



Fachhochschule Nordwestschweiz

Hochschule für Informatik

Master of Science in Engineering

Edge ML Camera for Citizen Science

Author:	Andri Wild
Supervisor:	Prof. Thomas Amberg
Advisors:	Prof. Thomas Amberg
Start Date:	16.09.2024
Submission Date:	07.02.2025

I confirm that this Master of Science in Engineering's thesis is my own work and
I have documented all sources and material used.

Windisch, 07.02.2025

Andri Wild

Abstract

Note:

1. **paragraph:** What is the motivation of your thesis? Why is it interesting from a scientific point of view? Which main problem do you like to solve?
2. **paragraph:** What is the purpose of the document? What is the main content, the main contribution?
3. **paragraph:** What is your methodology? How do you proceed?

Zusammenfassung

Note: Insert the German translation of the English abstract here.

Inhaltsverzeichnis

1 Einleitung	7
1.1 Ausgangslage	7
1.2 Zielsetzung	7
1.3 Abgrenzung	7
1.4 Aufbau des Berichts	7
2 Grundlagen	8
2.1 Citizen Science	8
2.2 Machine Learning Grundlagen	10
2.2.1 ML Frameworks	10
2.2.1.1 TensorFlow	10
2.2.1.2 TensorFlow Lite	10
2.2.1.3 PyTorch	10
2.2.1.4 ONNX	10
2.2.1.5 NCNN	11
2.2.1.6 Hailo	11
2.2.1.7 Ultralytics	11
2.2.2 Anatomie von Machine Learning Modellen	11
2.2.3 Geschwindigkeit einer Inferenz	13
2.3 Edge Computing	13
2.3.1 Edge Intelligence	14
2.3.2 Host Systeme	14
2.3.2.1 Raspberry Pi 4	14
2.3.2.2 Raspberry Pi 5	14
2.3.2.3 BeagleY-AI	15
2.3.3 Hardware Beschleuniger	15
2.3.3.1 Coral USB Accelerator	15
2.3.3.2 Coral PCI Accelerator	16
2.3.3.3 Hailo TPU	16
2.3.3.4 AI-Kamera	17
2.4 Mitwelten Biodiversitäts Monitoring	18
2.4.1 Mitwelten Bestäuber Analyse	18
2.4.2 Edge Architektur	19
3 Analyse	20
3.1 Zielgruppe	20
3.2 Use Cases	20
3.2.1 Referenz Szenario	20
3.3 Anforderungen	20
3.3.1 Funktionale	21
3.3.2 Nicht Funktionale	21
3.4 Evaluation	21
3.4.1 Methodik	22

3.4.2 ML Framework	22
3.4.2.1 Vergleich Inferenzzeiten	22
3.4.3 Hardware	23
3.4.3.1 Raspberry Pi Vergleiche	23
3.4.3.2 Coral Accelerator	24
3.4.3.3 Hailo	24
3.5 Edge ML Setups	25
4 Umsetzung	27
4.1 System Architektur	27
4.1.1 Komponenten Diagram	27
4.1.2 Qualitäten	27
4.1.3 Konfiguration	28
4.1.4 Webinterface	28
4.2 Software Design	29
4.3 Prototypen: verschiedene System Setups	31
4.3.1 Prototyp 1 - Mitwelten Analyse	31
4.3.1.1 ONNX	31
4.3.1.2 NCNN	32
4.3.1.3 Hailo	32
4.3.2 Prototyp 2 - AI-Camera	33
4.4 Adaptierung des Systems	34
4.5 Lizenzierung	34
5 Validierung	35
5.1 Versuchsaufbau und Temperaturentwicklung	35
5.2 Konkurrenz Produkte	36
6 Fazit	37
Abbildungsverzeichnis	38
Tabellenverzeichnis	40
Appendix A: Supplementary Material	41
Bibliographie	42

1 Einleitung

1.1 Ausgangslage

Das SNF Projekt „Mitwelten - Medienökologische Infrastrukturen für Biodiversität“ hat IoT-Technologie in urbanen Naturgebieten installiert, um dort vorhandene Tiere, Pflanzen und Umweltbedingungen genauer zu untersuchen. Ein Teilprojekt beschäftigte sich mit der Erkennung von Bestäuber-Arten auf Blumenblüten mittels Raspberry Pi Kameras. Die Machine-Learning-basierte Detektion und Kategorisierung geschieht zurzeit in einem zentralen Cloud-Backend.

Diese Architekturlösung bringt folgende Konsequenzen mit sich: Zum Einen muss der Betrieb eines Backends sichergestellt sein. Dies erfordert einen gewissen Wartungsaufwand und insbesondere ein erhöhtes technisches Verständnis für die Projektumsetzung. Für Forschende bedeutet dies, dass zusätzliches IT-Fachpersonal für den Aufbau und Betrieb der IT-Infrastruktur benötigt wird. Zum Anderen werden die aufgenommenen Bilder zur Analyse über Mobilfunknetze versendet, wodurch die Betriebskosten erhöht werden und das Einsatzgebiet auf dessen Abdeckung begrenzt. Beide der genannten Aspekte sind mit zusätzlichen Kosten verbunden und erhöhen die Gesamtkomplexität des Projekts.

1.2 Zielsetzung

Das Ziel ist die Portierung der ML-Pipeline des SNF Projekts Mitwelten auf eine Edge ML Kamera, um Citizen Science Projekte zu ermöglichen:

Dank der fortschreitenden Entwicklung von Edge-Computing-Hardware sind auf Machine Learning spezialisierte Geräte zu einem vergleichbaren Preis wie konventionelle Edge Computer erhältlich. Dies wirft die Frage auf, ob durch diesen technologischen Fortschritt die IT-Infrastruktur eines Edge-Kamera basierenden Systems so weit reduziert werden kann, dass die selbe Funktionalität ohne zentrales Backend umsetzbar ist. Der Einsatz von Edge-basierten Systemen könnte den Zugang zu Machine Learning Projekten für Forschende erheblich erleichtern, da ein System ohne zentrale Abhängigkeit mit weniger Aufwand betreibbar ist. Diese stand-alone Lösung öffnet auch die Türen für Citizen Science Projekte. Einzelpersonen oder Gemeinschaften können so eigenständig und unabhängig Forschungsprojekte durchführen. Daraus ergeben sich die folgenden Forschungsfragen:

- Welche Hardware und Frameworks gibt es, um Machine Learning (ML) dezentral, im Feld, mittels Edge Computing umzusetzen?
- Was sind Anforderungen an eine ML-Kamera für typische Citizen Science Anwendungen im Bereich Biodiversitätsmonitoring?
- Wie sieht eine konkrete Edge ML Kamera aus, für Monitoring von Biodiversität, wie im Projekt SNF Mitwelten angewendet?

1.3 Abgrenzung

Ein System zur Analyse der Biodiversität mittels Machine Learning beinhaltet viele verschiedene Aspekte. In diesem Projekt liegt der Fokus auf der aktuellsten Hardware und wie diese für Edge ML eingesetzt werden kann. Nicht Teil dieses Projekt ist das Trainieren von Machine Learning Modellen und deren Evaluation.

1.4 Aufbau des Berichts

Der Bericht besteht im wesentlichen aus vier Teilen. Im ersten Teil werden die Grundlagen vermittelt, welche relevant sind für das Verständnis des Berichts. Anschliessend folgt die Analyse, welche die Zielgruppen und Anforderungen definiert. In der Umsetzung wird eine konkrete Implementierung einer Edge ML Kamera vorgestellt und abschliessend wird die Evaluation der Resultate vorgenommen.

2 Grundlagen

Dieses Kapitel beschreibt den Theoretischen Inhalt dieser Arbeit. Um die späteren Analysen und Evaluationen verstehen zu können, werden die grundlegenden Konzepte, Technologien und Frameworks vorgestellt, die in diesem Projekt zum Einsatz kommen. Dazu gehören die Prinzipien des Machine Learnings, die Rolle von Edge Computing sowie die eingesetzten Hardware- und Softwarekomponenten.

2.1 Citizen Science

Das grundlegende Ziel dieser Arbeit ist die Entwicklung eines Systems, welches im Bereich von Citizen Science eingesetzt werden kann. Daher soll als erstes der Begriff erläutert werden. Eine treffende Definition von Citizen Science liefert das Grünbuch Citizen Science Strategie 2020 für Deutschland:

„Citizen Science beschreibt die Beteiligung von Personen an wissenschaftlichen Prozessen, die nicht in diesem Wissenschaftsbereich institutionell gebunden sind. Dabei kann die Beteiligung in der kurzzeitigen Erhebung von Daten bis hin zu einem intensiven Einsatz von Freizeit bestehen, um sich gemeinsam mit Wissenschaftlerinnen bzw. Wissenschaftlern und/oder anderen Ehrenamtlichen in ein Forschungsthema zu vertiefen. Obwohl viele ehrenamtliche Forscherinnen und Forscher eine akademische Ausbildung aufweisen, ist dies keine Voraussetzung für die Teilnahme an Forschungsprojekten. Wichtig ist allerdings die Einhaltung wissenschaftlicher Standards, wozu vor allem Transparenz im Hinblick auf die Methodik der Datenerhebung und die öffentliche Diskussion der Ergebnisse gehören.“

— A. Bonn u. a. [1]

Bürgerinnen und Bürger können also einen Beitrag zur Wissenschaft leisten, indem sie Daten sammeln, analysieren und interpretieren. Dies kann von der einfachen Datenerhebung bis hin zur intensiven Auseinandersetzung mit einem Forschungsthema reichen. Die Beteiligung an Citizen Science Projekten ist nicht an eine akademische Ausbildung gebunden, jedoch ist die Einhaltung wissenschaftlicher Standards unabdingbar. Es stellt sich die Frage, worin die motivation liegt sich freiwillig solchen Projekten zu widmen. Ein treibender Faktor kann der Wissenszuwachs für ein bestimmtes Thema sein. Die Auseinandersetzung in einem Bereich, für den man sich interessiert. Das Buch The Science of Citizen Science [2, p 283] diskutiert die verschiedenen Aspekte des Lernens in einem Citizen Science Projekt.

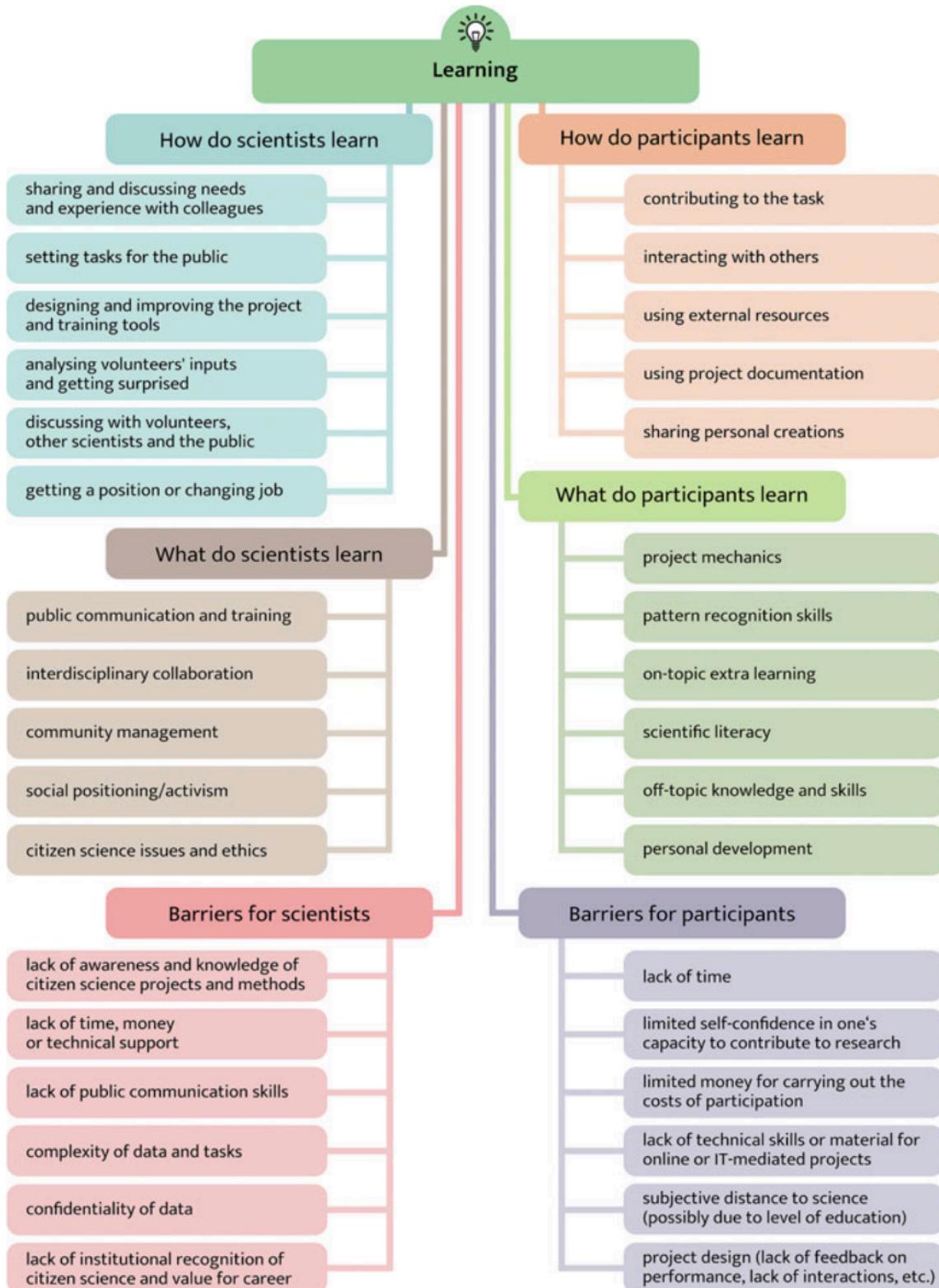


Abbildung 1: Erweiterte thematische Karte zum Lernen von Freiwilligen

(Quelle: The Science of Citizen Science, S.300)

Aus der Illustration geht hervor, dass verschiedene Aspekte die Bürgerinnen und Bürger zu einer aktiven Beteiligung motivieren kann. Nebst den Fachlichen sind auch Soziale Themen von Bedeutung.

Auf der anderen Seite profitieren auch die Projekt Initianten. Die Organisation und Zusammenarbeit mit Aussenstehenden kann eine spannende Aufgabe sein. Es stellt die Projekt-Verantwortlichen vor neue Herausforderungen um die Freiwilligen motivierend in das Projekt miteinzubeziehen. Ein Grundstein dieser Angelegenheit ist die Zugänglichkeit zu Projekt Instrumenten. In Bezug auf ein Projekt mit Edge ML kann es besonders wichtig sein,

dass die Freiwilligen verstehen wie das Setup und die damit verbundene Technik funktioniert und damit Faszination entfachen kann.

2.2 Machine Learning Grundlagen

2.2.1 ML Frameworks

Ein Machine-Learning-Framework bildet die Grundlage für die Entwicklung, das Training und die Bereitstellung von Modellen. Es bietet Werkzeuge und Bibliotheken, die den gesamten ML-Workflow unterstützen. Im Gegensatz dazu ist ein Machine-Learning-Modell das konkrete Ergebnis des Trainingsprozesses, das spezifische Aufgaben wie Klassifikation oder Objekterkennung ausführt. Wird ein Modell nach dem Training benutzt, um zum Beispiel um Objekte zu erkennen, spricht man von Inferenz. Dies beschreibt den Prozess der Ausführung des Modells auf ungewöhnlichen Daten. Das Training sowie die Inferenz eines Modells kann mit unterschiedlichen Frameworks vollzogen werden. Jedes Framework hat ihr eigenes Modell Format. So lassen sich beispielsweise PyTorch Modelle nicht ohne weiteres mit TensorFlow ausführen. Folgend sind diejenigen vorgestellt, welche in dieser Arbeit zum Einsatz kamen.

2.2.1.1 TensorFlow

TensorFlow [3] wurde ursprünglich von Google entwickelt und ist inzwischen ein Open Source Projekt. Das Framework ist eher für Systeme in Produktion gedacht.

- Vorteile: Weit verbreitet mit grosser Community und umfangreicher Dokumentation. Stellt viele Features zu Verfügung.
- Nachteile: Steile Lernkurve, insbesondere für Einsteiger. Komplex durch die vielen Features.

2.2.1.2 TensorFlow Lite

Stellt die für Edge Geräte optimierte Variante des TensorFlow Framework dar.

- Vorteile: Speziell für den Betrieb auf ressourcenbeschränkten Geräten optimiert. Reduzierte Speicher- und Rechenanforderungen für Echtzeit-Anwendungen.
- Nachteile: Eingeschränkte Unterstützung für komplexe Modelle, Quantisierung erforderlich.

2.2.1.3 PyTorch

PyTorch [4] ist ebenfalls Open Source und hat seinen Ursprung im Forschungsteam für künstliche Intelligenz von Facebook. Das Framework eignet sich für Experimente kann aber auch für Systeme in Produktion eingesetzt werden. ChatGPT von OpenAI arbeitet beispielsweise im Hintergrund mit dem Pytorch Framework [5].

- Vorteile: Hat eine gute Community und eine ausführliche Dokumentation. Viele Features und viele Tutorials.
- Nachteile: Steile Lernkurve, insbesondere für Einsteiger.

2.2.1.4 ONNX

ONNX (Open Neural Network Exchange) [6] ist ein generalisiertes Format von Deep-Learning Modellen. Dies ermöglicht den Austausch von Modellen zwischen verschiedenen Frameworks. ONNX wurde ursprünglich von Facebook und Microsoft entwickelt. Zurzeit wird das Framework von mehreren Partnern als Open-Source Projekt weiterentwickelt.

- Vorteile: Austauschformat, das Modelle zwischen verschiedenen Frameworks kompatibel macht. Optimierte Laufzeitumgebungen, die speziell für Edge-Geräte geeignet sind.
- Nachteile: Begrenzte Funktionalität für Modelltraining, primär für den Einsatz trainierter Modelle gedacht. Kleinere Community im Vergleich zu TensorFlow und PyTorch.

Eine Spezialität des ONNX Framework ist die Fähigkeit, Modelle von anderen Formaten in das ONNX Format zu konvertieren. Ebenso lassen sich ONNX Modelle wieder zu anderen Formaten exportieren. Dies führt dazu, dass ONNX als Art Drehscheibe zwischen den verschiedenen Frameworks fungiert [7].

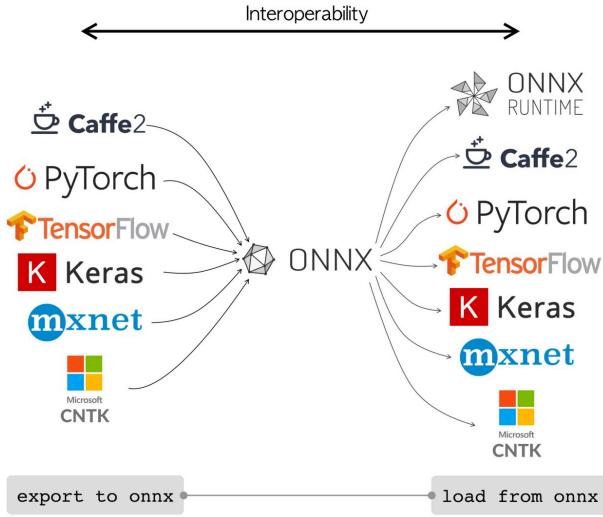


Abbildung 2:

(Quelle: <https://docs.ultralytics.com/integrations/onnx/>, aufgerufen am 02.02.2025)

2.2.1.5 NCNN

NCNN [8] ist eine high performance computing platform für mobile Endgeräte. Dieses Framework wird oft auf Smartphone verwendet, weil es optimiert für Geräte mit beschränkter Rechenleistung ist.

- Vorteile: Schnelle Inferenzzeiten auf der CPU.
- Nachteile: Im Vergleich zu anderen Frameworks weniger ausführlich dokumentiert.

2.2.1.6 Hailo

Hailo, gegründet im Jahr 2017, hat sich als führendes Unternehmen im Bereich leistungsfähiger KI-Prozessoren für Edge-Anwendungen etabliert. Mit ihrer eigens entwickelten Hardware-Architektur verfolgt Hailo das Ziel, Machine Learning auch ausserhalb von Rechenzentren zugänglich und effizient nutzbar zu machen [9]. Die Nutzung der Hailo-Chips erfordert eine Konvertierung der Modelle, wofür das Unternehmen eine umfassende Software-Suite bereitstellt. Diese Suite enthält alle notwendigen Werkzeuge, um Modelle auf die spezifischen Anforderungen der Hailo-Hardware abzustimmen.

- Vorteile: Ermöglicht sehr schnelle Inferenzen. Hailo bietet eine gute Dokumentation und ein grosses Ökosystem.
- Nachteile: Eine intensive Einarbeitung ist notwendig und der Konvertierungsprozess erfordert Modellspezifisches Wissen.

2.2.1.7 Ultralytics

Ultralytics [10] ist ein auf PyTorch basierendes Open-Source-Framework, das vor allem für die Entwicklung von Modellen zur Objekterkennung bekannt ist. Es wurde durch die Implementierung von YOLOv5 (You Only Look Once) populär und bietet eine benutzerfreundliche Umgebung für das Training und die Bereitstellung von Objekterkennungsmodellen. Inzwischen sind neuere Implementierungen der YOLO Familie verfügbar und mit Ultralytics anwendbar.

- Vorteile: Die Verwendung des Frameworks ist sehr einfach und dadurch geeignet für Einsteiger. Modelle können in andere Formate exportiert werden. Inferenzen werden für PyTorch Modelle angeboten.
- Nachteile: Die Flexibilität im Vergleich zu anderen Frameworks ist eingeschränkt.

2.2.2 Anatomie von Machine Learning Modellen

Ein Machine-Learning-Modell besteht aus einer Architektur, die dessen Aufbau und Funktionsweise definiert, und einem Satz von Parametern, die während des Trainings optimiert werden. Die Architektur eines Modells legt grundlegende Eigenschaften fest, wie die Struktur der Neuronen und Layer, die Verbindungen zwischen ihnen sowie Eingabe- und Ausgabetensoren. Die Eingabetensoren bestimmen, welche Form und Dimensionen die Daten haben müssen (z.

B. die Größe von Bildern), während die Ausgabetensoren das Ergebnis des Modells beschreiben, etwa Klassenetiketten oder Koordinaten für erkannte Objekte.

Es gibt verschiedene Typen von Machine-Learning-Modellen, die je nach Anwendungsfall eingesetzt werden. Zu den wichtigsten gehören Modelle für Objekterkennung, die Objekte in einem Bild lokalisieren und klassifizieren, Bildsegmentierung, die jedem Pixel eines Bildes eine Klasse zuordnet, und Posenschätzung, die die Körperhaltung von Personen oder Tieren analysiert. Die folgende Abbildung 3 zeigt eine Visualisierung der verschiedenen Disziplinen.

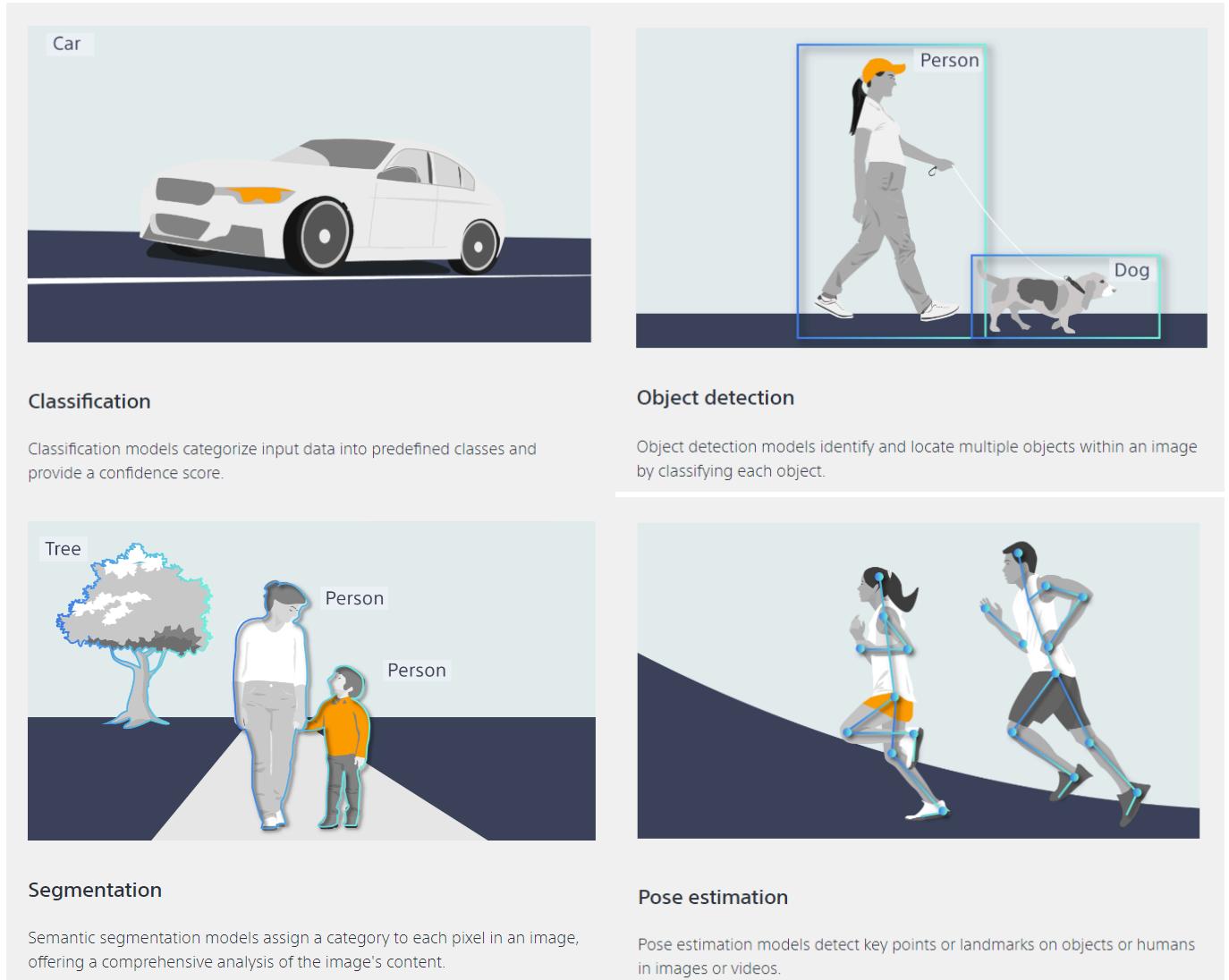


Abbildung 3: Machine Learning Tasks

(Quelle: https://github.com/sony/model_optimization, aufgerufen am 20.01.2025)

Für die Objekterkennung werden verschiedene Metriken verwendet, um die Modellleistung zu bewerten. Dazu zählen die Intersection over Union (IoU), die angibt, wie gut vorhergesagte und tatsächliche Direktionsfläche übereinstimmen, und die Mean Average Precision (mAP), die die Präzision über verschiedene Schwellenwerte hinweg aggregiert. Zusätzlich spielen Leistungskennzahlen wie die Inferenzzeit eine wichtige Rolle, insbesondere bei Echtzeitanwendungen.

Das Training eines Modells umfasst die Anpassung der Parameter auf Basis eines grossen Datensatzes, um Muster und Zusammenhänge zu lernen. Dabei kommen Optimierungsalgorithmen wie Gradient Descent zum Einsatz, die das Modell iterativ verbessern. Nach dem Training wird das Modell in der Inferenzphase genutzt.

Ein Modell hat definierte Input und Output Tensoren. Das heisst, dass die Daten für das Training und die Inferenz dem Input Tensor entsprechen müssen. Diesen Vorgang nennt man preprocessing. Arbeitet das Modell mit Bildern beinhaltet dieser Schritt das modifizieren des Bildes auf die Input Grösse des Modells. Ebenso variiert der Output Tensor je nach Modell und Framework. Die Output Daten im Falle von Object Detection beschreiben dabei den Ort

des Objekts und die Confidence, also wie sicher sich das Modell ist, dass dieses Objekt zu einer bestimmten Klasse gehört. Je nach Format dieser Daten müssen konvertiert und falls für die Anwendung und preprocessing notwendig wieder auf das Original Bild zurückgerechnet werden. Dieser Schritt wird postprocessing genannt.

2.2.3 Geschwindigkeit einer Inferenz

Die Dauer einer Inferenz hängt massgeblich von den Berechnungen des Modells und der Leistungsfähigkeit des Systems ab, auf dem sie ausgeführt wird. Die benötigte Anzahl an Berechnungen wird durch die Neuronenanzahl im neuronalen Netz bestimmt. Mit zunehmender Anzahl an Neuronen – sei es durch mehr oder grössere Hidden Layers – steigt der Rechenaufwand. Die Abbildung 4 illustriert den symbolischen Aufbau eines neuronalen Netzes mit drei Hidden Layers. Ein Netz mit mehr als einem Hidden Layer wird als „Deep Neural Network“ bezeichnet.

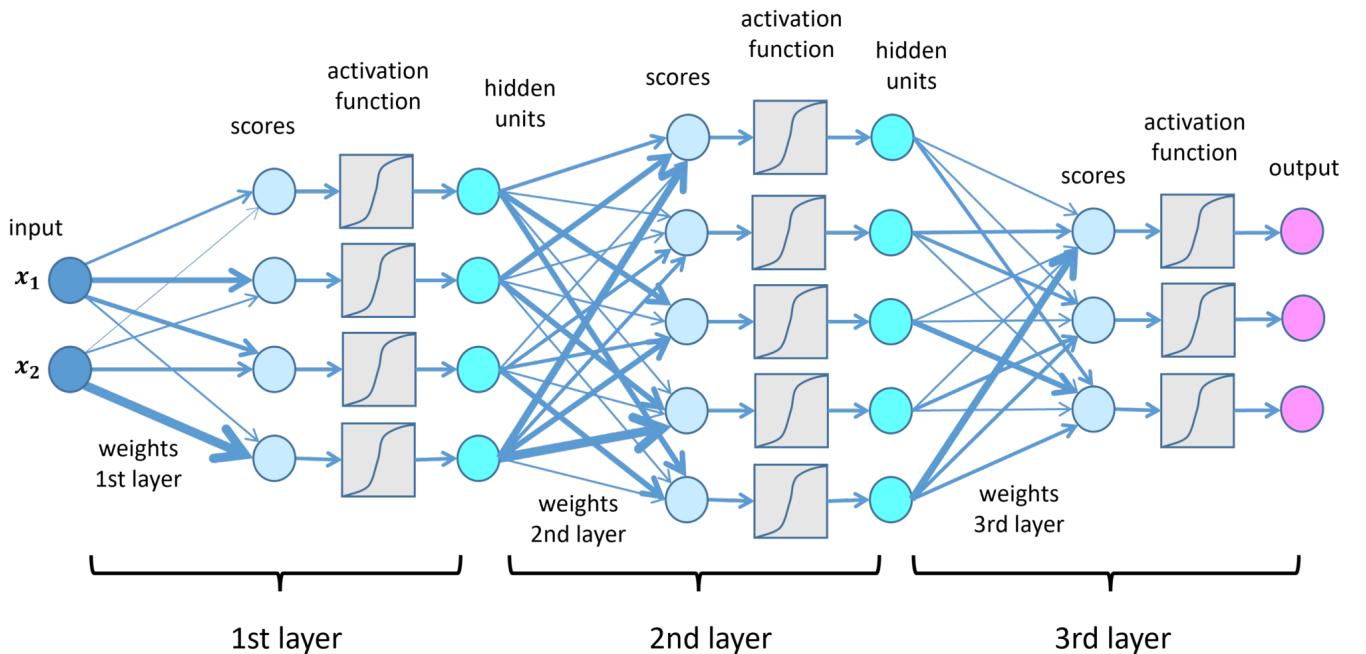


Abbildung 4: Symbolbild Neuronales Netzwerk

(Quelle: lamarr-institute.org/blog/deep-neural-networks, abgerufen am 23.01.2025)

Je mehr Hidden Layers und trainierte Gewichte ein Modell besitzt, desto präziser werden seine Vorhersagen. Allerdings steigt damit auch der Rechenaufwand. Aus diesem Grund existieren YOLO-Modelle, wie sie in dieser Arbeit verwendet wurden, in verschiedenen Dimensionen [11].

Ein weiterer entscheidender Faktor ist die Hardware, auf der die Berechnungen durchgeführt werden. Durch das Parallelisieren von Berechnungen lassen sich erhebliche Geschwindigkeitsvorteile erzielen. Besonders geeignet sind GPUs (Graphics Processing Units), TPUs (Tensor Processing Units) und NPUs (Neural Processing Units). Diese Komponenten werden in Abschnitt 2.3.3 beschrieben.

Während auf eine GPU grundsätzlich keine Vorbedingungen an das Modell gestellt werden, ist dies bei der TPU und NPU durchaus notwendig. Dabei spielen zwei Strategien eine wichtige Rolle: Quantisieren und Pruning. Beim Quantisieren werden die Fließkommazahlen der Gewichte zu Ganzzahlen konvertiert, wodurch die Präzision abnimmt. Dies reduziert den Speicherbedarf und ermöglicht eine schnellere Ausführung, weil weniger Rechenoperationen sind. Beim Pruning werden unwesentliche Verbindungen im Modell entfernt, was wiederum die Anzahl Rechenoperationen reduziert. Die Ansätze werden auch kombiniert angewendet [12].

2.3 Edge Computing

Edge Computing und Edge Machine Learning markieren einen Paradigmenwechsel in der Systemarchitektur, indem die Datenverarbeitung von zentralen Backend Server oder Cloud-Systemen hin zu Geräten am Rand des Netzwerks

(Edge) verlagert wird. Diese Architektur ermöglicht es, Berechnungen und Analysen direkt vor Ort durchzuführen, anstatt die Daten für die Verarbeitung zu versenden.

Dies bringt mehrere Vorteile mit sich. So ermöglicht die lokale Verarbeitung eine geringere Latenz, was vor allem für Echtzeitanwendungen entscheidend ist. Zudem reduziert sich der Datenverkehr zu einem Backend, was nicht nur die Betriebskosten senkt, sondern auch die Abhängigkeit von einer stabilen Internetverbindung minimiert. Nebst der Datenübertragung ist auch die Komplexität eines Systems von Bedeutung. Eine Systemarchitektur mit zentraler Einheit bedeutet nebst zusätzlichen Technologien auch einen erhöhten Wartungsaufwand, welcher gerade bei grosser Projektdauer nicht zu unterschätzen ist. Ein weiterer wesentlicher Vorteil liegt im Bereich der Datensicherheit. Sensible Informationen bleiben vor Ort und müssen nicht über Netzwerke übertragen werden, wodurch die Privatsphäre der Nutzer besser geschützt wird.

Allerdings bringt dieser Ansatz auch Herausforderungen mit sich. Edge-Geräte verfügen häufig über eingeschränkte Rechenleistung und Speicherressourcen, was die Ausführung komplexer Machine-Learning-Modelle erschweren kann. Darüber hinaus ist die Energieeffizienz ein zentraler Aspekt, da viele Edge-Geräte batteriebetrieben sind und die Optimierung des Energieverbrauchs entscheidend für den Betrieb sein kann. Schliesslich erfordert die Verwaltung und Skalierung einer Vielzahl von verteilten Geräten ohne zentrale Steuerung zusätzlichen Aufwand.

2.3.1 Edge Intelligence

Edge Intelligence kombiniert die Prinzipien des Edge Computing mit den Anforderungen moderner Machine-Learning-Algorithmen, indem die Rechenleistung direkt auf die Endgeräte verlagert wird. Man spricht in diesem Falle auch von AI on Edge [13]. Diese Algorithmen werden entweder auf der CPU ausgeführt oder durch spezialisierte Hardware beschleunigt. Diese ist darauf ausgelegt, Machine Learning Tasks performant, effizient und ressourcenschonend auszuführen. Beschleuniger Hardware ist dann erforderlich, wenn die Inferenz auf der CPU zu viel Zeit oder Ressourcen in Anspruch nimmt. Diese Hardware wird typischerweise mit einem Host System verbunden, welches mit der Beschleuniger Hardware kommunizieren kann.

Die Weiterentwicklung der kompakten Computer (Edge Device) sowie die auf ML spezialisierte Hardware öffnet neue Möglichkeiten im Bereich des Edge Computing. Zum Zeitpunkt dieser Arbeit sind die Entwicklungen im vollen Gange. Nicht nur auf Seite der Hardware, sondern auch im Umfeld des Machine Learning wird zur Zeit erforscht und entwickelt. Somit kann in diesem Projekt unter Anderem neuste Hardware eingesetzt werden, welche sich in der Industrie möglicherweise erst in der kommenden Zeit etablieren wird. Die im Projekt eingesetzte Hardware ist im folgenden aufgeführt.

2.3.2 Host Systeme

Der einfachste Weg Machine Learning im Bereich Edge zu betreiben ist die CPU eines Edge Computers. Um Erfahrungen zu sammeln sind verschiedene Einplatinen Computer evaluiert worden. Hauptsächlich hat sich die Verwendung von Raspberry Pi Computer durchgesetzt. Sie weisen ein gutes Preis Leistungsverhältnis auf und sind durch Dokumentation und Community extrem gut unterstützt.

2.3.2.1 Raspberry Pi 4

Das Raspberry Pi 4 8 GB ist für rund 85Fr. [14] erhältlich. Mit einer CPU Taktfrequenz von 1.5GHz lassen sich schon viele Anwendungen realisieren. Nachteil des Model 4 ist, dass keine PCI Schnittstelle vorhanden ist.

2.3.2.2 Raspberry Pi 5

Das Raspberry Pi 5 stellt den Nachfolger vom Model 4 dar und ist mit 8 GB RAM und einer CPU Taktfrequenz von 2.4GHz ausgerüstet. Ebenso ist eine PCI Schnittstelle verfügbar, wodurch das anschliessen von leistungsstarker Beschleuniger Hardware möglich wird. Dieses Setup ist für ca. 85 Fr. [15] erhältlich.

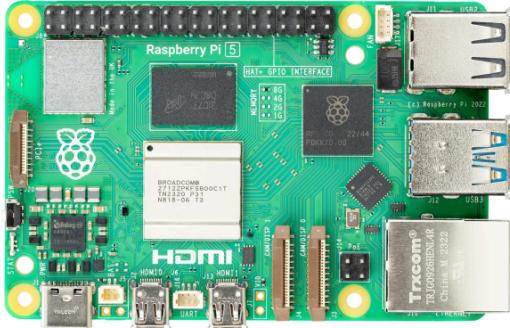


Abbildung 5: Raspberry Pi 5

(Quelle: <https://www.pi-shop.ch/raspberry-pi-5-16gb-ram?src=raspberrypi>, aufgerufen am 23.01.2025)

2.3.2.3 BeagleY-AI

Das BeagleY-AI Board hat eine CPU Taktfrequenz von 1.5GHz und 4 GB RAM. Das besondere an diesem Board ist der integrierte AI Accelerator mit 4 TOPS (Terra Operation per Second). Das Board ist für run 70 Fr. [16] erhältlich.

2.3.3 Hardware Beschleuniger

Machine-Learning-Beschleuniger sind spezialisierte Hardwarekomponenten, die entwickelt wurden, um die Ausführung von Machine-Learning-Modellen zu optimieren. Sie sind insbesondere für rechenintensive Aufgaben wie Matrixmultiplikationen und Tensorberechnungen ausgelegt, die in vielen ML-Algorithmen eine zentrale Rolle spielen. Zu den bekanntesten Typen solcher Beschleuniger gehören GPUs (Graphics Processing Units) und TPUs (Tensor Processing Units).

Eine GPU wurde ursprünglich für die Verarbeitung von Grafikanwendungen entwickelt, hat sich jedoch aufgrund ihrer Fähigkeit zur parallelen Verarbeitung von Operationen gleichzeitig als ideal für Machine-Learning-Aufgaben erwiesen. GPUs kommen häufig bei grossen Modellen und im Training von neuronalen Netzwerken zum Einsatz, da sie eine hohe Rechenleistung bieten.

Eine TPU [17] ist ein speziell entwickelter Prozessor, der ausschliesslich für Machine-Learning-Workloads optimiert wurde. Sie wurde von Google entwickelt und ist besonders effizient bei der Verarbeitung von Tensoroperationen, wie sie in Frameworks wie TensorFlow genutzt werden. TPU's sind für das Training und die Inferenz geeignet, wobei sie bei letzterem aufgrund ihrer geringen Latenz und Effizienz eine herausragende Rolle spielen.

Ebenso wurde die NPU [18] entwickelt, um ML-Tasks hochgradig parallelisiert und effizient auszuführen. In bestimmten Szenarien kann eine NPU eine GPU in Sachen Geschwindigkeit deutlich übertreffen [19].

NPU ausführen

2.3.3.1 Coral USB Accelerator

Der USB-Dongle von Google, Coral USB Accelerator, hat eine dedizierte TPU (Tensor Processing Unit). Solche Geräte lassen sich leicht in bestehende Systeme integrieren und ermöglichen die schnelle Ausführung von ML-Algorithmen ohne signifikante Anpassungen der Infrastruktur. Der Preis bewegt sich um 89.90Fr.[20].



Abbildung 6: Coral USB Accelerator

(Quelle: <https://www.coral.ai/products/accelerator>, aufgerufen am 22.01.2025)

2.3.3.2 Coral PCI Accelerator

Nebst den USB-Dongles stellt Google auch über die PCI Schnittstelle angeschlossene Beschleuniger her. Diese sind als single oder dual TPU erhältlich. Der Preis für die signle Lösung mit 4TOPS liegt bei 47.90Fr. [21], für die dual Lösung mit 8 TOPS bezahlt man 96.90 Fr.[22].



Abbildung 7: Coral PCI Accelerator

(Quelle: <https://www.coral.ai/products/m2-accelerator-dual-edgetpu>, aufgerufen am 22.01.2025)

2.3.3.3 Hailo TPU

Hailo ist eine Firma die sich auf Edge-KI Prozessoren spezialisiert hat. Diese sind moderner als die Lösungen von Google und erreichen eine höhere Leistungsfähigkeit. Die in diesem Projekt verwendeten Beschleuniger haben 13 und 26 TOPS zu 79.90 Fr.[23] resp. zu 122.90 Fr.[24]. Die folgende Abbildung zeigt ein Hailo Brschleuniger auf einem Raspberry Pi Abbildung 8.



Abbildung 8: Hailo Accelerator

(Quelle: <https://www.pi-shop.ch/raspberry-pi-ai-hat-13t>, aufgerufen am 22.01.2025)

2.3.3.4 AI-Kamera

Eine weitere Kategorie sind AI-Kameras, die mit integrierten ML-Chips ausgestattet sind. Diese Kameras, wie auf Abbildung 9 dargestellt, können nicht nur Bilder aufnehmen, sondern auch direkt auf der Kamera Inferenzdaten bereitstellen, beispielsweise durch die Erkennung und Klassifikation von Objekten. Raspberry Pi stellt eine solche Kamera zur Verfügung für 77.90 Fr. [25].



Abbildung 9: Raspberry Pi AI Camera

(Quelle: <https://www.berrybase.ch/raspberry-pi-ai-camera>, aufgerufen am 22.01.25)

Dieser Ansatz verlagert die intensiven Berechnungen womit der Edge Computer weniger Ressourcen bereitstellen muss. Die Abbildung 10 verdeutlicht, wie die Architektur eines solchen Systems aufgebaut ist. Die linke Seite zeigt den Aufbau herkömmlicher Kamera Architekturen, während rechts die Lösung mittels AI Accelerator dargestellt ist.

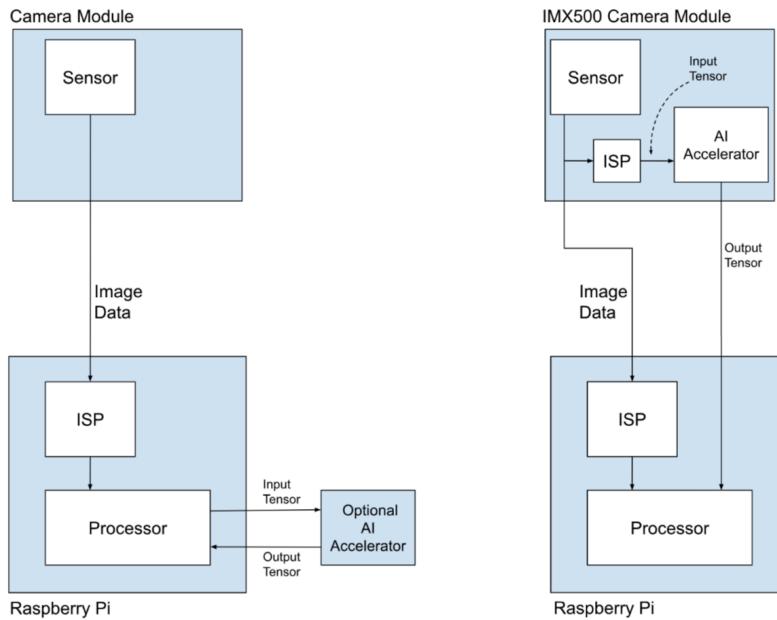


Abbildung 10: Raspberry Pi AI Camera Architektur

(Quelle: <https://www.raspberrypi.com/documentation/accessories/ai-camera.html>, aufgerufen am 20.01.2025)

Beim Sensor handelt es sich um einen Sony IMX500 Chip [26], welcher das Bild erfasst. Die Daten gelangen als RAW-Image zum ISP, einem kleinen Image Signal Processor. Dieser wandelt das RAW-Image zu einem Input Tensor, welcher vom AI-Accelerator als Input Grösse entgegen genommen wird. Nach durchführen der Inferenz gelangen die Resultate, in diesem Fall der Output Tensor zum Processor des Host Systems. Gleichzeitig gelangt auch das Image auf das Host-System, um wenn nötig die Resultate anzuzeigen oder weiterverarbeitet zu werden. Die Bilder können wahlweise mit 10 FPS bei einer Auflösung von 4056x3040 oder bei 30 FPS mit 2028x1520 aufgenommen werden.

2.4 Mitwelten Biodiversitäts Monitoring

Biodiversität Monitoring befasst sich grundsätzlich mit dem Beobachten und Analysieren von Tier - und Pflanzenwelt.

„Die biologische Vielfalt bildet eine Lebensgrundlage der Schweiz. Deshalb ist es wichtig, ihren Zustand und ihre Entwicklung zu kennen.“

— bdm [27]

Aus diesem Grund ist es von grosser Bedeutung die Umwelt zu beobachten und Veränderungen festzustellen. Um diese Arbeit zu vereinfachen, sind automatisierte Lösungen für ein Monitoring gefragt.

Ein System zur automatisierten Datenerfassung wurde im Rahmen des Projekts Mitwelten entwickelt [28], [29]. Hierfür wurden verschiedene Sensoren zu einem IoT-Toolkit kombiniert. Dieses Toolkit ermöglicht es, Daten von dezentralen Systemen zu erfassen und an ein zentrales Backend weiterzuleiten.

Das IoT-Toolkit umfasst unter anderem eine Kamera, die in regelmässigen Abständen Fotos von Blüten aufnimmt. Im Rahmen des Projekts wurden so insgesamt 1,5 Millionen Bilder generiert. Abbildung 11 zeigt eine im Feld aufgestellte IoT-Kamera.



Abbildung 11: IoT Kamera im Feld

(Quelle: mitwelten.ch, aufgerufen am 23.01.2025)

Die enorme Menge an Bilddaten lässt sich nicht manuell analysieren. Deshalb entwickelte das Projekt-Team eine Machine-Learning-Pipeline, die Bestäuber automatisch erkennt. Die technische Architektur des Systems verbindet die Kamera mit einem leistungsstarken Backend. Die Kamera, bestehend aus einem Raspberry Pi und einer angeschlossenen Kameraeinheit, erfasst die Bilder und überträgt diese über einen Access Point an das Backend. Auf dem Backend Server läuft eine Machine Learning Pipeline, welche die empfangenen Bildern nach Bestäuber untersucht.

2.4.1 Mitwelten Bestäuber Analyse

Die Bestäuber Detektion erfolgt in den folgenden Schritten. Ein Foto von mehreren Blüten, zum Beispiel von einem Blumentopf, durchläuft in einem ersten Schritt eine Machine Learning Analyse zur Detektion von Blüten. Die auf dem Bild detektierten Blüten werden anschliessen ausgeschnitten und somit zu einzelnen kleineren Bildern. Jedes dieser Bilder wird anschliessend mit einer weiteren Analyse und einem anderen Machine Learning Modell auf Bestäuber untersucht. Eine grosse Herausforderung in diesem Setup ist, dass die Anzahl Bestäuber-Erkennungen mit der Anzahl detektierten Blüten steigt. Dies kann bei grossen Blützenzahlen zu langen Analysen eines einzigen Bildes führen. Die folgende Abbildung 12 verdeutlicht den Prozess visuell.



Abbildung 12: Vorgang Mitwelten Bestäuber Analyse

(Quelle: Automated Analysis for Urban Biodiversity Monitoring, Timeo Wullschleger)

Um Bestäuber auf einer Blüte zu detektieren, ist ein Intervall von 15 Sekunden zur Erfassung von Bildern einzuhalten. Dies stellt die in etwa die Zeit dar, auf dem sich ein Bestäuber auf einer Blüte befindet.

15 Sekunden Intervall Quelle

2.4.2 Edge Architektur

Im Kontext dieser Entwicklung wurden verschiedene Architekturen untersucht [29, p 58]. Unter anderem ein System, auf dem die Machine-Learning-Pipeline auf einem Raspberry Pi 3 ausgeführt wird. Der Edge-Computer analysiert die Bilder vor Ort und schickt nur die Resultate an das Backend. Die Problematik mit der steigenden Anzahl Inferenzen pro detektierten Blüte kommt aber in diesem Setup besonders zu tragen. Die Abbildung 13 zeigt das Verhältnis der detektierten Blüten zur Analysezeit auf.

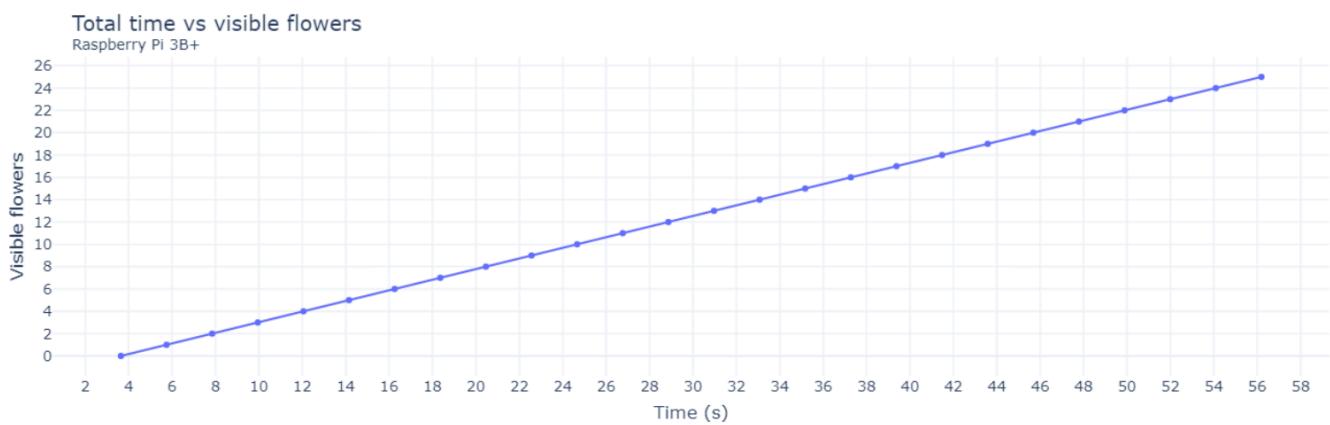


Abbildung 13: Mitwelten Analyse auf Raspberry Pi 3

(Quelle: Automated Analysis for Urban Biodiversity Monitoring S.61, Timeo Wullschleger)

Mit wachsender Anzahl Blüten auf dem Bild steigt die Zeit für die Bestäuber Analyse stark an. Da eine Kamera typischerweise immer die selben Blüten untersucht, würde ein solches Setup bei gleichbleibendem Bildintervall von 15 Sekunden immer mehr in Verzögerung geraten. Selbst wenn die Zeiten ohne Aufnahmen aufgrund schlechter Lichtverhältnisse für die Bildanalyse genutzt würden, bleibt diese Architektur in diesem Setup nicht realisierbar. Bei einem Bildintervall von 15s und 12h genügenden Lichtverhältnissen dürfte die Analyse maximal doppelt so lange, also 30 Sekunden für alle Blüten in Anspruch nehmen. Das Machine Learning Model wurde mit einer `max_detection` von 30 Blüten trainiert. Wie aus Abbildung 13 ersichtlich, reicht die Zeit nicht, die Analyse auf der Kamera, also dem Raspberry Pi auszuführen.

3 Analyse

Dieses Kapitel befasst sich mit der Analyse und Bewertung der Möglichkeiten für die Umsetzung einer Edge ML Kamera. Zuerst werden die Randbedingungen für das System eruiert in dem die Zielgruppe und die Anforderungen erarbeitet werden. Das Unterkapitel Evaluation stellt anschliessend verschiedene Resultate zu Untersuchungen von ML Frameworks vor. Zum Abschluss von diesem Kapitel werden verschiedene Möglichkeiten für von Hardware Setups vorgestellt.

3.1 Zielgruppe

Ein Edge ML Kamera, welche flexibel für verschiedene Zwecke einsetzbar ist, deckt dementsprechend eine breite Zielgruppe ab. Anwendungen könnten für die folgenden Interessengruppen von Bedeutung sein:

- Forschende: Wissenschaftlerinnen und Wissenschaftler aus den Bereichen Biodiversität, Ökologie oder Umweltwissenschaften.
- Citizen Scientists: Ehrenamtliche, die sich an wissenschaftlichen Projekten beteiligen.
- Naturschutzorganisationen: Institutionen, die Artenschutz- oder Umweltprojekte durchführen.
- Bildungseinrichtungen: Schulen, Universitäten und andere Lernstätten für Lehrzwecke und studentische Forschungsprojekte.
- Landwirtschaftliche Betriebe: Landwirte, die ihr Wissen über Bestäuber und Schädlingsbekämpfung erweitern möchten.

Die Bediener des Systems müssen die Edge ML Kamera in erster Linie für ihre Zwecke und Bedürfnisse anpassen können. Aus diesem Grund ist es von grosser Bedeutung, das System so einfach wie möglich zu halten. Da eine Anwendung spezifische Analyse von Bildern mittels Machine Learning nicht generalisiert werden kann, müssen die Benutzer ihre eigene Implementation in das System integrieren können. Ebenfalls muss die Erfassung von Bildern und die Destination der Analyse Resultate flexible gestaltet werden können. Beide Aspekte erfordern einen Eingriff in das System. Um diese Vorgänge zu vereinfachen, müssen dieses anhand von Anleitungen und Beispielen unterstützt werden.

3.2 Use Cases

Durch die grosse Anzahl an verschiedenen Zielgruppen gibt es auch viele verschiedene Szenarien, in dem der Einsatz einer Edge ML Kamera vorstellbar ist.

- Biodiversitätsforschung: Überwachung von Bestäubern, Vögeln, Säugetieren oder Pflanzenwachstum.
- Artenschutz: Identifikation und Überwachung gefährdeter Tier- oder Pflanzenarten.
- Umweltüberwachung: Aufzeichnung und Analyse des Einflusses von Umweltbedingungen wie Temperatur, Feuchtigkeit oder Lichtverhältnissen auf die Pflanzen oder Tierwelt.
- Landwirtschaft: Erkennung von Schädlingen, Optimierung des Pflanzenwachstums.

3.2.1 Referenz Szenario

Die Mitwelten Bestäuber Analyse dient während der Entwicklung der Edge ML Kamera als referenz Use Case. Um das System zielgerichtet voranzutreiben, wurden Tests und Evaluationen mittels dieser Analyse bewertet. Dies bietet sich an, weil bereits Machine Learning Modelle bestehen und dementsprechend auch Daten, um die Modelle zu trainieren.

3.3 Anforderungen

Wie in den Kapitel zuvor diskutiert, gibt es eine breite Anwendung für eine Edge ML Kamera. Der Referenz Use Case für die Entwicklung des Systems ist die in der Mitwelten Projekt eingesetzte Bestäuber Analyse. Dieser Anwendungsfall dient somit zur Definierung der Anforderungen an das System.

3.3.1 Funktionale

Die folgenden funktionalen Anforderungen an eine Edge ML Kamera wurden identifiziert:

- Datenerfassung: Das System muss anhand einer angeschlossenen Kameraeinheit Bilder kontinuierlich erfassen können
- Analyse Resultate: Das System muss die Analysedaten für den jeweiligen Anwendungszweck zur Verfügung stellen
- Benutzerinteraktion: Das System muss vom Benutzer konfiguriert werden können
- Energieversorgung: Während des Betriebs muss eine stabile Energieversorgung vorhanden sein

3.3.2 Nicht Funktionale

Die folgenden nicht funktionalen Anforderungen an eine Edge ML Kamera wurden identifiziert:

- Analysedauer: Im Mitwelten Projekt ist definiert, dass die Analyse von Bestäubern in 15 Sekunden erfolgen muss [29, p 62]. Die Zeit ist von der Verweildauer von Bestäuber auf einer Blume limitiert.
- Betrieb: Das System muss zuverlässig im Dauerbetrieb funktionieren.
- Kosten: Der Preis für das Gesamtsystem muss so tief sein, dass Citizen Science Projekte realistisch sind

3.4 Evaluation

Im folgenden werden Resultate der Untersuchungen bezüglich ML Frameworks und Hardware aufgezeigt. Dabei sind Inferenzen mit PyTorch, Tensorflow lite, ONNX, NCNN und Hailo Modellen implementiert worden. Die Inferenzen wurden auf Raspberry Pi's mit und ohne Beschleuniger Hardware ausgeführt und gemessen. Um die Resultate miteinander zu vergleichen und anhand von Grafiken zu untersuchen wurde ein Dashboard entwickelt [30]. Das Dashboard basiert auf CSV Daten, welche Test Scripts zur Messung von Inferenzzeiten automatisch generieren. Für jedes Framework existiert ein separates Script, welches ein Framework auf einer spezifischer Hardware mit variabler Anzahl Bilder ausführt. Die folgende Abbildung 14 zeigt den Start Screen der Applikation.

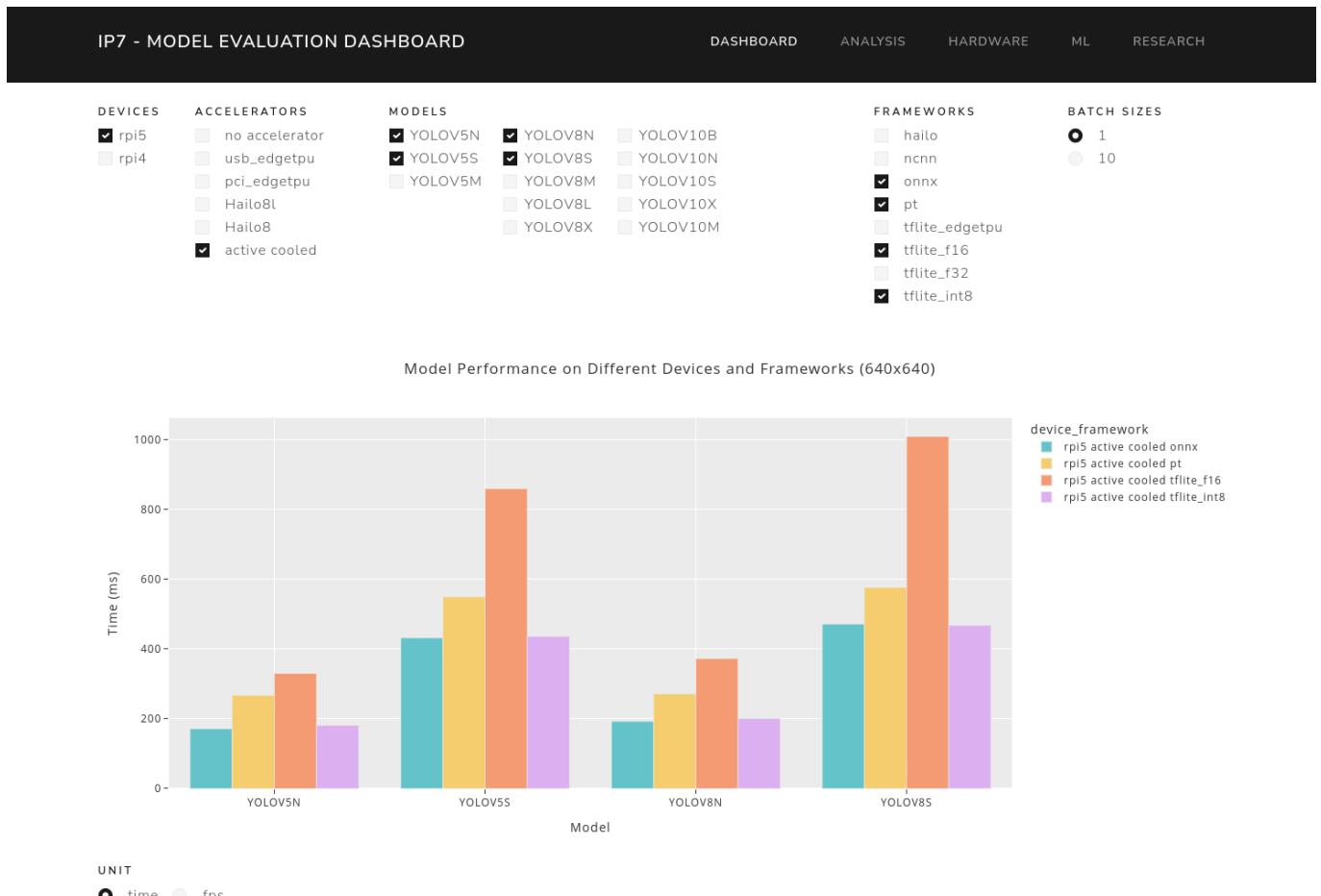


Abbildung 14: ML Exploration Dashboard
(Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)

Im Repository [31] befinden sich nebst dem Dashboard auch die Scripts zur Messung der Inferenzzeiten. Je nach Test muss die entsprechende Hardware an das System angeschlossen sein.

3.4.1 Methodik

Bei allen verwendeten Modellen handelt es sich um Yolo Modelle verschiedener Generation und Grösse. Die Modelle zur Bestäuber Analyse aus Abschnitt 2.4.1 sind ebenfalls Yolo Modelle der Generation 5. Inzwischen sind die Modelle weiter entwickelt worden. Während dieser Arbeit wurde die 11te Generation veröffentlicht. Verglichen wurden die Generationen 5,8 und 10 um festzustellen, wie sich die Metriken über die Generationen verhalten. Die Modelle stammen von Ultralytics [32] und haben eine Input Grösse von 640x640 Pixel. Die Inferenzen wurden jeweils auf dem COCO: Common Object in Context [33] durchgeführt. Dies ist der state-of-the-art Bilder Datensatz um einheitliche Metriken für Machine Learning Tasks zu bestimmen. Der Fokus in den folgenden Analysen liegt auf den Inferenzzeiten, wobei die Genauigkeit eine sekundäre Rolle spielt. Die verwendeten Yolo Modelle haben alle publizierte Metriken au dem COCO Datenset [34].

3.4.2 ML Framework

3.4.2.1 Vergleich Inferenzzeiten

Die folgende Abbildung Abbildung 15 zeigt auf, wie sich die Inferenzzeiten der jeweiligen ML Frameworks in Bezug auf die YOLO Generation verhält. Ersichtlich ist, dass hauptsächlich die Inferenzzeit der PyTorch Inferenz mit der neuen Generation signifikant gestiegen ist. Die anderen Frameworks weisen nur geringe Unterschiede auf, wobei die Tendenz des Anstieges bei jeder Kategorie feststellbar ist. Ein weiteres Merkmal ist, dass die Inferenz mit dem ONNX Model bei allen Versuchen am wenigsten Zeit benötigt.

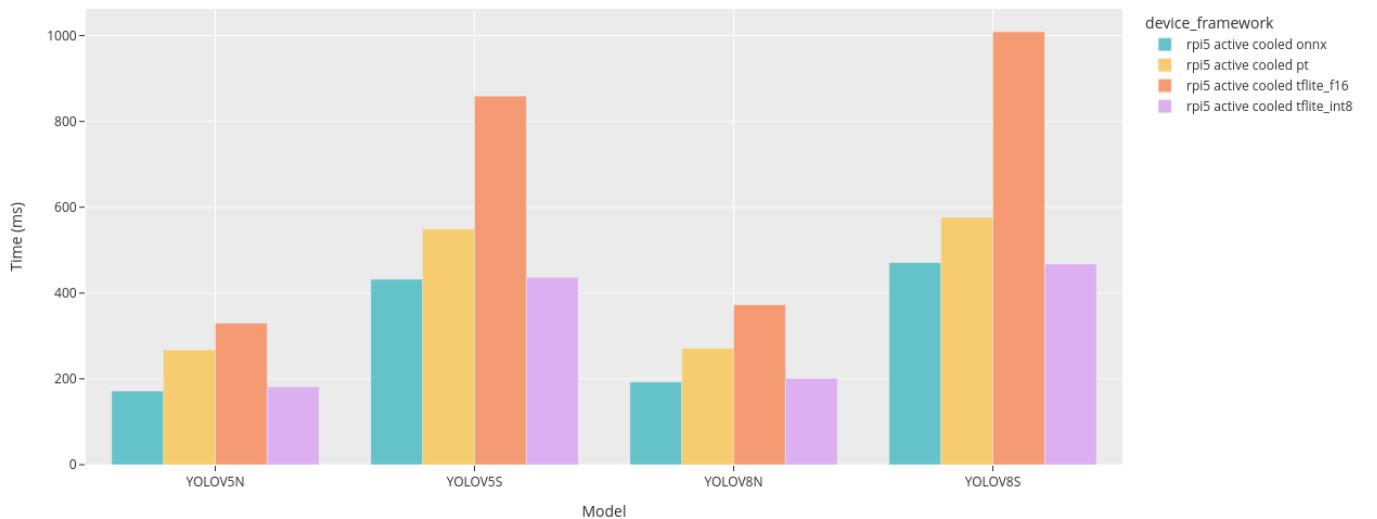


Abbildung 15: Vergleich: Inferenzzeit YOLOv5, v8 auf Raspberry Pi 5

In einem nächsten Schritt wird ONNX mit NCNN verglichen. Wie in Abschnitt 2.2.1.5 erwähnt, ist NCNN für Geräte mit begrenzter Rechenleistung optimiert. Dies zeigt sich auch in der folgenden Abbildung Abbildung 16. Die Inferenzzeiten halbieren sich nochmals gegenüber der Inferenz mit ONNX.

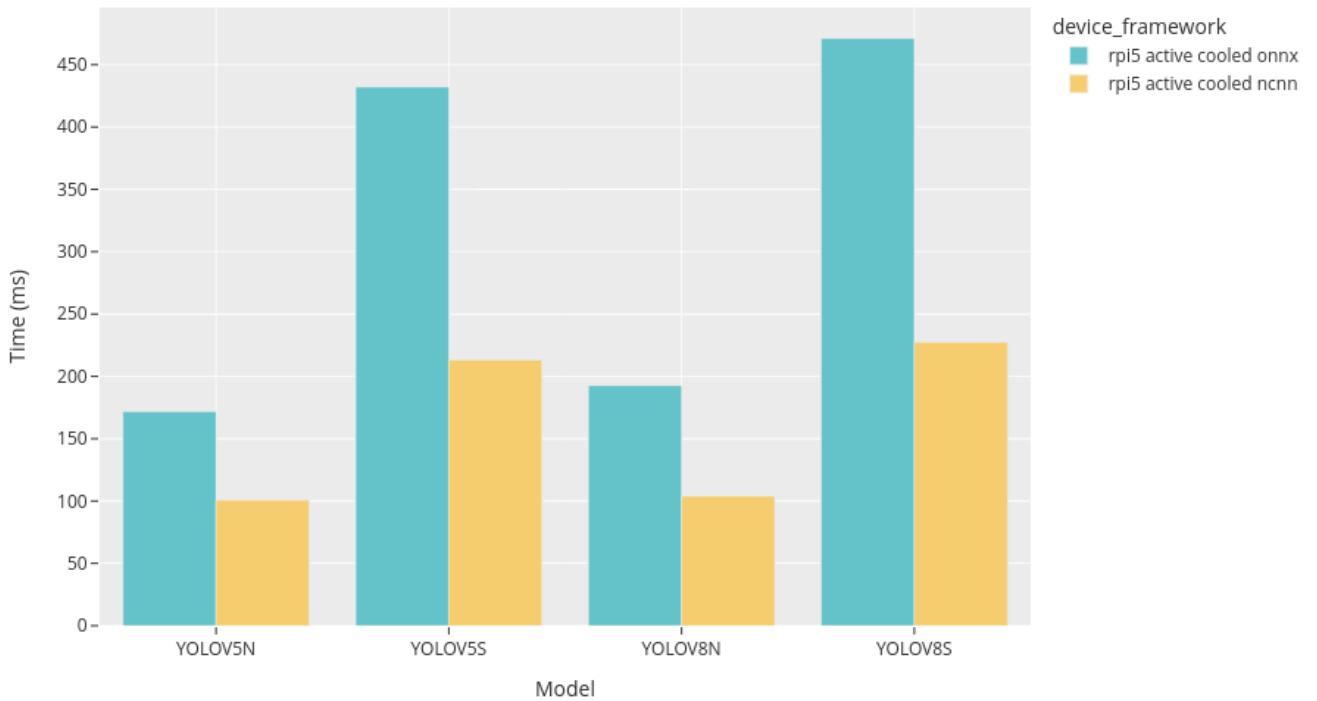


Abbildung 16: Vergleich: ONNX und NCNN auf Raspberry Pi 5

Aus diesen Analysen geht hervor, dass mit NCNN auf einer Raspberry Pi 5 CPU die schnellsten Ergebnisse erzielt werden können. Um nun die Inferenzzeiten weiter zu beschleunigen, wird zusätzliche Hardware benötigt.

3.4.3 Hardware

Im folgenden werden verschiedene Hardware Setups miteinander verglichen. Teilweise beansprucht spezifische Hardware auch definierte ML Frameworks. Somit ist der Vergleich von verschiedener Hardware nicht mit gleichen Frameworks möglich. Dennoch können die Inferenzzeiten verglichen werden.

3.4.3.1 Raspberry Pi Vergleiche

Der erste Vergleich zeigt den Fortschritt der Raspberry Pi Generationen auf. Die Inferenzzeiten eines ONNX Models ist auf einem Raspberry Pi 5 in grün mehr als doppelt so schnell.

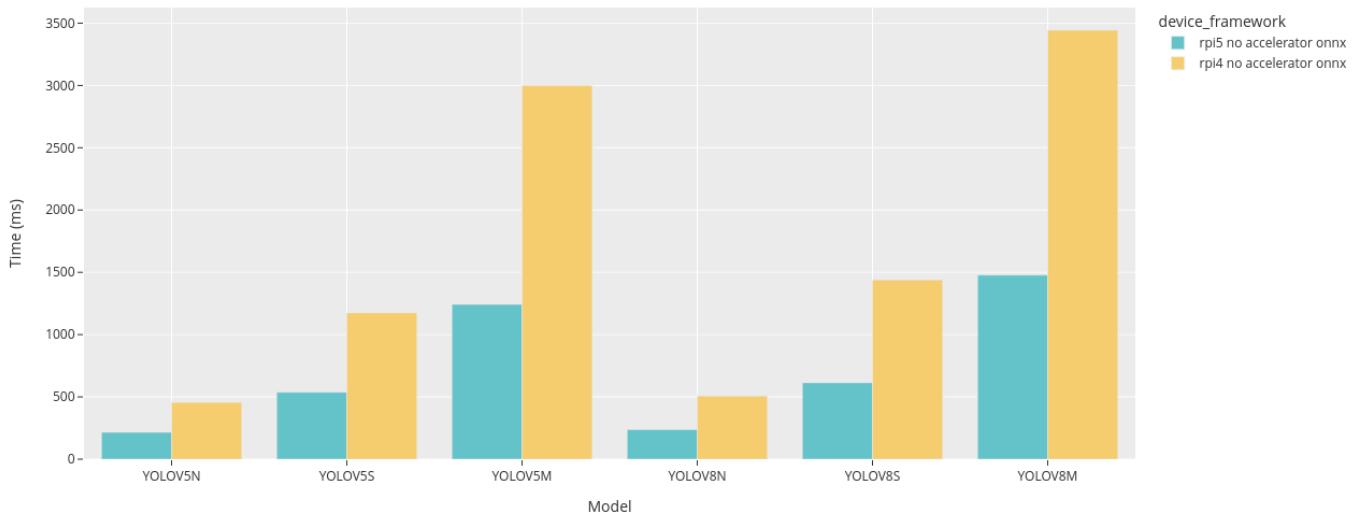


Abbildung 17: Vergleich: Raspberry Pi 4 vs. Pi 5

(Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)

Ein weiterer wichtiger Punkt ist die Kühlung des Systems, ansonsten wird die CPU Leistungs des Rechners gedrosselt. In der folgenden Abbildung 18 ist ersichtlich, dass die Inferenzzeiten mit Kühlung rund 20% Prozent schneller gegenüber der selben Inferenz auf dem Vorgängermodell Raspberry Pi 4.

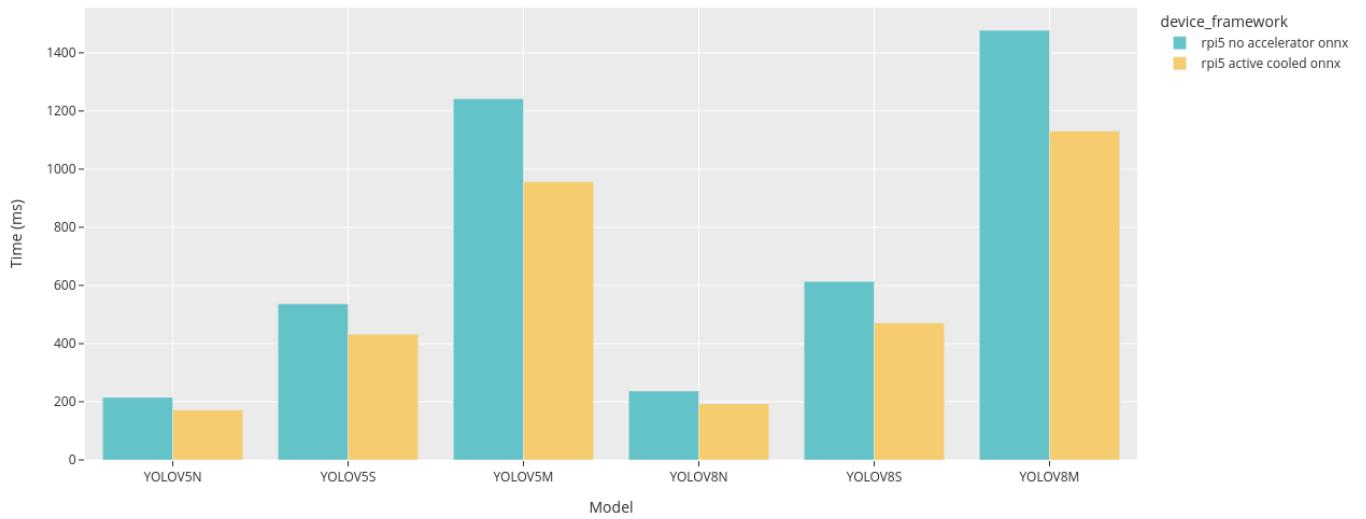


Abbildung 18: Vergleich: Raspberry Pi 5 mit und ohne Kühlung

(Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)

3.4.3.2 Coral Accelerator

Das Toolkit Coral [35] wurde von Google entwickelt und stellt Beschleuniger verschiedener Art zur Verfügung. Im Kontext dieser Arbeit wurden zwei dieser Beschleuniger untersucht: Ein USB Dongle mit einer Edge TPU von 4 TOPS und ein über PCI verbundenes Board mit einer TPU mit 8 TOPS. Bei den Versuchen hat sich herausgestellt, dass der USB Dongle unzuverlässig ist. Es kam während länger laufenden Tests wiederholt zu Verbindungsunterbrüchen. Die Edge TPU über PCI funktionierte zuverlässig.

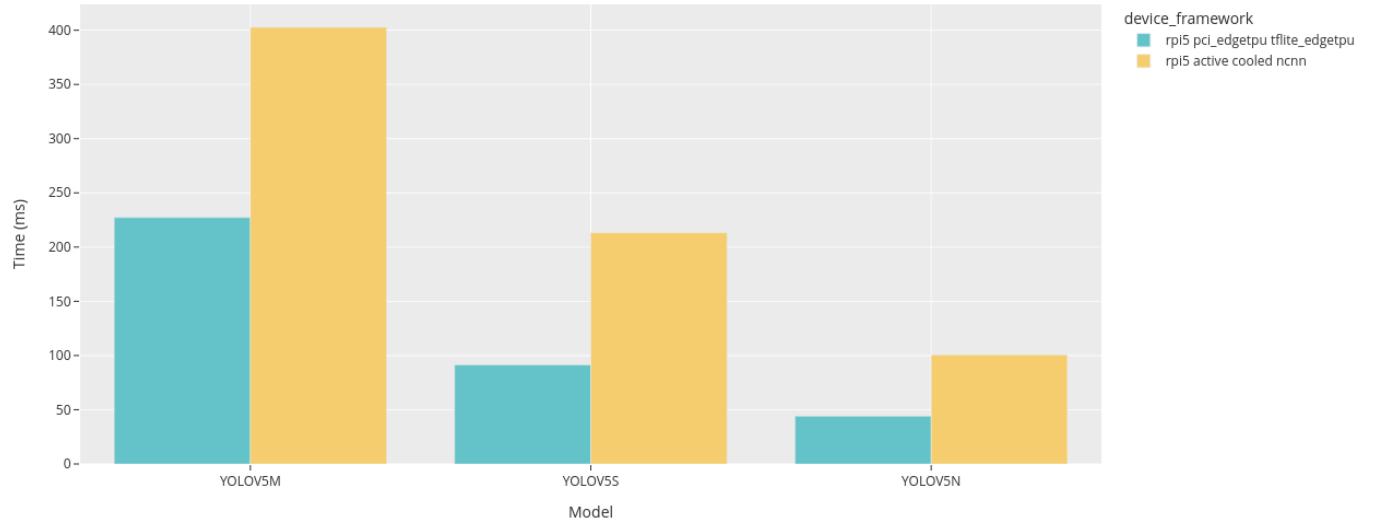


Abbildung 19: Vergleich: Raspberry Pi 5 vs. Coral PCI Edge TPU

Es ist sofort ersichtlich, dass die Inferenz auf der Edge TPU um ein Vielfaches schneller ist als auf der CPU mit NCNN des Raspberry Pi 5. Allerdings muss hier erwähnt werden, dass die EdgeTPU nur quanzierte TensorFlow lite Modelle ausführen kann und somit besteht auch ein Verlust der Präzision.

3.4.3.3 Hailo

In dieser Untersuchung wurden die beiden Modelle Hailo8l (13 TOPS) und Hailo8 (26 TOPS) eingesetzt. Die beiden Module werden jeweils über die PCI Schnittstelle mit dem Raspberry Pi 5 verbunden. Aufgrund der benötigten PCI Schnittstelle kann das Raspberry Pi 4 Modell nicht verwendet werden. Für die Messungen der Inferenzzeit ist das

von Hailo zur Vergütung gesetzte ML-Framework verwendet worden. Zudem müssen die Modelle im Hailo eigenen Format .hef sein, um sie auf den Accelerator auszuführen. Hailo stellt ein Model Zoo [36] zur Verfügung, welcher die Modelle YOLOv5s und YOLOv5m beinhaltet. Die folgende Abbildung 20 zeigt den Vergleich mit der Coral Edge TPU.

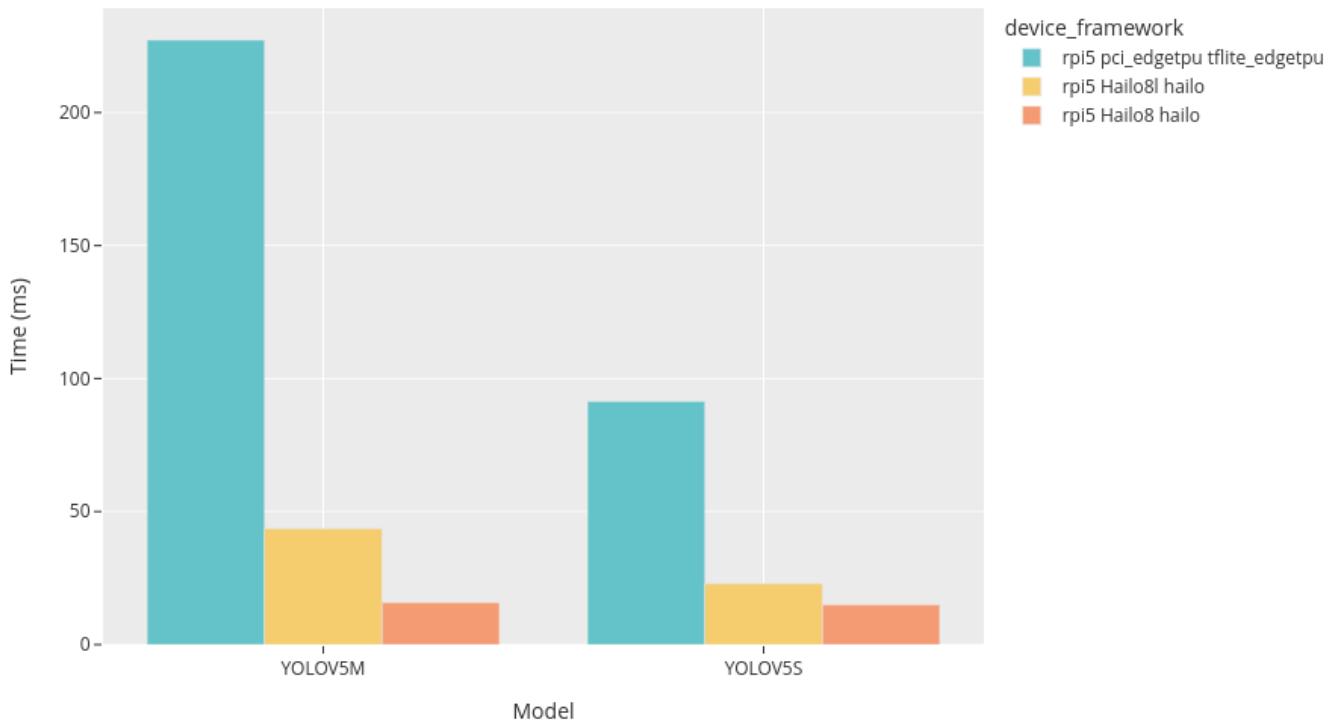


Abbildung 20: Vergleich Hailo Accelerator vs. Coral PCI Edge TPU

(Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)

Die beiden Hailo Accelerator sind signifikant schneller als der schon etwas ältere Coral Accelerator. Ebenso ist ersichtlich, dass sich die doppelte Anzahl TOPS direkt auf die Inferenz Geschwindigkeit auswirkt. Die schnellste Inferenz wird somit mit dem Hailo8 Accelerator erzielt und beträgt run 15ms.

3.5 Edge ML Setups

Aus der Selektion der Hardware und den in diesem Kapitel aufgeführten Evaluationen von Inferenzen lassen sich nun verschiedene Konstellationen mit unterschiedlichen Stärken, resp. Schwächen definieren. Die Abbildung 21 zeigt das Verhältnis von Kosten zu Inferenzgeschwindigkeit der vielversprechendsten Kombinationen mit dem YOLOv8n Model.

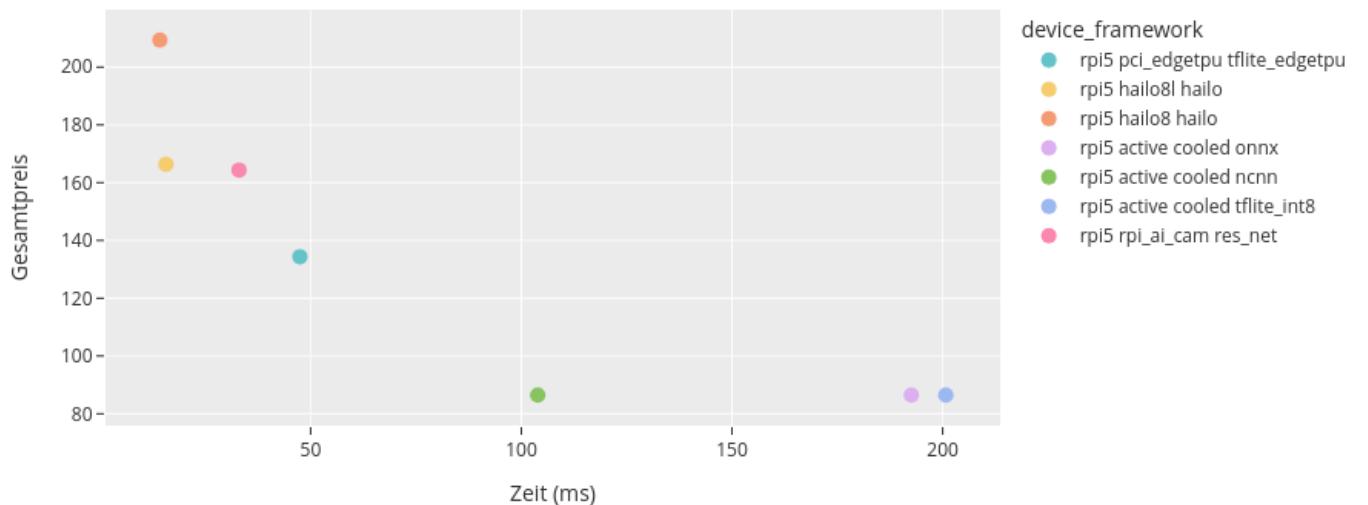


Abbildung 21: Vergleich: Kosten vs. Geschwindigkeit mit YOLOv8n

(Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)

Paper referenzieren

Weil das Raspberry 4 kaum billiger ist als sein Nachfolger Pi 5 ist das Model 4 für diese Bewertung auszuschliessen. Die mit Abstand am Kostengünstigsten Varianten sind jene ohne zusätzliche Beschleuniger-Hardware. Dabei schneidet das NCNN Framework auf dem Raspberry Pi 5 mit ca 9.6 FPS am besten ab gefolgt von ONNX mit rund 5.2 FPS.

Bei den Setups mit Beschleuniger sehen wir, dass mit höheren Ausgaben entsprechend mehr FPS zu erzielen sind. Die schnellsten Inferenzen liefert der teuerste Beschleuniger Hailo8 mit 26 TOPS von Hailo. Der kleinere Hailo Beschleuniger befindet sich auf Stufe der AI-Camera von Raspberry Pi. Etwas abgeschlagen, dafür auch kostengünstiger ist der M.2 PCI Edge TPU von Coral.

Um auf dem Raspberry Pi die beste Performance zu erzielen, empfiehlt sich ein Chip von Hailo. Eine Edge TPU von Google ist aufgrund des abnehmenden Supports und der weniger ausführlichen Dokumentation nicht empfehlenswert.

Lässt es die Applikation zu, ist auch die AI-Kamera von Raspberry Pi eine gute Wahl. Dabei muss berücksichtigt werden, dass in Systemen mit mehr als einer Inferenz nur die erste auf der Kamera durchgeführt werden kann. Darauffolgende Inferenzen müssten auf der CPU oder auf einem weiteren Beschleuniger ausgeführt werden. Ein Setup mit einem weiteren Beschleuniger würde natürlich auch die Kosten erhöhen.

4 Umsetzung

Dieses Kapitel beschreibt die Umsetzung einer Software zum Betreiben einer Machine Learning Pipeline auf einem Edge Device. Das System umfasst die Erfassung von Bildern, ausführen einer Operation jeglicher Art die aus einem Bild Resultate generiert und der Weiterverarbeitung deren Resultate. Die Entwicklung erfolgte drei Iterationen, bei welchen jeweils die erstellte Architektur reflektiert und anhand der gewünschten Qualitäten angepasst wurden. Zuerst ist die System Architektur vorgestellt gefolgt von den wichtigsten Aspekten des Software Designs. Anschliessend folgen verschiedene Prototypen zur Umsetzung konkreter Beispiele.

4.1 System Architektur

Damit das System für Citizen Science geeignet ist, muss es self-contained und flexibel an neue Szenarien anpassbar sein. Dafür sind drei modular austauschbare Komponenten entscheidend: die Bildquelle mit Buffer, die Bildanalyse durch eine Operation und die Verarbeitung der Analysedaten. Besonders die Bildanalyse, die meist eine Machine-Learning-Inferenz umfasst, erfordert individuelle Anpassungen. Da Modelle spezifisches Pre- und Postprocessing benötigen, lässt sich keine allgemeine Implementierung für verschiedene ML-Frameworks realisieren.

4.1.1 Komponenten Diagramm

Die folgende Abbildung 22 zeigt eine übersicht des Systems. Eine Source repräsentiert die Quelle eines Bildes, wie zum Beispiel eine Kamera. Als Buffer zwischen der Analyse und der Source dient eine Queue. Bei der Operation handelt es sich um die Komponente der Analyse. Dies kann eine Machine Learning Inferenz sein oder eine andere Operation, die ein Bild verarbeitet und ein Resultat generiert. Anschliessend gelangt der Output der Operation in eine oder mehrere Sinks. Die Sink ist jener Komponent, welches das Resultat weiterverarbeitet. Dies ist beispielsweise eine Datenbank.

Frames weglassen

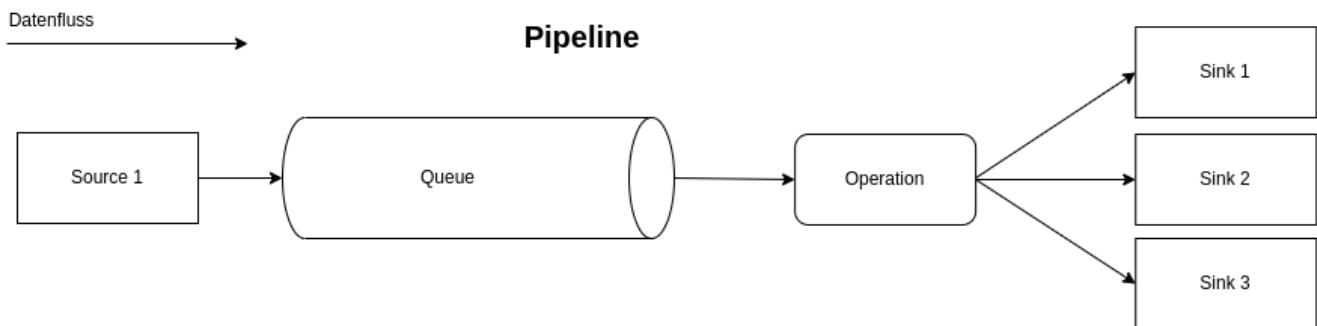


Abbildung 22: Pipeline Komponenten

(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

Ein erwähnenswerter Punkt ist der Buffer für erfasste Bilder. Grundsätzlich ist das System für Biodiversitäts Monitoring ausgelegt, weshalb die Zeit zur Erfassung von Bildern limitiert durch die Lichtverhältnisse ist. Somit ist für die Analyse potentiell mehr Zeit verfügbar als für die Aufnahme der Bilder. Default mässig wird eine In-Memory Queue verwendet. Es ist aber durchaus möglich, dass beispielsweise eine Queue mit Zugriff auf eine externe Festplatte verwendet wird. Diese Implementierung lässt sich durch einhalten der Queue Schnittstelle leicht auf eigene Bedürfnisse anpassen.

4.1.2 Qualitäten

Da die Software für verschiedenste Anwendungen zum Einsatz kommen kann und dafür jeweils auch angepasst oder erweitert werden muss, ist das Hauptaugenmerk auf die folgenden Qualitäts Kriterien gelegt worden:

- Einfachheit: Der Code ist in Python geschrieben, eine Programmiersprache die gerade bei Forschenden hohen Bekanntheitsgrad geniesst. Implementierungen sind jeweils einfach gehalten, so dass sie beim lesen des Codes leicht verständlich und interpretierbar sind.

- Modularisierung: Alle Komponenten die austauschbar sind, sind durch abstrakte Klassen ohne Implementierung der Methoden vorgegeben. Interfaces, wie man es aus anderen Sprachen kennt gibt es in Python nicht.
- Dokumentation: Die Applikation muss je nach Use Case angepasst oder erweitert werden. Dies gelingt besser, wenn der Code gut strukturiert und dokumentiert ist.

4.1.3 Konfiguration

Das System lässt sich mittels Konfigurations File definieren. Die Konfiguration enthält im wesentlichen drei Komponenten, einen für jeden auswechselbaren Teil der Software. Somit lassen sich mehrere Sources, Operations und Sinks definieren. Diese Einzelteile können dann zur Laufzeit mittels Webinterface angepasst werden. Dieser Mechanismus erleichtert das angenehme beobachten und anpassen einer Pipeline. Das folgende Listing 1 zeigt eine Beispiel Konfiguration mit jeweils einer Option für jeden Komponenten Typ.

```

1   sources:
2     - name: webcam
3       file_path: ./source/impl/webcam.py
4       class_name: Webcam
5       parameters:
6         device: "/dev/video0"
7         width: 640
8         height: 640
9   operations:
10    - name: detect coco objects
11      class_name: UlDetect
12      file_path: ./pipe/impl.ulDetect.py
13      parameters:
14        model_path: ./resources/ml_models/yololnn.onnx
15        label_path: ./resources/labels/coco.txt
16        confidence_threshold: 0.5
17        nms_threshold: 0.5
18   sinks:
19     - name: console
20       class_name: Console
21       file_path: ./sink/impl/console.py

```

Listing 1: Beispiel Pipeline Konfiguration

Jede Komponente definiert die folgenden Attribute:

- **name**: Der name des Komponenten, damit man ihn später über das Konfigurations Interface erkennen kann.
- **file_path**: Der Pfad zum File, in der die zu ladende Klasse implementiert ist.
- **class_name**: Die zu ladende Klasse aus dem definierten File. In Python können mehrere Klassen in einem File implementiert sein.
- **parameters** (optional): Ist ein Satz an Parameter, die der Klasse beim initialisieren übergeben werden. Weil diese Parameter sehr Anwendungsspezifisch sind, ist die Struktur dieses Objekts offen gelassen.

Die Applikation lädt alle Komponenten beim Start mithilfe eines **ClassLoader**-Mechanismus und speichert deren Instanzen intern. Durch definierte Namen können die Komponenten zur Laufzeit flexibel ein- oder ausgeschaltet werden. Ein einfaches Anwendungsbeispiel ist das Umschalten zwischen einem Kamera-Stream und einem Testbild.

4.1.4 Webinterface

Die Applikation startet nebst der genannten Komponenten auch ein Webinterface zur Konfiguration des laufenden Systems. Komponenten, welche im Konfigurationsfile definiert sind, lassen sich über die Drop Down Menus einstellen. Die Abbildung 23 zeigt ein Screenshot des Interfaces.

delete Choose in interface

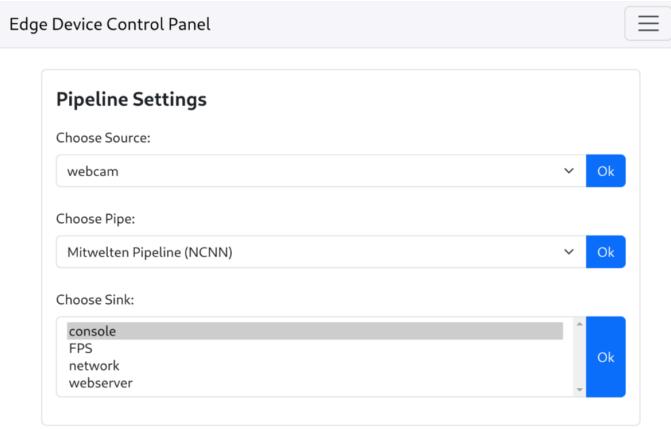


Abbildung 23: Klassen Diagramm

(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

4.2 Software Design

Referenz auf Buch

In diesem Abschnitt wird aufgezeigt, wie die einzelnen Komponenten mit einander kommunizieren. Zu diesem Zweck ist folgend ein Klassen Diagramm mit den relevanten Elementen darstellt. Die zentrale Komponente stellt die Klasse Pipeline dar. Diese hat eine Beziehung zu der Source, Operation und Sink und orchestriert den Datenfluss zwischen den Komponenten. Mittels setter-Methoden auf der Pipeline lassen sich die Austauschbaren Komponenten mit dem Webinterface konfigurieren.

Während der Entwicklung wurde das Design kontinuierlich angepasst um sicherzustellen, dass Abhängigkeiten reduziert werden und somit die Einfachheit gefördert wird. Gleichzeitig wurde versucht, sich auf die Kern Funktionalitäten des Systems zu beschränken und Feature mit wenig Funktionalität zu beseitigen.

Kardinalität im Diagramm

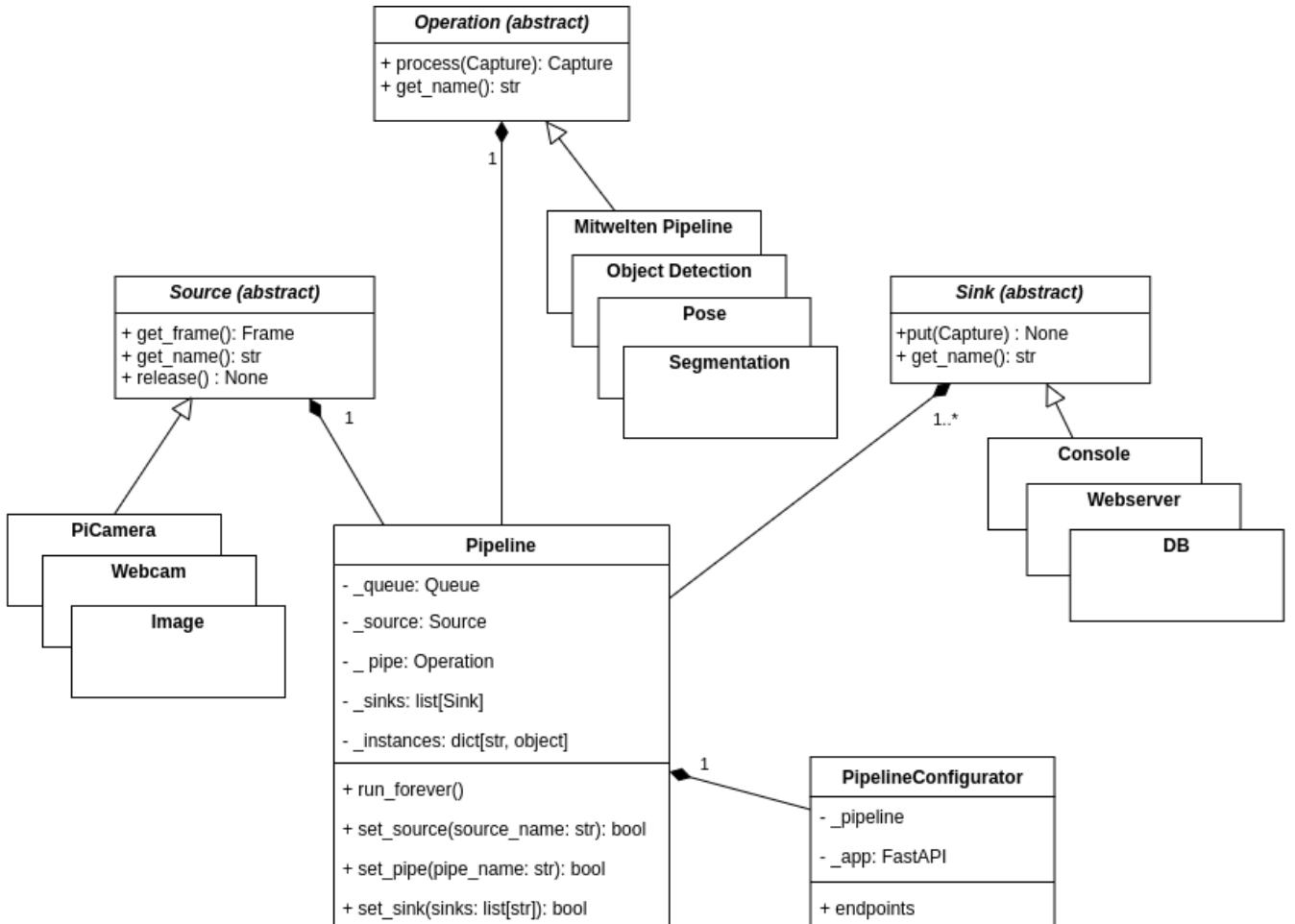


Abbildung 24: Klassen Diagramm

(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

Um die zeitlichen Abläufe besser dazustellen ist folgend ein vereinfachtes Sequenz Diagramm dargestellt. Die Pipeline ist dafür verantwortlich, dass ein Thread zur Erfassung von Bildern gestartet wird. Diese Bilder werden in eine Thread-Safe Queue zwischengespeichert. Im Haupt-Thread der Applikation holt die Pipeline ein Bild von Queue ab und führt die Operation aus, wobei es sich zum Beispiel um die Mitwelten Analyse handelt. Anschliessend übergibt die Pipeline die Resultate einer oder mehreren Sinks. Es kann mehrere Sinks geben, weil die Möglichkeit bestehen soll, Resultate abzuspeichern und gleichzeitig auf der Konsole oder auf einem Web Interface visuell darzustellen.

Mehrzahl Sinks, Thread1 -> Thread

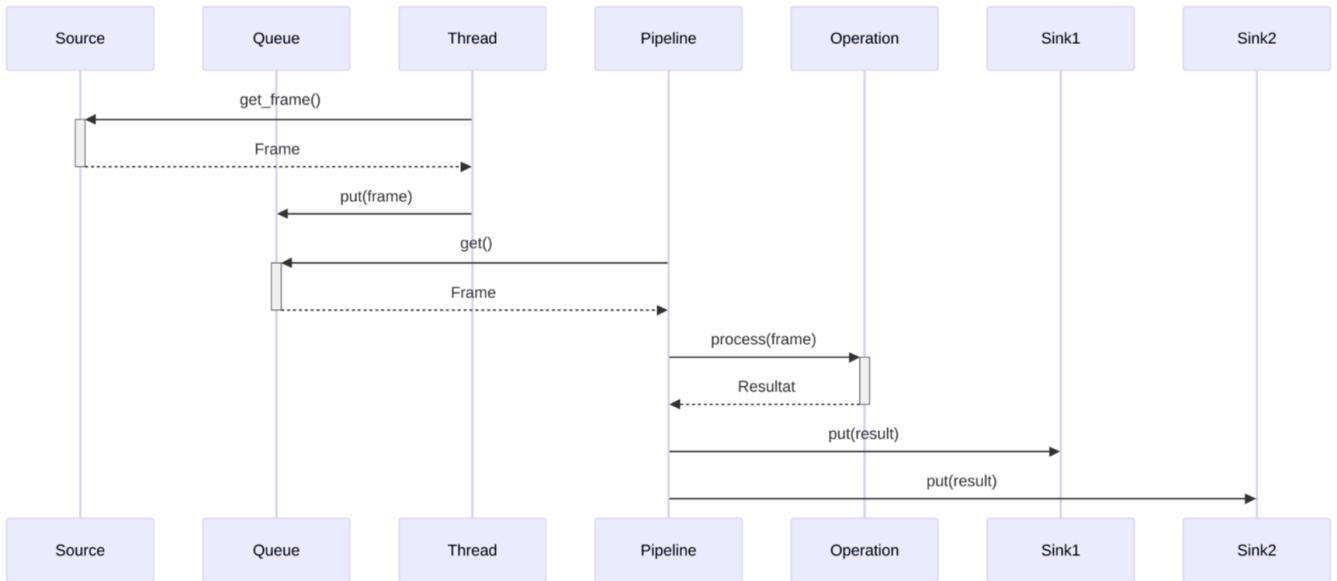


Abbildung 25: Pipeline Komponenten
(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

4.3 Prototypen: verschiedene System Setups

In diesem Kapitel werden zwei unterschiedliche Prototypen vorgestellt, die verschiedene System-Setups für die Anwendung der entwickelten Applikation demonstrieren. Beide Prototypen setzen Machine Learning auf dem Raspberry Pi 5 ein, unterscheiden sich jedoch in ihrer verwendeten Hardware und Konfiguration. Durch den Vergleich dieser Setups lassen sich die Vor- und Nachteile verschiedener Ansätze analysieren und bewerten.

4.3.1 Prototyp 1 - Mitwelten Analyse

Dieser Prototyp setzt den Referenz Use Case aus Abschnitt 2.4.1 mit verschiedenen Modellen um. Die Analysen wurden jeweils mit Testbildern durchgeführt, daher ist der Energieverbrauch ohne Kamera zu betrachten. Die folgenden Abbildung 26 zeigt die Beziehung zwischen Anzahl Blumen auf einem Bild und der Zeit für die Bestäuber Analyse. Die angegebene Zeit beinhaltet die Detektierung der Blüten sowie die Detektierung von Bestäuber auf jeder detektierten Blüte.

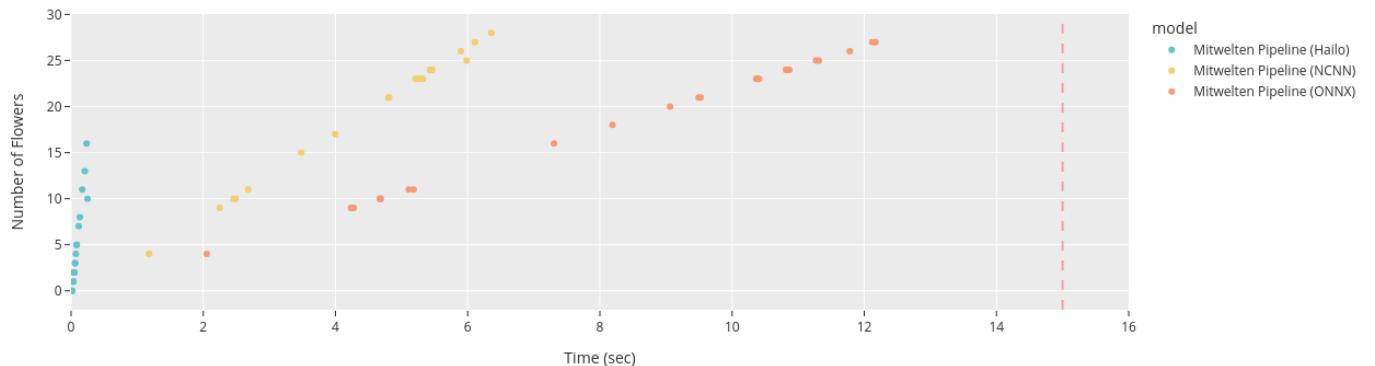


Abbildung 26: Mitwelten Analyse - Framework Vergleich
(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

Aus Abbildung 26 geht hervor, dass alle drei Pipelines die zeitlichen Anforderungen erfüllen. Dabei ist auch ersichtlich, dass sich die Ergebnisse aus der Analyse in Kapitel 3 wiederspiegeln.

4.3.1.1 ONNX

Die Analyse hat gezeigt, dass die Ausführung der Modelle im ONNX Format schon eine beträchtliche Zeiteinsparung pro Inferenz zur Folge hat. Zu diesem Zweck sind die Mitwelten Modelle mittels Ultralytics export Funktion in das

ONNX Format konvertiert worden. Aus Abbildung 26 ist ersichtlich, dass der Zeitbedarf der Mitwelten Analyse im Bereich des akzeptierbaren liegt. Der Energieverbrauch liegt bei rund 10W. Durch den linearen Zusammenhang zwischen Anzahl Blumen und Inferenzzeit können wir die maximale Anzahl Blumen ermitteln, welche innerhalb der 15 Sekunden berechenbar sind. Mit den Datenpunkten: 4 Blumen, 2.051s und 27 Blumen, 12.168s lässt sich für 15 Sekunden rund 33 Blumen analysieren.

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{27 - 4}{12.168s - 2.051s} = 2.275$$

$$b = y_1 - (m * x_1) = 4 - (2.275 * 2.051s) = -0.66$$

$$y = m * x + b = 2.275 * 15 - 0.66 = 33.465$$

4.3.1.2 NCNN

Aus Kapitel 3 ist zu erwarten, dass die Umsetzung der Mitwelten Analyse mit NCNN Modellen nochmals deutlich schneller wird. Dies zeigt auch die Umsetzung dieses Prototyp. Der Zeitbedarf entspricht etwa der Hälfte von der Umsetzung mit ONNX Modellen, während der Energiebedarf mit rund 10W etwa gleich bleibt.

Die Konvertierung des Bestäuber Modells in das NCNN Format verursachte einige Schwierigkeiten. Folglich wurde das Yolov8n als alternative trainiert und verwendet. Die Yolov8 Generation ist eine Weiterentwicklung der Yolov5 Modelle. Das Yolov8n (nano) Modell ist von der Geschwindigkeit vergleichbar mit dem Yolov5s (small), welches für die Bestäuberanalyse verwendet wurde [37]. Allerdings soll an dieser Stelle erwähnt sein, dass das ursprüngliche Model mit einer grösse von 480x480 trainiert worden ist. Dies wurde bei der adaptierung auf Yolov8 nicht berücksichtigt, somit dürfte sich durch das leicht kleinere Modell die Inferenz Dauer in einem ähnlichen Bereich bewegen. Die Berechnung, wie viele Blüten in 15 Sekunden analysiert werden könnten ergab in diesem Fall durch Einsetzen der Datenpunkte: 4 Blüten in 1.179 Sekunden und 27 Blüten in 6.11 Sekunden, erhalten wir das Resultat von 68.47 Blüten, welche in 15 Sekunden analysiert werden könnten.

4.3.1.3 Hailo

Eine weitere Implementierung wurde mit der Beschleuniger Hardware von Hailo umgesetzt. Um Modelle auf dieser Hardware auszuführen, müssen diese in das von Hailo definierte hef Format konvertiert werden. Dies erreicht man mit der vom Hersteller zur Verfügung gestellten Software Suite [38]. Diese beinhaltet im wesentlichen einen parser, optimierer und compiler. Diese Prozesse müssen für jedes Modell richtig konfiguriert werden. Dafür ist die Untersuchung des zu konvertierenden Modells unerlässlich. Der folgende Befehl in Listing 2 zeigt beispielhaft, wie ein komplizierungs Prozess ausgeführt werden kann.

```
1 hailomz compile \
2   --hw-arch hailo8l \
3   --yaml ./hailo_model_zoo/hailo_model_zoo/cfg/networks/yolov8n.yaml \
4   --ckpt /local/shared_with_docker/yolov8n_pollinator_ep50_v1.onnx \
5   --classes 5 \
6   --end-node-names /model.22/cv2.0/cv2.0.2/Conv /model.22/cv3.0/cv3.0.2/Conv /model.22/cv2.1/cv2.1.2/Conv /model.22/cv3.1/cv3.1.2/Conv /model.22/cv2.2/cv2.2.2/
7   Conv /model.22/cv3.2/cv3.2.2/Conv \
8   --calib-path /local/shared_with_docker/pollinators/
```

Listing 2: Hailo Konvertierung

Speziell für die end-node-names muss das Model mit einem Tool wie netron.app untersucht werden damit diese gefunden werden können. Ebenso können Anpassungen im yaml file, welches mit dem Flag --yaml angegeben wird nötig sein. Dieses yaml file definiert weitere Parameter zur Beschreibung des Modells. Zusätzlich wird ein .alls file geladen, welches verwendet wird, um den Output Node richtig zu konfigurieren. Auch dieses File verweist noch auf ein postprocessing config file, welches die end-nodes nach dem parsing Prozess von Hailo definiert. Diese müssen je nach Modell auch angepasst werden. Die verschiedenen Konfigurationsfiles werden von Hailo in ihrem eigenen Model Zoo zur Verfügung gestellt [39]. Um die Nodes nach dem parsing zu finden stellt Hailo ein Profiler zur Verfügung.

Folgende Abbildung 27 zeigt ein Screenshot des Hailo Profiler Interfaces. Der Ausschnitt zeigt einige Nodes eines Yolov8n Modells.

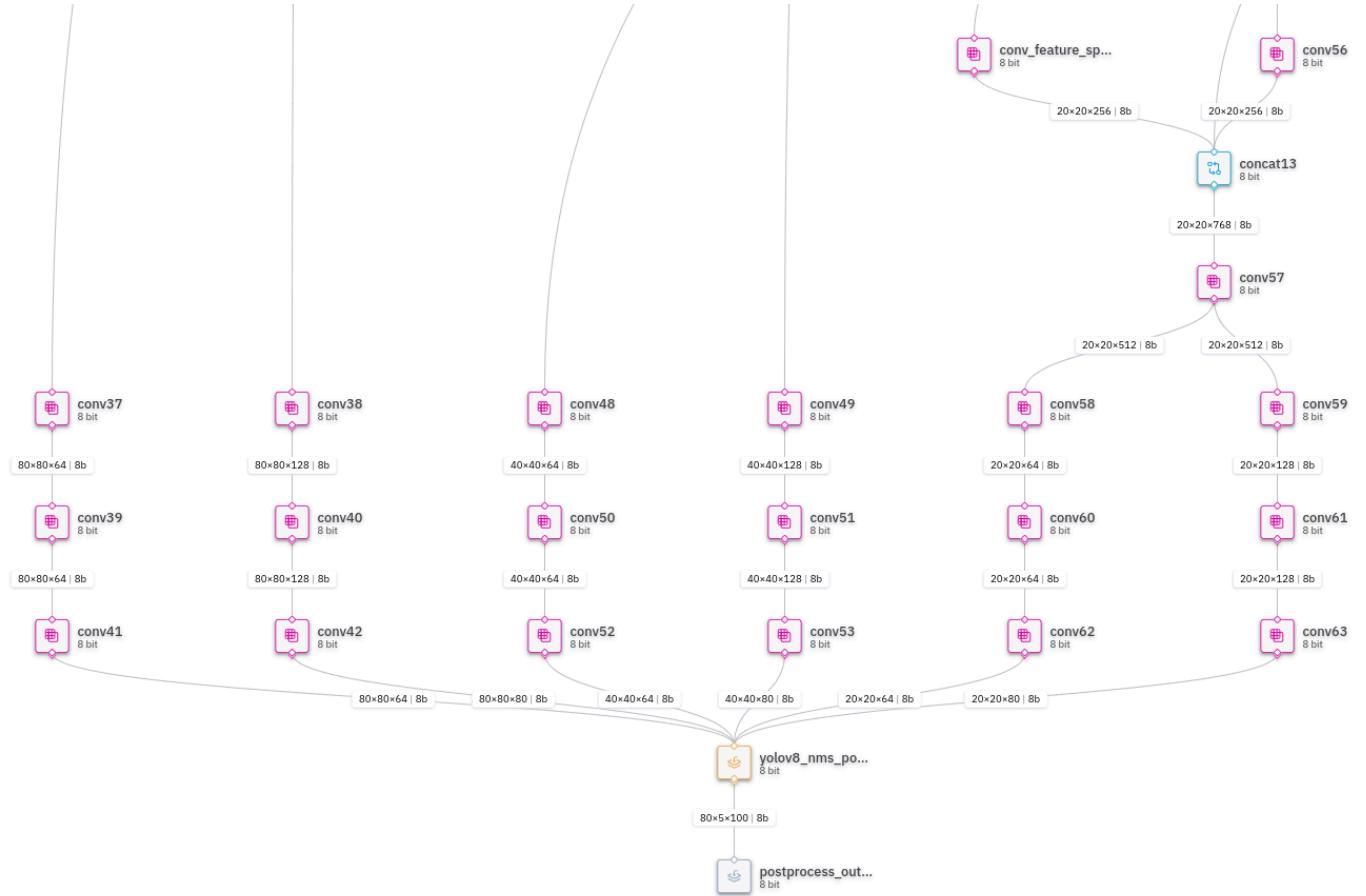


Abbildung 27: Teil eines Hailo Model Graph Yolov8n

(Quelle: Screenshot Hailo Profiler, Andri Wild, 2025)

Weil dieser Prozess für die bestehenden Yolov5 Modelle nicht erfolgreich war, kamen an dieser Stelle neu trainierte Yolov8 Modelle zum Einsatz. Das Trainig der Modelle lag an dieser Stelle nicht im Fokus, daher weisen diese nicht die gleiche Präzision auf, wie die Yolov5 Modelle. Dies ist auch der Grund, dass die Abbildung 26 weniger Blüten bei der Hailo Pipeline hat. Die Analyse der Inferenzzeiten für die Beustäuber Analyse zeigt beeindruckende Resultate. Auch mit vielen Blüten ist die Performance sehr gut. Die Gesamtzeit der Inferenz von 13 Blüten dauert rund 20ms. Des weiteren ist auch der Energiebedarf bei rund 5W, was der Hälfte der CPU betriebenen Pipelines entspricht.

4.3.2 Prototyp 2 - AI-Camera

Der zweite Prototyp verwendet die AI-Camera. Um eigens trainierte Modelle auf der AI-Camera auszuführen, müssen diese für den Sony IMX500 Chip konvertiert werden [40]. Dieser Prozess ist nicht trivial und konnte im Kontext dieser Arbeit nicht mehr erarbeitet werden. Trotzdem soll an dieser Stelle ein Use Case mit dieser Hardware vorgestellt werden. Eine besondere Eigenschaft der AI-Camera ist, dass diese zugleich mit dem Bild die Inferenz Resultate liefert. Dies hat zur Folge, dass das Host System praktisch unbelastet ist. Dies ermöglicht das abarbeiten von anderen Tasks durch die freie Rechenleistung oder das Verwenden von weniger Leistungsfähiger Hardware.

Erwähnenswert ist in diesem Setup die Konfiguration. Das Listing 3 zeigt, dass die AI-Kamera sowohl als **source** als auch als **pipe** definiert ist.

```

1   sources:
2     - name: AI-Camera
3       file_path: ./source/impl/aiCamera.py
4       class_name: AiCamera
5   parameters:
6     width: 640
7     height: 640
8   pipes:
9     - name: AI-Camera
10    file_path: ./source/impl/aiCamera.py
11    class_name: AiCamera
12 sinks:
13   - name: console
14     class_name: Console
15     file_path: ./sink/impl/console.py

```

Listing 3: Konfiguration AI-Kamera

Durch die Modularität der Applikation erhält hier die AI-Kamera eine Doppelrolle. Effektiv existiert in der Applikation nur eine Instanz der Klasse `AiCamera`. Die Applikation sorgt beim Laden der Klassen dafür, dass dies sichergestellt ist.

Cache, Singleton nicht möglich

4.4 Adaptierung des Systems

Die Installation der Anwendung ist im `README.md` des Github Repository ausführlich beschrieben und dient als Anleitung für die Inbetriebnahme. In der Anleitung werden auch die wichtigsten Schritte zur Erweiterung des Systems aufgeführt. Zudem bestehen für die auswechselbaren Komponenten Beispielimplementierungen.

4.5 Lizenzierung

Der Beispielcode für die Inferenz nutzt die Bibliothek Ultralytics, die unter der GNU AGPL-3.0 lizenziert ist. Deshalb unterliegt der entsprechende Codeabschnitt den Vorgaben der AGPL, was insbesondere bedeutet, dass Änderungen an diesem Teil ebenfalls unter der AGPL offengelegt werden müssen. Der übrige Teil der Applikation ist von diesen Einschränkungen nicht betroffen und kann frei verwendet werden. Soll die Anwendung nicht als Open-Source Projekt veröffentlicht werden, muss sichergestellt sein, dass die entsprechenden Source Codes von Ultralytics gelöscht werden.

5 Validierung

Der in Abschnitt 4.3.1 vorgestellte Prototyp zeigt, dass die Mitwelten-Bestäuber-Analyse auch auf einem Edge-Device realisierbar ist. Eine solche Umsetzung war mit Hardware im gleichen Preissegment bislang noch nicht möglich.

Die Entwicklung einer Applikation für unterschiedliche Einsatzbereiche hat deutlich gemacht, dass sich im Bereich der Edge-AI-Hardware und Software einiges bewegt. Dank der Weiterentwicklung spezialisierter Machine-Learning-Hardware können immer mehr Anwendungen direkt an der Edge realisiert werden, sodass die Leistungsfähigkeit moderner Edge-Computer zunehmend weniger als limitierender Faktor wirkt.

5.1 Versuchsaufbau und Temperaturrentwicklung

Besonders hervorzuheben ist der Einsatz von Beschleunigern, wie etwa den Komponenten von Hailo, die nicht nur effizienter arbeiten, sondern auch die Wärmeentwicklung signifikant reduzieren. Ein einfacher Versuchsaufbau auf Abbildung 28 illustriert den Unterschied zur reinen CPU-Inferenz: Während das Gehäuse bei Einsatz der Hailo-Inferenz konstant auf einem niedrigen Temperaturniveau bleibt, führt die CPU-Inferenz zu einem rapiden Temperaturanstieg.

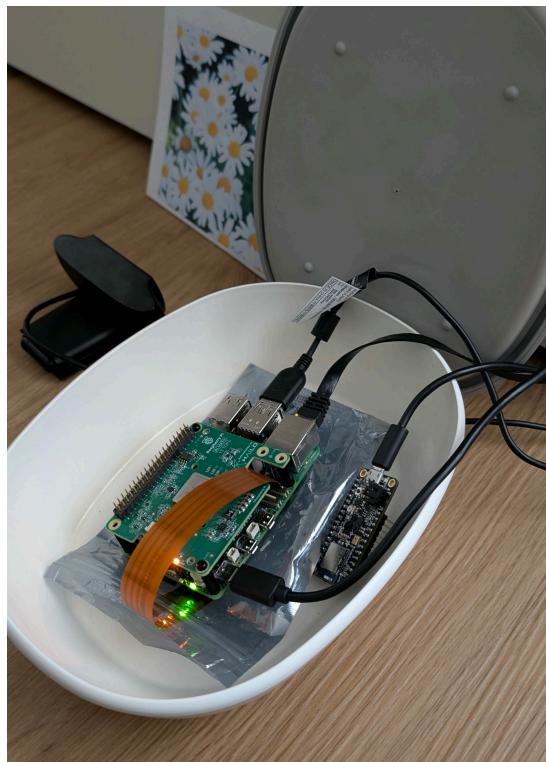


Abbildung 28: Versuchsaufbau Temperatur Messung

(Quelle: Eigene Aufnahme, Andri Wild, 2025)

Mit der CPU betriebenen Inferenz erhöhte sich die Temperatur im geschlossenen Gehäuse innerhalb einer Stunde auf 40 Grad, steigend. Die CPU Temperatur lag bei ca. 85 Grad, womit die Drosselung aktiviert wird und die Rechenleistung sinkt.

Diese Eigenschaft hat positive Auswirkungen auf das Gehäusedesign – insbesondere bei Anwendungen wie Biodiversitätsanalysen, bei denen das Gerät oft im Freien und in wasserdichten Gehäusen eingesetzt wird. Der Wegfall von aktiver Belüftung eröffnet hier wesentliche Konstruktionsvorteile.

Ein Punkt beim Umgang mit den Hailo Beschleuniger, welcher nochmals untersucht werden muss ist die Stabilität in Betrieb. Während länger laufenden Betriebstest kam es mehrfach zu Segmentation fault. Dieser Umstand ist zum Zeitpunkt dieser Arbeit noch nicht geklärt und bedarf an weiteren Untersuchungen.

5.2 Konkurrenz Produkte

Die Möglichkeit, Analysen direkt auf einem Edge-Device auszuführen, ist eine vielversprechende Idee, was durch die Verfügbarkeit von Produkten auf dem Markt bestätigt wird. Zwei besonders interessante Geräte sind die reCamera [41] und die EcoEye-Kamera [42], die beide über SeeedStudio erhältlich sind.

Die reCamera lässt sich mithilfe von Node-Red konfigurieren, wodurch die Konfiguration von Applikationen ohne Programmierkenntnisse möglich ist. Dies bietet insbesondere für Projekte ohne informatikaffine Mitarbeitende oder für Citizen-Science-Anwendungen einen erheblichen Vorteil. Die Kamera ist äusserst kompakt, jedoch nicht wasserdicht und erfordert daher einen zusätzlichen Schutz für den Ausseneinsatz.

Die EcoEye-Kamera wurde speziell für den Ausseneinsatz entwickelt und verfügt über einen grossen Akku, der den Betrieb in Gebieten ohne externe Stromversorgung ermöglicht. Ihre Bauweise ähnelt der einer herkömmlichen Wildkamera, bietet jedoch zusätzliche Rechenleistung, um Analysen direkt auf dem Gerät durchzuführen. Dadurch können Daten bereits vor Ort verarbeitet werden, was den Bedarf an externer Infrastruktur reduziert.

6 Fazit

Ausblick, was kann man besser machen: Gehäuse, Tests, User Tests

Die Inferenz Zeiten beschleunigt mithilfe von Hailo Komponenten sind für die Bestäuber Analyse kürzer als Verlangt. Daher Abschliessend lässt sich sagen, dass die entwickelte Edge ML Kamera grosses Potenzial für Citizen Science-Projekte bietet, da sie den Zugang zu modernen Machine-Learning-Methoden auch ausserhalb zentraler Infrastrukturen ermöglicht. In dieser Arbeit wurde gezeigt, wie der gezielte Einsatz optimierter ML-Frameworks und spezialisierter Hardware – etwa moderner Hailo-Beschleuniger, NCNN-optimierter Modelle und des Raspberry Pi 5 – zu deutlich schnelleren Inferenzzeiten und einem geringeren Energieverbrauch führt.

Dank dieser schnellen Inferenzzeiten können zudem grössere und präzisere Modelle trainiert werden, die trotzdem die Zeitvorgaben erfüllen. Ausserdem legt die Analyse nahe, dass es prinzipiell möglich wäre, mehr als einen Kamerastream gleichzeitig auf einem einzigen Gerät auszuwerten. Dies wäre besonders in abgelegenen oder ressourcenbegrenzten Umgebungen von Vorteil, da so mehrere Datenquellen parallel verarbeitet werden könnten.

Abbildungsverzeichnis

Abbildung 1: Erweiterte thematische Karte zum Lernen von Freiwilligen (Quelle: The Science of Citizen Science, S.300)	9
Abbildung 2: (Quelle: https://docs.ultralytics.com/integrations/onnx/ , aufgerufen am 02.02.2025)	11
Abbildung 3: Machine Learning Tasks (Quelle: https://github.com/sony/model_optimization , aufgerufen am 20.01.2025)	12
Abbildung 4: Symbolbild Neuronales Netzwerk (Quelle: https://lamarr-institute.org/blog/deep-neural-networks , abgerufen am 23.01.2025)	13
Abbildung 5: Raspberry Pi 5 (Quelle: https://www.pi-shop.ch/raspberry-pi-5-16gb-ram?src=raspberrypi , aufgerufen am 23.01.2025)	15
Abbildung 6: Coral USB Accelerator (Quelle: https://www.coral.ai/products/accelerator , aufgerufen am 22.01.2025)	16
Abbildung 7: Coral PCI Accelerator (Quelle: https://www.coral.ai/products/m2-accelerator-dual-edgetpu , aufgerufen am 22.01.2025)	16
Abbildung 8: Hailo Accelerator (Quelle: https://www.pi-shop.ch/raspberry-pi-ai-hat-13t , aufgerufen am 22.01.2025)	16
Abbildung 9: Raspberry Pi AI Camera (Quelle: https://www.berrybase.ch/raspberry-pi-ai-camera , aufgerufen am 22.01.25)	17
Abbildung 10: Raspberry Pi AI Camera Architektur (Quelle: https://www.raspberrypi.com/documentation/accessories/ai-camera.html , aufgerufen am 20.01.2025)	17
Abbildung 11: IoT Kamera im Feld (Quelle: mitwelten.ch , aufgerufen am 23.01.2025)	18
Abbildung 12: Vorgang Mitwelten Bestäuber Analyse (Quelle: Automated Analysis for Urban Biodiversity Monitoring, Timeo Wullschleger)	19
Abbildung 13: Mitwelten Analyse auf Raspberry Pi 3 (Quelle: Automated Analysis for Urban Biodiversity Monitoring S.61, Timeo Wullschleger)	19
Abbildung 14: ML Exploration Dashboard (Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)	21
Abbildung 15: Vergleich: Inferenzzeit YOLOv5, v8 auf Raspberry Pi 5	22
Abbildung 16: Vergleich: ONNX und NCNN auf Raspberry Pi 5	23
Abbildung 17: Vergleich: Raspberry Pi 4 vs. Pi 5 (Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)	23
Abbildung 18: Vergleich: Raspberry Pi 5 mit und ohne Kühlung (Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)	24
Abbildung 19: Vergleich: Raspberry Pi 5 vs. Coral PCI Edge TPU	24
Abbildung 20: Vergleich Hailo Accelerator vs. Coral PCI Edge TPU (Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)	25
Abbildung 21: Vergleich: Kosten vs. Geschwindigkeit mit YOLOv8n (Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	25
Abbildung 22: Pipeline Komponenten (Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	27
Abbildung 23: Klassen Diagramm	

(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	29
Abbildung 24: Klassen Diagramm	
(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	30
Abbildung 25: Pipeline Komponenten	
(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	31
Abbildung 26: Mitwelten Analyse - Framework Vergleich	
(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	31
Abbildung 27: Teil eines Hailo Model Graph Yolov8n	
(Quelle: Screenshot Hailo Profiler, Andri Wild, 2025)	33
Abbildung 28: Versuchsaufbau Temperatur Messung	
(Quelle: Eigene Aufnahme, Andri Wild, 2025)	35

Tabellenverzeichnis

Appendix A: Supplementary Material

– Supplementary Material –

Glossar

Begriff	Definition
AI	Abkürzung für Artificial Intelligence (Künstliche Intelligenz).
Backend	Serverseitiger Teil eines Systems, der Logik, Datenhaltung und Verarbeitung übernimmt.
Cloud	Netzwerk von Remote-Servern, das zur Datenspeicherung und -verarbeitung über das Internet genutzt wird.
Deep Learning	Teilgebiet des Machine Learning, das auf künstlichen neuronalen Netzen mit vielen Schichten basiert.
Edge	Der Rand eines Netzwerks, an dem Daten erzeugt und häufig lokal verarbeitet werden.
Edge ML	Ausführung von Machine Learning direkt am Netzwerkrand zur schnellen und lokalen Entscheidungsfindung.
IoT - Internet of Things	Netzwerk physischer Objekte, die mittels Sensoren vernetzt sind, um Daten auszutauschen.
ML	Abkürzung für Machine Learning, ein Bereich der KI, der auf der Erkennung von Mustern in Daten basiert.
ML Edge Kamera	Kamera, die Machine Learning direkt an der Edge ausführt, um Echtzeit-Analysen vor Ort zu ermöglichen.

Bibliographie

- [1] A. Bonn *u. a.*, „Grünbuch Citizen Science Strategie 2020 für Deutschland“, Berlin, 2016.
- [2] K. Vohland *u. a.*, Hrsg., „The Science of Citizen Science“. Springer International Publishing, Cham, 2021. doi: 10.1007/978-3-030-58278-4.
- [3] „TensorFlow“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://www.tensorflow.org/>
- [4] „PyTorch“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://pytorch.org/>
- [5] „OpenAI standardizes on PyTorch“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://openai.com/index/openai-pytorch/>
- [6] „ONNX | Home“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://onnx.ai/>
- [7] Ultralytics, „ONNX“. Zugegriffen: 2. Februar 2025. [Online]. Verfügbar unter: <https://docs.ultralytics.com/integrations/onnx>
- [8] H. Ni und The ncnn contributors, „ncnn“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://github.com/Tencent/ncnn>
- [9] „Das führende KI-Chip-Unternehmen für Edge-Geräte“. Zugegriffen: 27. Januar 2025. [Online]. Verfügbar unter: <https://hailo.ai/de/company-overview/>
- [10] Ultralytics, „Home“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://docs.ultralytics.com/>
- [11] Ultralytics, „YOLOv5“. Zugegriffen: 23. Januar 2025. [Online]. Verfügbar unter: <https://docs.ultralytics.com/models/yolov5>
- [12] „Sparsification: Compressing Neural Networks | Neural Magic Documentation“. Zugegriffen: 4. Februar 2025. [Online]. Verfügbar unter: <https://docs.neuralmagic.com/guides/sparsification/>
- [13] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, und A. Y. Zomaya, „Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence“, *IEEE Internet of Things Journal*, Bd. 7, Nr. 8, S. 7457–7469, Aug. 2020, doi: 10.1109/JIOT.2020.2984887.
- [14] „Raspberry Pi 4 Model B - 8GB“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.pi-shop.ch/raspberry-pi-4-model-b-8gb>
- [15] „Raspberry Pi 5 Model B 8GB“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.pi-shop.ch/raspberry-pi-5-8-gb>
- [16] „BeagleY®-AI“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.seeedstudio.com/BeagleYr-AI-beagleboard-orgr-4-TOPS-AI-Acceleration-powered-by-TI-AM67A.html>
- [17] „TPU (Tensor Processing Unit)“. Zugegriffen: 23. Januar 2025. [Online]. Verfügbar unter: <https://www.ultralytics.com/glossary/tpu-tensor-processing-unit>
- [18] „What is a Neural Processing Unit (NPU)? | IBM“. Zugegriffen: 23. Januar 2025. [Online]. Verfügbar unter: <https://www.ibm.com/think/topics/neural-processing-unit>
- [19] „NPU vs GPU: What's the Difference? | IBM“. Zugegriffen: 23. Januar 2025. [Online]. Verfügbar unter: <https://www.ibm.com/think/topics/npu-vs-gpu>
- [20] „Coral USB Accelerator“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.pi-shop.ch/coral-usb-accelerator>
- [21] „Pineboards Hat AI! Dual, Edge Coral TPU für Raspberry Pi 5, Bundle - kaufen bei BerryBase“ Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.berrybase.ch/pineboards-hat-ai-dual-edge-coral-tpu-fuer-raspberry-pi-5-bundle>

- [22] „Pineboards Hat AI!, Coral Edge TPU Erweiterung für Raspberry Pi 5, Bundle - kaufen bei BerryBase“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.berrybase.ch/pineboards-hat-ai-coral-edge-tpu-erweiterung-fuer-raspberry-pi-5-bundle>
- [23] „Raspberry Pi AI HAT+ (13T)“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.pi-shop.ch/raspberry-pi-ai-hat-13t>
- [24] „Raspberry Pi AI HAT+ (26T)“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.pi-shop.ch/raspberry-pi-ai-hat-26t>
- [25] „Raspberry Pi AI Camera - kaufen bei BerryBase“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.berrybase.ch/raspberry-pi-ai-camera>
- [26] „AI Camera - Raspberry Pi Documentation“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.raspberrypi.com/documentation/accessories/ai-camera.html>
- [27] bdm, „Verlässliche Daten über unsere Lebensgrundlage“. Zugegriffen: 15. Januar 2025. [Online]. Verfügbar unter: <https://www.biodiversitymonitoring.ch/index.php/de/>
- [28] T. Wullschleger, „Data Acquisition for Urban Biodiversity Monitoring“.
- [29] T. Wullschleger, „Automated Analysis for Urban Biodiversity Monitoring“.
- [30] „ip7-ml-model-eval/dashboard at main · andriwild/ip7-ml-model-eval“. Zugegriffen: 2. Februar 2025. [Online]. Verfügbar unter: <https://github.com/andriwild/ip7-ml-model-eval/tree/main/dashboard>
- [31] awild, „andriwild/ip7-ml-model-eval“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://github.com/andriwild/ip7-ml-model-eval>
- [32] G. Jocher, A. Chaurasia, und J. Qiu, „Ultralytics YOLOv8“. [Online]. Verfügbar unter: <https://github.com/ultralytics/ultralytics>
- [33] T.-Y. Lin *u. a.*, „Microsoft COCO: Common Objects in Context“. [Online]. Verfügbar unter: <https://arxiv.org/abs/1405.0312>
- [34] M. Hussain, „YOLOv5, YOLOv8 and YOLOv10: The Go-To Detectors for Real-time Vision“. Zugegriffen: 4. Februar 2025. [Online]. Verfügbar unter: <http://arxiv.org/abs/2407.02988>
- [35] „Products“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://coral.ai/products/#prototyping-products>
- [36] T. Tapuhi *u. a.*, „Hailo Model Zoo“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: https://github.com/hailo-ai/hailo_model_zoo
- [37] Ultralytics, „YOLOv5 vs YOLOv8: A Detailed Comparison for Object Detection“. Zugegriffen: 29. Januar 2025. [Online]. Verfügbar unter: <https://docs.ultralytics.com/compare/yolov5-vs-yolov8>
- [38] „Software Suite for AI Applications & Deep Learning | Hailo AI“. Zugegriffen: 29. Januar 2025. [Online]. Verfügbar unter: <https://hailo.ai/products/hailo-software/hailo-ai-software-suite/>
- [39] „hailo_model_zoo/hailo_model_zoo/cfg at master · hailo-ai/hailo_model_zoo“. Zugegriffen: 29. Januar 2025. [Online]. Verfügbar unter: https://github.com/hailo-ai/hailo_model_zoo/tree/master/hailo_model_zoo/cfg
- [40] „IMX500 Converter | Sony Semiconductor Solutions Group“. Zugegriffen: 29. Januar 2025. [Online]. Verfügbar unter: <https://developer.aitrios.sony-semicon.com/en/raspberrypi-ai-camera/documentation/imx500-converter?version=3.14.3&progLang=>
- [41] „reCamera“. Zugegriffen: 1. Februar 2025. [Online]. Verfügbar unter: https://www.seeedstudio.com/recamera?srsltid=AfmBOopbSxSK-BPmVHDRVm0pGJWa_aRpSmQuWH0XmIjh2ARMrHknYQyG
- [42] „EcoEye: OpenMV H7+ Camera with Sensor Integration for Environmental Monitoring“. Zugegriffen: 1. Februar 2025. [Online]. Verfügbar unter: <https://www.seeedstudio.com/EcoEye-Embedded-Vision-Camera-p-5843.html>