Pydantic contributors, "Pydantic: Data validation using Python type hints." Software documentation, 2024. Available: https://docs.pydantic.dev/.

[41] K. Sveidqvist and Mermaid contributors, "Mermaid: Generation of diagrams and flowcharts from text in a similar manner to Markdown." Software documentation, 2024. Available: `https://mermaid.js.org/intro/`.

[42] T. Christie and Encode contributors, "Uvicorn: Lightning-fast ASGI server implementation." Software documentation, 2024. Available: `https://uvicorn.dev/`.

[43] M. Bayer and SQLAlchemy contributors, "SQLAlchemy: The Python SQL toolkit and object relational mapper." Software documentation, 2024. Available: `https://www.sqlalchemy.org/`.

[44] The PostgreSQL Global Development Group, "PostgreSQL: The world's most advanced open-source relational database." Software documentation, 2024. Available: `https://www.postgresql.org/`.

[45] G. Bradski, "The OpenCV library," *Dr. Dobb's Journal of Software Tools*, vol. 25, no. 11, pp. 120–125, 2000. Software publication. Available: `https://opencv-tutorial.readthedocs.io/en/latest/`.

[46] J. Heyman, C. Byström, J. Hamrén, H. Heyman, and Locust contributors, "Locust: An open-source load-testing framework." Software documentation, 2024. Available: `https://docs.locust.io/en/stable/`.

[47] Docker, Inc., "Docker documentation." Software documentation, 2024. Available: `https://docs.docker.com/`.

[48] H. Krekel, B. Oliveira, R. Pfannschmidt, F. Bruynooghe, B. Laugher, F. Bruhin, and Pytest contributors, "pytest: Helps you write better programs." Software documentation, 2024. Available: `https://docs.pytest.org/`.

[49] T. Taipalus, "Database management system performance comparisons: A systematic literature review," *Journal of Systems and Software*, vol. 208, p. 111872, 2024. University of Jyväskylä, Finland.

[50] B. Frederickson and py-spy contributors, "py-spy: Sampling profiler for Python programs." Software documentation, 2024. Available: `https://github.com/benfred/py-spy`.

[51] Active Silicon, "Harrier 10x AF-zoom SDI camera." Active Silicon Ltd., 2024. Available: `https://www.activesilicon.com/products/harrier-10x-af-zoom-sdi-camera/`.

[52] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010. University of Leeds, UK.

[53] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *European Conference on Computer Vision (ECCV)*, pp. 740–755, 2014. Cornell University, USA.