# Framework für Embedded/Edge ML - Optimiertes Modell Deployment auf Embedded Systemen in der Fertigung

Sebastian Munoz Segovia

29. Oktober 2024

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1

# Einleitung

In der heutigen Industrie 4.0 spielen **Embedded Systems und Edge-Computing** eine zentrale Rolle in der Automatisierung und Optimierung von Produktionsprozessen. Diese Technologien ermöglichen es, Maschinen und Produktionssysteme mit Intelligenz auszustatten. Die Daten werden direkt an der Quelle verarbeiten um schnelle und datengesteuerte Entscheidungen zu treffen. Besonders im industriellen Umfeld, wo Ressourcen wie Speicher und Rechenleistung oft begrenzt sind, bietet der Einsatz von Machine Learning auf Embedded Systemen eine vielversprechende, aber zugleich herausfordernde Möglichkeit.

Die Relevanz von Machine Learning in der Industrie steht außer Frage. Von der **vorausschauenden Wartung** [10] bis hin zur **Qualitätskontrolle** [8] bietet Machine Learning zahlreiche Möglichkeiten zur Effizienzsteigerung und Kostensenkung. Allerdings sind viele bestehende Machine-Learning-Modelle nicht für den Einsatz auf Embedded Systemen optimiert [3]. Die besonderen Herausforderungen, die sich durch eingeschränkte Ressourcen und strenge Echtzeitanforderungen ergeben, erfordern spezialisierte Ansätze für das Modell-Deployment auf solchen Systemen.

Aktuelle Forschung zeigt, dass der Einsatz von Machine Learning in Embedded Systemen zunehmend an Bedeutung gewinnt [1]. Zahlreiche Studien betonen die Vorteile von Edge-Computing in industriellen Anwendungen, insbesondere im Hinblick auf **Echtzeitanalysen und datengesteuerte Entscheidungsprozesse** [5]. Diese Arbeit knüpft an diese Forschung an und erweitert sie, indem ein Framework entwickelt wird, das speziell auf die Herausforderungen im industriellen Umfeld zugeschnitten ist.

Das Hauptziel dieser Arbeit ist es, ein optimiertes Modell-Deployment auf Embedded Systemen zu realisieren, das den spezifischen Bedingungen in der industriellen Umgebung gerecht wird. Dabei werden sowohl die technischen Herausforderungen als auch die praktischen Implikationen des Einsatzes von Machine Learning auf speicher- und rechenleistungseingeschränkten Systemen untersucht. Die Innovation dieser Arbeit liegt in der Entwicklung eines **optimierten Frameworks**, das sowohl die spezifischen Hardware-Einschränkungen von Embedded Systemen als auch die strengen Echtzeitanforderungen im industriellen Kontext berücksichtigt. Während bestehende Ansätze oft allgemeine Lösungen bieten, konzentriert sich dieses Framework auf die besonderen Herausforderungen in speicher- und rechenleistungseingeschränkten Umgebungen.

Ein spezifisch angepasstes ML-Framework zur Bereitstellung und Optimierung von ML-Modellen auf Embedded Systemen ist in der industriellen Praxis von hoher Relevanz. Der Entwurf eines ML-Frameworks, das die Anforderungen von Embedded Systemen in der Fertigung erfüllt, erfordert eine genaue Analyse

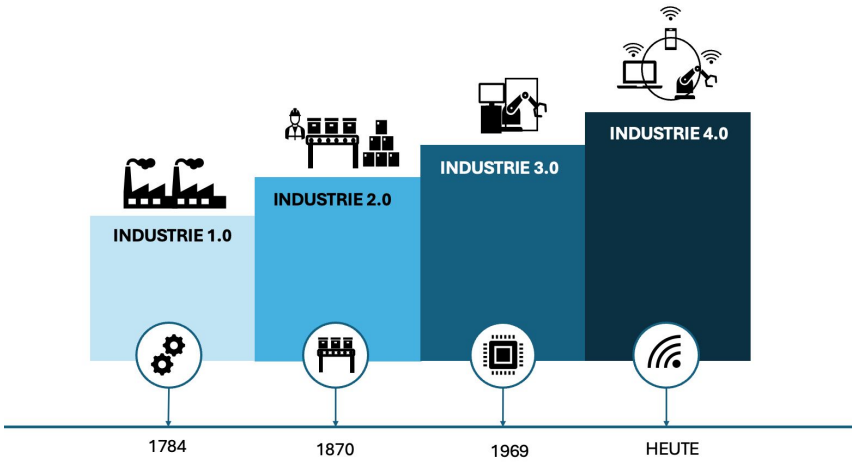


Abbildung 1.1: Die vier industriellen Revolutionen: Mechanisierung, Massenproduktion, Automatisierung, Vernetzte Systeme.

der Hardwarebeschränkungen und die Implementierung **effizienter Algorithmen**, die bei begrenzter Rechenkapazität eine hohe Modellgenauigkeit bieten. Ein entscheidender Vorteil des Edge-Computings liegt in der lokalen Datenverarbeitung, die Latenzzeiten reduziert und gleichzeitig die Sicherheit und Vertraulichkeit der Daten erhöht. Durch die Entwicklung eines solchen Frameworks könnte diese Arbeit dazu beitragen, Embedded Systeme besser für den Einsatz von ML-Anwendungen nutzbar zu machen und so die Produktionsprozesse zu verbessern.

Um dieses Ziel zu erreichen, kombiniert diese Arbeit eine strukturierte Literaturrecherche sowie eine praxisnahe Anforderungsanalyse. Zunächst werden die theoretischen Grundlagen von Embedded Systems und Machine Learning in der Industrie 4.0 untersucht. Anschließend entsteht ein Framework, das speziell auf die Anforderungen von Embedded Systemen abgestimmt ist. Dieses Framework wird in einer industriellen Umgebung getestet, um seine Leistungsfähigkeit und Effizienz zu validieren. Dabei sollen folgende Forschungsfragen beantwortet werden:

**Forschungsfragen:**

1. Wie kann ein ML-Framework für Embedded Systeme optimiert werden, um den Anforderungen der Industrie 4.0 zu entsprechen?
2. Welche spezifischen Anpassungen sind für ressourcenbeschränkte Umgebungen notwendig?

Die Arbeit ist wie folgt strukturiert: In Kapitel 2 werden die theoretischen Grundlagen und der Stand der Technik im Bereich Embedded Systems und Machine Learning in der Industrie 4.0 erörtert. Kapitel 3 beschreibt die Methodik, die bei der Entwicklung des Frameworks angewendet wird. In Kapitel 4 wird das entwickelte Framework im Detail vorgestellt, gefolgt von der Implementierung und Optimierung in Kapitel 5. Die Evaluation des Frameworks erfolgt in Kapitel 6, bevor in Kapitel 7 die Ergebnisse diskutiert werden. Abschließend fasst Kapitel 8 die Arbeit zusammen und gibt einen Ausblick auf zukünftige Entwicklungen.

# Kapitel 2

# Theoretischer Hintergrund

Um den aktuellen **Stand der Technik** und die wichtigsten Konzepte im Bereich der Embedded Systems und des Machine Learning in der Industrie 4.0 zu untersuchen, wird eine strukturierte **Literaturrecherche** durchgeführt. Diese Methodik gewährleistet eine umfassende und wissenschaftlich fundierte Darstellung des Themas, die auf relevanten Fachartikeln [2], Büchern [6] und Konferenzberichten [7] basiert. Der Fokus liegt auf der Identifizierung und Analyse von Herausforderungen und Lösungsansätzen für die Implementierung von Machine Learning (ML) auf Embedded Systems unter den Bedingungen der Industrie 4.0. Durch die strukturierte Recherche werden folgende Kernbereiche für das Kapitel ermittelt: die Rolle von Embedded Systems und ML in der Industrie 4.0, spezifische technische Herausforderungen sowie Ansätze zur Optimierung von ML-Modellen für ressourcenbeschränkte Umgebungen.

## 2.1 Embedded Systems und Machine Learning in der Industrie 4.0

### 2.1.1 Embedded Systems

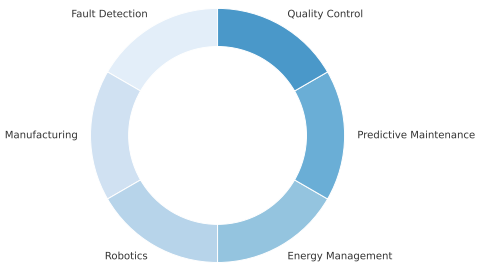


Abbildung 2.1: Arten von Embedded Systems in der Industrie 4.0

Embedded Systems sind spezialisierte Computersysteme, die in größere Maschinen oder Geräte integriert sind, um spezifische Aufgaben zu erfüllen. Sie sind oft in Umgebungen mit strengen Anforderungen an Zuverlässigkeit, Echtzeitfähigkeit und Energieeffizienz im Einsatz. Beispiele für Embedded Systems finden

sich in Automobilen, medizinischen Geräten, Industrieanlagen und Haushaltsgeräten. In der Industrie 4.0 spielen Embedded Systems eine Schlüsselrolle, da sie die intelligente Vernetzung und Steuerung von Maschinen ermöglichen.

### 2.1.2 Industrie 4.0

Der Begriff Ïndustrie 4.0"beschreibt die vierte industrielle Revolution, die durch die Digitalisierung und Vernetzung von Produktionsprozessen gekennzeichnet ist. Diese Transformation ermöglicht die Schaffung **intelligenter Fabriken**, in denen Maschinen und Systeme miteinander kommunizieren und **autonom** Entscheidungen treffen können. Embedded Systems sind dabei das Rückgrat der Industrie 4.0 , da sie die notwendige Hardwarebasis für die Integration von Sensoren, Aktoren und Kommunikationsschnittstellen bieten.

### 2.1.3 Machine Learning in der Industrie 4.0

Machine Learning ist ein zentraler Bestandteil der Industrie 4.0 , da es die Analyse großer Datenmengen und die Ableitung von Entscheidungen in Echtzeit ermöglicht. In Produktionsumgebungen wird Machine Learning zur vorausschauenden Wartung, Qualitätskontrolle, Prozessoptimierung [4] und vielen weiteren Anwendungen eingesetzt. Dabei müssen ML-Modelle häufig auf Embedded Systems ausgeführt werden, um Entscheidungen direkt vor Ort treffen zu können. Dies stellt jedoch besondere Anforderungen an die Modellgröße, Rechenleistung und Energieeffizienz.

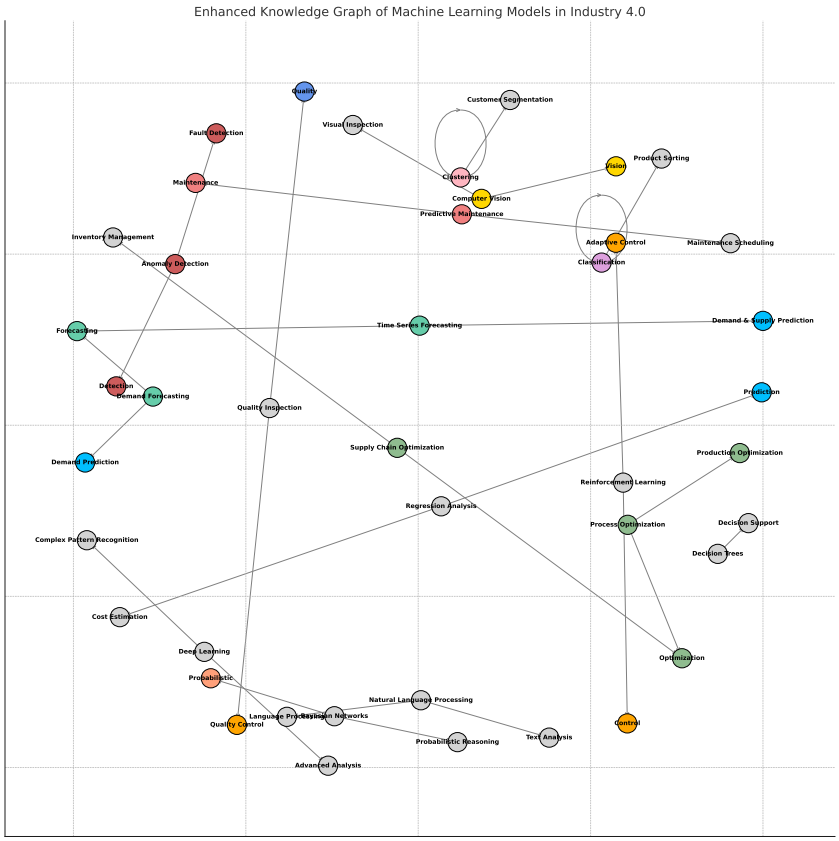


Abbildung 2.2: ML Modelle und ihre Anwendungen in der Industrie 4.0

### 2.1.4 Herausforderungen bei der Implementierung von Machine Learning auf Embedded Systems

#### Rechenleistung und Ressourcenbeschänkungen

Ein Hauptproblem bei der Implementierung von Machine Learning auf Embedded Systems ist die begrenzte Rechenleistung und der eingeschränkte Speicherplatz. Im Gegensatz zu leistungsstarken Servern oder Cloud-Umgebungen verfügen Embedded Systems , inbesondere Speicherprogrammierbare Steuerung und Industrie-PC (IPC), über deutlich weniger Ressourcen. Speicherprogrammierbare Steuerung (SPS) sind speziell für industrielle Automatisierungsaufgaben audgelegt und optimiert, haben jedoch nicht die Rechenkapazität, um komplexe Machine Learning Algorithmen auszuführen. IPCs sind zwar leistungsfähiger, stoßen jedoch ebenfalls schnell an ihre Grenzen, wenn großse ML-modelle oder rechenintensive Aufgaben direkt auf der Hardware ausgeführt werden sollen.

| **Aspekt** | **SPS (PLC)** | **ML-Computes (AI/ML-Instanzen)** |
|---|---|---|
| **Funktion** | Industrielle Steuerung, Automatisierung | KI/ML-Modelltraining, Datenverarbeitung |
| **Beispiele** | Siemens SIMATIC, B&R X20 | AWS P4d, Azure ND H100 v5 |
| **Leistung** | Echtzeit, geringe Latenz | Hohe Rechenleistung, parallele Verarbeitung |
| **Hardware** | ARM/x86-CPUs, I/O-Module | NVIDIA-GPUs (A100, H100), TPUs |
| **Netzwerk** | Fieldbus, Ethernet für Steuerung | InfiniBand, Hochgeschwindigkeits-Ethernet |
| **Speicher** | Moderater RAM, Flash | Hoher RAM (bis zu 1 TB), SSD/NVMe |
| **Programmierung** | Ladder-Logik, IEC 61131-3 | Python, CUDA, ML-Frameworks |
| **Skalierbarkeit** | Begrenzt, physische Erweiterung | Hoch, Cloud-Skalierung über GPUs |
| **Anwendung** | Maschinensteuerung, Echtzeitsysteme | KI, Deep Learning, Big Data |

Tabelle 2.1: Vergleich zwischen SPS und ML-Computes

#### Echtzeitanforderungen

Viele industrielle Anwendungen erfordern Echtzeitentscheidungen. Das bedeutet, dass die Zeit, die ein ML-Algorithmus für die Verarbeitung und Entscheidung benötigt, extrem kurz sein muss. Echtzeitsysteme müssen garantieren, dass eine Etscheidung innerhalb einer festgelegtenFrist getroffen wird. Dies stellt eine enorm Herausforderung für ML-Modelle dar, die in der Regel große Datenmengen verarbeiten und komplexe Berechnungen durchführen. In ressourcebeschränkten Umgebungen wie Embedded Systems kann dies zu significanten Latenzen führen, die in Echtzeitsysteme nicht toleriert werden können.

Beispiel

Ein Produktionsband in einer Automobilfabrik verwendet Machine Learning zur Qualitätsüberwachung von Karosserieteilen. Jedes Teil wird von Kameras gescannt, und die Daten werden in Echtzeit verarbeitet, um Defekte zu erkennen. Wenn das ML-System nicht innerhalb von wenigen Sekunden eine Entscheidung trifft, ob das Teil in Ordnung ist oder aussortiert werden muss, kann das gesamte Produktionsband verzögert werden. Diese Verzögerungen führen zu Produktionsausfällen und erhöhen die Kosten.

#### Modellkomplexität und Optimierung

Die Komplexität der ML-Modelle muss für den Einsatz auf Embedded Systems reduziert werden. Moderne ML-Modelle, wie tiefe neuronale Netze (Deep Neural Networks, DNNs), sind oft sehr groß und benötigen erhebliche Rechenleistung. Um diese Modelle auf Embedded Systemen auszuführen, müssen sie optimiert werden. Techniken wie Model **Pruning** (das Entfernen von unwichtigen Teilen des Modells) und **Quantisierung** (die Reduktion der Genauigkeit von Modellparametern) [9] werden häufig verwendet, um die Modelle kleiner und effizienter zu machen. Diese Optimierungstechniken sind jedoch oft mit einem Verlust an Genauigkeit verbunden, was den Einsatz in sicherheitskritischen Systemen einschränken kann.

#### Energieeffizienz

In vielen industriellen Anwendungen, insbesondere im Kontext der Industrie 4.0 , spielt die Energieeffizienz eine entscheidende Rolle. Embedded Systems sind oft batteriebetrieben oder müssen in energiearmen Umgebungen arbeiten. Das bedeutet, dass ML-Modelle so optimiert werden müssen, dass sie möglichst wenig Energie verbrauchen. Der Einsatz energieeffizienter Algorithmen und die Optimierung der Hardware, beispielsweise durch die Verwendung von Low-Power-Chips, sind essenziell, um ML-Modelle in eingebetteten Systemen langlebig und nachhaltig zu betreiben.

## 2.2 Fazit

Zusammenfassend lässt sich sagen, dass Embedded Systems und Machine Learning wesentliche Bausteine der Industrie 4.0 sind. Die Implementierung von Machine Learning auf Embedded Systems birgt jedoch signifikante Herausforderungen, insbesondere in Bezug auf Rechenleistung, Echtzeitanforderungen, Modellkomplexität und Energieeffizienz. Diese Herausforderungen müssen überwunden werden, um ML-Modelle effektiv in industriellen Produktionsumgebungen einzusetzen und den Nutzen von intelligenten, autonomen Systemen voll auszuschöpfen.

# Kapitel 3

# Methodik

n diesem Kapitel wird die Methodik beschrieben, die für die Entwicklung des Frameworks verwendet wird, das ML-Modelle für den Einsatz auf Embedded Systems optimiert. Dazu gehören die Durchführung der **Literaturrecherche**, die **Anforderungsanalyse** sowie die Auswahl der **Entwicklungsumgebung**, die verwendeten **Optimierungstechniken** und die **Implementierungsschritte** für das Framework. Der Fokus liegt auf der effizienten Ausführung von ML-Modellen auf ressourcenbeschränkten Geräten wie SPS und IPC in der industriellen Fertigung.

## 3.1 Literaturrecherche und Anforderungsanalyse

Um eine fundierte Basis für die Entwicklung des Frameworks zu schaffen, werden sowohl eine strukturierte Literaturrecherche als auch eine Anforderungsanalyse durchgeführt.

### 3.1.1 Literaturrecherche

Die Literaturrecherche dient dazu, aktuelle Ansätze und Techniken im Bereich der Embedded Systems und des Machine Learning in der Industrie 4.0 zu identifizieren. Folgende Schritte werden befolgt:

**Prozess:**

1. Definition der Forschungsfrage: Die zentrale Fragestellung zielt darauf ab, die Herausforderungen und Lösungsansätze zur effizienten Implementierung von ML-Modellen auf Embedded Systems in industriellen Anwendungen zu identifizieren.
2. Festlegung der Suchstrategie: Relevante wissenschaftliche Datenbanken wie IEEE Xplore, ACM Digital Library und SpringerLink werden genutzt. Suchbegriffe wie „Embedded Systems“, „Machine Learning“, „Edge Computing“ und „Industrie 4.0“ bilden die Grundlage für die Suche.
3. Filterung und Auswahl: Durch spezifische Inklusions- und Exklusionskriterien, wie Veröffentlichungsjahr und Relevanz für industrielle Anwendungen, werden die Suchergebnisse eingegrenzt. Insgesamt werden über 50 Fachartikel und Berichte analysiert, von denen 30 für die weitere Untersuchung relevant sind.

4. Analyse und Synthese: Die ausgewählten Artikel werden systematisch analysiert und die Ergebnisse in Bezug auf Schlüsselthemen und Herausforderungen zusammengefasst.

### 3.1.2 Anforderungsanalyse

Parallel zur Literaturrecherche wird eine Anforderungsanalyse durchgeführt, um spezifische Anforderungen der Industriepartner zu berücksichtigen. Diese Analyse folgt einem strukturierten Ansatz:

**Prozess:**

1. **Interviews und Workshops mit Industriepartnern**: Zur Identifikation praxisrelevanter Anforderungen werden Interviews und Workshops mit Industriepartnern durchgeführt, die Embedded Systems im Produktionsumfeld einsetzen. Hier werden Anforderungen bezüglich Rechenleistung, Energieeffizienz, Echtzeitanforderungen und Modellflexibilität identifiziert.
2. **Analyse der technischen Dokumentationen**: Technische Spezifikationen der verwendeten Hardware (z. B. SPS und IPC) werden untersucht, um Hardware-Einschränkungen und Schnittstellenanforderungen zu berücksichtigen.
3. **Kategorisierung der Anforderungen**: Die identifizierten Anforderungen werden in technische und funktionale Anforderungen unterteilt. Technische Anforderungen beziehen sich auf die Hardwareeinschränkungen und notwendige Optimierungstechniken, während funktionale Anforderungen Aspekte wie Echtzeitfähigkeit und Robustheit umfassen.
4. **Priorisierung**: Eine Priorisierung der Anforderungen wird vorgenommen, um sicherzustellen, dass die wichtigsten Kriterien wie Echtzeitanforderungen und Effizienz bei der Entwicklung des Frameworks berücksichtigt werden.

Priorisierung: Eine Priorisierung der Anforderungen wurde vorgenommen, um sicherzustellen, dass die wichtigsten Kriterien wie Echtzeitanforderungen und Effizienz bei der Entwicklung des Frameworks berücksichtigt werden.

## 3.2 Framework-Entwurf

Der erste Schritt zur Entwicklung des Frameworks ist die Definition der Anforderungen basierend auf den spezifischen Einschränkungen von Embedded Systems. Diese Anforderungen werden in Zusammenarbeit mit Industriepartnern und durch eine umfassende Literaturrecherche zu bestehenden Lösungen festgelegt. Hierbei werden nicht nur die Optimierung und Konfiguration der ML-Modelle berücksichtigt, sondern auch Aspekte wie die Datenverarbeitung, Priorisierung von Aufgaben, Logging und das Deployment von Modellen auf unterschiedlichen Zielplattformen. Die Schwerpunkte des Frameworks liegen auf:

- **Reduzierung der Modellgröße**: Um die effiziente Ausführung von ML-Modellen auf ressourcenbeschränkten Embedded Systems wie SPS oder IPCs zu ermöglichen, wird die Modellgröße reduziert, ohne dass die Genauigkeit signifikant leidet. Dies ist besonders für Echtzeitanwendungen entscheidend.

- **Datenverarbeitung und Vorverarbeitung**: Neben der Modelloptimierung ist eine effiziente Datenverarbeitung erforderlich. Die Vorverarbeitung von Sensordaten erfolgt in Echtzeit und filtert irrelevante Informationen, um sicherzustellen, dass nur relevante Daten für Modellvorhersagen genutzt werden. Dies unterstützt die Fähigkeit, Entscheidungen in Echtzeit zu treffen.
- **Deployment und Aktualisierung der Modelle**: Eine zentrale Herausforderung in industriellen Umgebungen ist das flexible Deployment von ML-Modellen auf verschiedenen Geräten (SPS, IPCs etc.). Das Framework ermöglicht eine einfache Bereitstellung neuer oder aktualisierter Modelle, ohne den Produktionsprozess zu stören, und bietet ein robustes Update-Management für nahtlose Integration in bestehende Systeme.
- **Priorisierung von Aufgaben**: In Produktionsumgebungen ist die Priorisierung bestimmter Aufgaben zur Erfüllung von Betriebsanforderungen notwendig. Das Framework bevorzugt daher priorisierte ML-Tasks gegenüber weniger kritischen Aufgaben, was insbesondere bei begrenzten Ressourcen und parallelem Prozessbetrieb entscheidend ist.
- **Logging und Überwachung**: Um die Systemausführung und -wartung zu unterstützen, integriert das Framework ein umfassendes Logging- und Überwachungssystem. Dieses protokolliert Modellvorhersagen sowie Systemstatusinformationen (z.B. Ressourcenauslastung, Latenzen) und dient der Identifikation von Fehlern, der Leistungsüberwachung und der langfristigen Wartung des Systems.
- **Sicherstellung der Echtzeitfähigkeit**: Da viele industrielle Anwendungen Echtzeitanforderungen erfordern, stellt das Framework sicher, dass Vorhersagen innerhalb vorgegebener Zeitgrenzen getroffen werden. Dies erfordert eine optimierte Laufzeitleistung und die Einhaltung zeitlicher Anforderungen des Produktionsprozesses.
- **Einfache Integration in bestehende industrielle Steuerungssysteme**: Das Framework ist so konzipiert, dass es sich problemlos in bestehende Steuerungsarchitekturen (wie IPC oder SPS) integriert, um den Einsatz ohne grundlegende Änderungen an der Infrastruktur zu ermöglichen.

Das Framework wird als Middleware entwickelt, die als Adapter zwischen den ML-Modellen und den Zielsystemen (SPS, IPC) fungiert. Es unterstützt verschiedene Optimierungstechniken, um die Ausführung der Modelle auf hardwarebeschränkten Systemen zu ermöglichen.

## 3.3 Programmiersprachen und Entwicklungsumgebungen

### 3.3.1 Python

Python ist eine der am häufigsten verwendeten Programmiersprachen für Machine Learning und bietet eine Vielzahl von Bibliotheken und Frameworks, die für industrielle Anwendungen und Embedded-Modelle nützlich sind. Es ist bekannt für seine einfache Handhabung und große Community, was es zur ersten Wahl für viele ML-Entwickler macht.

- **TensorFlow und TensorFlow Lite**: TensorFlow bietet eine leistungsstarke Umgebung für die Entwicklung von neuronalen Netzen. TensorFlow Lite wurde speziell entwickelt, um TensorFlow-Modelle für ressourcenbeschränkte Geräte zu optimieren und diese auf Embedded-Systemen lauffähig zu machen.

- **scikit-learn**: Dieses Framework bietet eine umfassende Bibliothek für klassische Machine-Learning-Algorithmen wie Entscheidungsbäume, Random Forests, Support Vector Machines (SVMs), lineare Modelle, K-Means und mehr. Diese Modelle sind oft weniger rechenintensiv und daher für viele industrielle Anwendungen gut geeignet.
- **XGBoost**: XGBoost ist bekannt für seine Effizienz und Leistung bei Gradient-Boosting-Modellen und eignet sich gut für industrielle Anwendungen wie Anomalieerkennung, Qualitätskontrolle und vorausschauende Wartung.
- **LightGBM**: LightGBM bietet eine ähnliche Funktionalität wie XGBoost, ist jedoch für größere Datensätze und schnellere Trainingszeiten optimiert. Es wird häufig in Echtzeitanwendungen verwendet.
- **PyTorch**: PyTorch ist eine flexible Alternative zu TensorFlow, insbesondere für Forschungsprojekte und Produktionssysteme, die dynamische Berechnungsgrafen erfordern. Es wird auch in Embedded-ML-Anwendungen verwendet, wenn größere Modelle benötigt werden.
- **CatBoost**: Ein Gradient-Boosting-Framework, das speziell für den Umgang mit kategorischen Daten entwickelt wurde. CatBoost eignet sich besonders gut für industrielle Anwendungen, bei denen viele der Eingabedaten kategorisch sind.
- **ONNX**: ONNX bietet eine plattformübergreifende Möglichkeit, ML-Modelle zu konvertieren und auszuführen, die in verschiedenen Frameworks wie TensorFlow, PyTorch oder scikit-learn entwickelt wurden. ONNX erleichtert das Deployment von Modellen auf Embedded- und Edge-Geräten.

### 3.3.2 Rust

Rust ist eine moderne Systemprogrammiersprache, die besonders für ihre Speicher- und Speichersicherheitsfunktionen bekannt ist. Sie wird zunehmend für Embedded- und Edge-Anwendungen verwendet, da sie hohe Leistung und Sicherheit bietet.

- **Rust-ML-Bibliotheken**: Obwohl Rust weniger bekannt für Machine Learning ist als Python, gibt es Bibliotheken wie **Linfa**, die klassische ML-Algorithmen wie lineare Regression, K-Means und Entscheidungsbäume bieten.
- **Tch-RS (PyTorch in Rust)**: Tch-RS ist eine Rust-Bindung für PyTorch, die es ermöglicht, PyTorch-Modelle in der Rust-Umgebung zu verwenden. Diese Bibliothek bietet Zugriff auf viele der in PyTorch vorhandenen Funktionen und kann in eingebetteten Umgebungen verwendet werden, in denen Rust bevorzugt wird.
- **SmartCore**: Eine weitere Machine-Learning-Bibliothek in Rust, die viele der klassischen ML-Algorithmen wie Entscheidungsbäume, Random Forests und lineare Modelle unterstützt. Sie eignet sich gut für eingebettete Anwendungen, bei denen Performance und Sicherheit eine Rolle spielen.
- **ONNX mit Rust**: ONNX kann auch in Rust integriert werden, um neuronale Netze oder klassische Modelle, die in anderen Frameworks entwickelt wurden, auf Embedded-Systemen bereitzustellen.

## 3.4 Tools für Modelloptimierung und Konvertierung

### 3.4.1 TensorFlow Lite

TensorFlow Lite ist eine leichtgewichtige Version von TensorFlow, die speziell für den Einsatz auf mobilen und eingebetteten Geräten entwickelt wurde. Es unterstützt verschiedene Techniken wie Quantisierung, Pruning und Delegates, um die Modellgröße und Rechenanforderungen zu reduzieren.

### 3.4.2 ONNX Runtime

ONNX Runtime ermöglicht es, ML-Modelle, die in verschiedenen Frameworks wie TensorFlow, PyTorch oder scikit-learn entwickelt wurden, auf einer Vielzahl von Hardwareplattformen auszuführen. Es bietet eine Möglichkeit, Modelle plattformübergreifend zu optimieren und effizient auf Edge- und Embedded-Systemen bereitzustellen.

### 3.4.3 Apache TVM

Apache TVM ist ein Maschinelles Learning Compiler-Framework, das es ermöglicht, ML-Modelle für eine Vielzahl von Hardwarearchitekturen zu optimieren. TVM bietet spezielle Optimierungen für Embedded-Geräte und ermöglicht es, Modelle effizient auf Mikrocontrollern, GPUs und Edge Computing Plattformen auszuführen.

### 3.4.4 TinyML Frameworks

TinyML Frameworks wie TensorFlow Micro und uTensor sind speziell für die Ausführung von ML-Modellen auf extrem ressourcenbeschränkten Geräten wie Mikrocontrollern und Sensoren konzipiert. Diese Frameworks ermöglichen es, Modelle auf Kleinstgeräten zu implementieren, ohne die Rechenressourcen zu überlasten.

### 3.4.5 TensorRT

TensorRT ist ein High-Performance-Deep-Learning-Inferenz-Optimierer, der von NVIDIA entwickelt wurde. Es wird verwendet, um die Inferenzleistung von neuronalen Netzen auf NVIDIA-Plattformen wie Jetson Nano zu beschleunigen, und eignet sich besonders für Echtzeitanwendungen, die auf leistungsstarker Edge-Hardware laufen.

## 3.5 Tools für Deployment-Optimierung und Kontrolle

### 3.5.1 Docker

Docker wird verwendet, um Machine-Learning-Modelle und deren Abhängigkeiten in Containern zu verpacken, sodass sie auf verschiedenen Plattformen bereitgestellt werden können. Dies vereinfacht das Deployment von Modellen in Produktionsumgebungen und ermöglicht eine einfache Skalierbarkeit und Wiederholbarkeit.

### 3.5.2 K3s (leichtgewichtige Kubernetes-Distribution)

K3s ist eine leichtgewichtige Kubernetes-Distribution, die speziell für Edge- und IoT-Geräte entwickelt wurde. Es ermöglicht die Orchestrierung von Machine-Learning-Modellen auf verschiedenen Geräten und die Verwaltung von ML-Diensten in einer Edge-Computing-Umgebung.

### 3.5.3 MLflow

MLflow wird zur Verwaltung des gesamten Lebenszyklus von Machine-Learning-Modellen eingesetzt. Es ermöglicht die Versionierung, das Tracking und das Deployment von Modellen und ist besonders nützlich, wenn verschiedene Modellversionen verwaltet und überwacht werden müssen.

### 3.5.4 PyInstaller

PyInstaller wird verwendet, um Python-Anwendungen und Machine-Learning-Modelle in ausführbare Dateien zu konvertieren, die auf Geräten ohne Python-Laufzeitumgebung ausgeführt werden können. Es ermöglicht das Deployment von ML-Modellen als Windows-Dienste oder eigenständige Anwendungen auf IPCs oder Windows-basierten Embedded-Systemen.

### 3.5.5 Zephyr RTOS

Zephyr ist ein Echtzeitbetriebssystem (RTOS), das speziell für Embedded-Geräte entwickelt wurde. Es bietet eine stabile Plattform für die Ausführung von ML-Modellen auf Mikrocontrollern und unterstützt Echtzeitanforderungen, die in der industriellen Fertigung entscheidend sind.

### 3.5.6 Conda und Pip

**Conda** und **Pip** sind essentielle Werkzeuge, um Abhängigkeiten für Machine-Learning-Projekte zu verwalten. Beide Systeme wurden verwendet, um die unterschiedlichen Anforderungen des Projekts zu erfüllen:

- **Conda** wurde zur Verwaltung von umfangreichen Abhängigkeiten wie *TensorFlow*, *PyTorch* und *OpenVINO* verwendet. Conda bietet eine effizientere Verwaltung großer Bibliotheken und ermöglicht

die Integration von Systembibliotheken mit Python-Paketen. Dadurch wurde sichergestellt, dass das Framework auch auf ressourcenbeschränkten Geräten effizient läuft.

- **Pip** wurde für Bibliotheken eingesetzt, die nicht über Conda verfügbar sind oder spezifische Anforderungen haben, wie z.B. *onnxmltools* oder *tensorflow-model-optimization*.

Die Kombination dieser Werkzeuge ermöglichte es, ein Framework zu entwickeln, das ML-Modelle effizient auf Embedded Systems implementiert.

## 3.6 Optimierungstechniken

Die Optimierung von ML-Modellen für Embedded Systems ist ein kritischer Bestandteil dieses Projekts. Folgende Techniken wurden implementiert:

### 3.6.1 Quantisierung

Durch Quantisierung wird die Größe eines ML-Modells reduziert, indem die Genauigkeit der Gewichtungen von 32-Bit-Gleitkommazahlen auf 8-Bit-Ganzzahlen verringert wird. Dies führt zu einer deutlichen Reduktion des Speicherbedarfs und ermöglicht es, die Modelle auf Geräten mit begrenzten Ressourcen auszuführen. Diese Technik wurde insbesondere bei Modellen angewendet, die auf SPS und IPCs mit begrenzter Speicher- und Rechenkapazität eingesetzt werden.

### 3.6.2 Model Pruning

Beim Model Pruning werden unwichtige Neuronen und Verbindungen aus dem ML-Modell entfernt. Dieser Prozess verringert die Modellgröße, ohne die Genauigkeit signifikant zu beeinträchtigen. Durch das Pruning konnten wir die Ausführungsgeschwindigkeit auf Embedded Systems verbessern, während die Genauigkeit nur minimal reduziert wurde.

### 3.6.3 Edge-Optimierte Algorithmen

Zusätzlich zu den oben genannten Techniken wurden speziell für Embedded Systems optimierte Algorithmen implementiert. Diese Algorithmen sind darauf ausgelegt, mit minimalem Rechenaufwand auszukommen und gleichzeitig Echtzeitentscheidungen zu ermöglichen. Ein Beispiel hierfür sind binäre Entscheidungsbäume und lineare Modelle, die weniger komplex sind als neuronale Netzwerke, jedoch in vielen industriellen Anwendungen ausreichend leistungsfähig sind.

## 3.7 Implementierungsschritte

### 3.7.1 Anforderungsanalyse

Zunächst wurden die Anforderungen der industriellen Partner erfasst und eine Anforderungsanalyse durchgeführt. Die wichtigsten Anforderungen konzentrierten sich auf die Reduktion der Modellgröße, die Sicherstellung der Echtzeitfähigkeit sowie die ressourcenschonende Ausführung auf Embedded-Devices. Besondere Beachtung fanden dabei die spezifischen Hardware-Einschränkungen der eingesetzten SPS- und IPC-Systeme. Zudem wurden Zielfunktionen und Prioritäten festgelegt, um eine Balance zwischen Modellkomplexität, Laufzeitleistung und Speicherverbrauch zu finden.

### 3.7.2 Entwicklung des Frameworks

Das Framework wurde so entworfen, dass es als flexible Schnittstelle zwischen den optimierten ML-Modellen und den Embedded Systems dient. Die Architektur wurde in verschiedene Module unterteilt, die sowohl die Modelloptimierung als auch die Kommunikation zwischen den Geräten erleichtern. Wesentliche Aspekte der Entwicklung umfassten:

- **Architekturentwurf**: Die Architektur wurde modular entworfen, um die einfache Integration und den Austausch verschiedener ML-Modelle zu ermöglichen.
- **Modelloptimierungstechniken**: Techniken wie Quantisierung, Pruning und Kompression wurden eingesetzt, um die Modellgröße zu reduzieren und die Ausführungseffizienz zu steigern.
- **API-Design**: Eine flexible API wurde entwickelt, um den Zugriff auf verschiedene ML-Modelle zu ermöglichen und eine einfache Bereitstellung auf verschiedenen Embedded Devices sicherzustellen.

### 3.7.3 Integration und Tests

Das Framework wurde auf verschiedene Embedded Devices, einschließlich SPS und IPCs, integriert und umfassend getestet. Dieser Schritt umfasste:

- **Hardware-Integration**: Die Integration des Frameworks auf unterschiedlichen Embedded-Platformen und die Anpassung an deren spezifische Hardware-Eigenschaften.
- **Unit-Tests**: Unit-Tests wurden verwendet, um die korrekte Funktionalität der wichtigsten Komponenten sicherzustellen.
- **Hardware-in-the-Loop (HIL) Tests**: HIL-Tests wurden durchgeführt, um das Framework unter realen Bedingungen zu testen und sicherzustellen, dass die Modelle in Echtzeit mit der Hardware interagieren.
- **Laufzeitleistungsprüfung**: Es wurden spezifische Benchmarks zur Überprüfung der Ausführungszeiten definiert, um sicherzustellen, dass die Echtzeitanforderungen der industriellen Anwendungen erfüllt werden.

### 3.7.4 Evaluation der Ergebnisse

Nach der Integration wurde das Framework anhand vordefinierter Leistungsmetriken evaluiert:

- **Leistungsmetriken**: Die wichtigsten Metriken umfassen die Ausführungszeit der Modelle, den Speicherverbrauch und den Energieverbrauch auf den Embedded Devices.
- **Vergleich mit den Anforderungen**: Die Ergebnisse wurden mit den in der Anforderungsanalyse definierten Zielwerten verglichen, um zu überprüfen, inwieweit die Anforderungen erfüllt wurden.
- **Robustheit und Fehlerbehandlung**: Es wurden Tests durchgeführt, um die Robustheit des Frameworks und seine Fähigkeit, auf Fehler wie Hardwareausfälle oder Modellfehler zu reagieren, zu bewerten.
- **Optimierungspotential**: Basierend auf den Evaluierungsergebnissen wurde Potenzial für weitere Optimierungen, wie die Reduktion des Energieverbrauchs oder die Verbesserung der Ausführungszeiten, identifiziert.

### 3.7.5 Zusätzliche Anforderungen

Weitere Themen die zu beachten sind:

- **Sicherheit und Datenschutz**: Es wurde darauf geachtet, dass das Framework den Anforderungen an die Datensicherheit in der industriellen Fertigung entspricht.
- **Flexibilität und Erweiterbarkeit**: Das Framework wurde so gestaltet, dass zukünftige Erweiterungen und neue ML-Modelle einfach integriert werden können.
- **Wartung und Updates**: Mechanismen für einfache Wartung und Modell-Updates wurden implementiert, um die langfristige Verwendbarkeit des Frameworks zu gewährleisten.

## 3.8 Zusammenfassung

Das entwickelte Framework wurde basierend auf einer detaillierten Anforderungsanalyse erstellt, modular entworfen und für die Ausführung auf Embedded Devices optimiert. Durch umfassende Tests und Evaluationen konnte die Effizienz, Echtzeitfähigkeit und Robustheit des Frameworks sichergestellt werden. Optimierungsmöglichkeiten wurden identifiziert und bilden die Grundlage für zukünftige Verbesserungen.

# Kapitel 4

# Entwicklung des Frameworks

In diesem Kapitel wird die Architektur und die Entwicklung des Frameworks detailliert beschrieben. Es wird erläutert, wie das Framework entworfen wurde, um ML-Modelle für ressourcenbeschränkte Umgebungen, wie Embedded Systems und Edge-Devices, zu optimieren. Der Fokus liegt auf der modularen Struktur des Frameworks, der Flexibilität für verschiedene ML-Modelle und den Optimierungstechniken, die angewendet wurden, um die Ausführungseffizienz zu maximieren.

## 4.1 Architektur des Frameworks

Die Architektur des Frameworks ist modular aufgebaut, um Flexibilität, Erweiterbarkeit und einfache Wartbarkeit sicherzustellen. Die wichtigsten Komponenten des Frameworks sind in einzelne Module unterteilt, die jeweils spezifische Aufgaben erfüllen. Das Framework ist darauf ausgelegt, in ressourcenbeschränkten Umgebungen zu laufen und unterstützt sowohl neuronale Netze als auch klassische Machine Learning Modelle.

### 4.1.1 Modulare Architektur

Das Framework besteht aus mehreren Modulen, die miteinander kommunizieren, um die Ausführung von Machine Learning Modellen auf Embedded Systems zu ermöglichen. Die folgenden Hauptmodule wurden implementiert:

- **Modul für Modelloptimierung**: Dieses Modul ist für die Optimierung der ML-Modelle verantwortlich. Es umfasst Techniken wie Pruning, Quantisierung und Modellkompression, um die Modelle für die Ausführung auf Geräten mit begrenzter Rechenleistung und Speicherressourcen vorzubereiten.
- **Modul für Datenverarbeitung**: Dieses Modul übernimmt die Vorverarbeitung und Filterung der Eingabedaten, die von den angeschlossenen Sensoren oder anderen Quellen bereitgestellt werden. Die Vorverarbeitung wird direkt auf dem Embedded Device durchgeführt, um die Datengröße zu reduzieren und nur relevante Informationen an das ML-Modell weiterzuleiten.

- **Modul für das Modelldepot**: Dieses Modul verwaltet die Speicherung und Bereitstellung von ML-Modellen. Es unterstützt verschiedene Modelldeployment-Strategien, darunter das lokale Deployment auf Embedded Devices und das remote Deployment über Edge-Server.
- **Modul für Echtzeitausführung**: Da viele industrielle Anwendungen Echtzeitanforderungen haben, sorgt dieses Modul dafür, dass ML-Modelle innerhalb vorgegebener Zeitgrenzen ausgeführt werden. Es überwacht die Laufzeitleistung und sorgt für eine effiziente Aufgabenpriorisierung.
- **Modul für Überwachung und Logging**: Dieses Modul bietet Funktionen zur Überwachung der Systemleistung und zur Protokollierung von Modellvorhersagen sowie Systemereignissen. Es ermöglicht es, Leistungskennzahlen wie CPU-Auslastung, Speichernutzung und Energieverbrauch zu überwachen.

### 4.1.2 Kommunikationsschnittstellen und API

Die Architektur umfasst eine flexible API, die es ermöglicht, verschiedene ML-Modelle einfach in das Framework zu integrieren. Die API stellt Funktionen bereit, um:

- ML-Modelle zu laden und zu konfigurieren.
- Vorhersagen zu treffen und die Ergebnisse in Echtzeit zu übermitteln.
- Modelle zu aktualisieren oder neu zu deployen.

Die API ist darauf ausgelegt, mit unterschiedlichen Protokollen (z.B. MQTT, HTTP, OPC-UA) zu arbeiten, um die Kommunikation zwischen Embedded Systems und Edge-Servern zu ermöglichen.

## 4.2 Optimierung der ML-Modelle für ressourcenbeschränkte Umgebungen

Ein zentraler Bestandteil der Framework-Entwicklung war die Anpassung der ML-Modelle an die beschränkten Ressourcen von Embedded Systems. Diese Optimierung erfolgte durch verschiedene Techniken, die im Folgenden erläutert werden.

### 4.2.1 Pruning

Pruning wurde verwendet, um unnötige Verbindungen in tiefen neuronalen Netzen zu entfernen und so die Komplexität der Modelle zu reduzieren. Durch das Entfernen dieser Verbindungen konnte der Speicherbedarf der Modelle deutlich gesenkt werden, ohne dass die Genauigkeit signifikant beeinträchtigt wurde.

### 4.2.2 Quantisierung

Quantisierung ist eine Technik, die verwendet wird, um die Modellgewichte von 32-Bit-Floating-Point-Zahlen auf 8-Bit-Ganzzahlen zu reduzieren. Diese Technik reduziert sowohl den Speicherbedarf als auch

die Berechnungsanforderungen des Modells. Dies ist besonders wichtig für Geräte mit begrenzter Rechenleistung, wie Mikrocontroller oder Embedded Systems in der industriellen Fertigung.

### 4.2.3 Modellkompression

Um die Größe der ML-Modelle weiter zu reduzieren, wurden Techniken der Modellkompression angewendet. Hierbei wurden redundante oder weniger wichtige Parameter des Modells komprimiert, um sowohl den Speicherverbrauch als auch die Ladezeiten der Modelle zu minimieren.

## 4.3 Anpassung an verschiedene Embedded- und Edge-Geräte

Das Framework wurde entwickelt, um auf einer Vielzahl von Hardwareplattformen zu laufen, von Mikrocontrollern bis hin zu leistungsstarken Edge-Geräten. Dabei wurde sichergestellt, dass das Framework flexibel genug ist, um auf unterschiedlichen Geräten mit variierenden Ressourcenanforderungen zu funktionieren.

### 4.3.1 Unterstützte Hardwareplattformen

Das Framework wurde für die folgenden Hardwareplattformen optimiert:

- **Speicherprogrammierbare Steuerungen (SPS)**: Diese Geräte werden in der industriellen Automatisierung verwendet und erfordern robuste und echtzeitfähige ML-Modelle.
- **Industrie-PCs (IPCs)**: IPCs bieten mehr Rechenleistung und Speicher als SPS und eignen sich für komplexere ML-Modelle und Datenverarbeitungsaufgaben.
- **Mikrocontroller**: Mikrocontroller sind stark ressourcenbeschränkte Geräte, auf denen besonders kleine und optimierte ML-Modelle eingesetzt werden müssen.
- **Edge-Devices**: Leistungsstarke Edge-Geräte, die zur Vorverarbeitung großer Datenmengen und zur Ausführung komplexerer ML-Modelle verwendet werden können.

### 4.3.2 Anpassung an verschiedene Ressourcenprofile

Jede Hardwareplattform hat unterschiedliche Anforderungen an Speicher, Rechenleistung und Energieverbrauch. Das Framework bietet Mechanismen, um Modelle und ihre Ausführungsumgebung dynamisch an die verfügbaren Ressourcen der Zielplattform anzupassen. Dazu gehört die automatische Auswahl der geeigneten Optimierungstechniken (z.B. Quantisierung oder Pruning) basierend auf den verfügbaren Ressourcen.

## 4.4 Zusammenfassung der Framework-Entwicklung

Das entwickelte Framework wurde so konzipiert, dass es ML-Modelle effizient auf Embedded- und Edge-Geräten ausführen kann. Durch die modulare Architektur und die umfassende Unterstützung von Optimierungstechniken wie Pruning, Quantisierung und Modellkompression wird sichergestellt, dass das Framework in einer Vielzahl von ressourcenbeschränkten Umgebungen eingesetzt werden kann. Die flexible API und die anpassbare Struktur ermöglichen es, unterschiedliche ML-Modelle einfach zu integrieren und zu verwalten. Die Entwicklung des Frameworks legt den Grundstein für eine zukunftssichere Lösung, die in der industriellen Fertigung eingesetzt werden kann.

# Kapitel 5

# Implementierung und Optimierung

In diesem Kapitel wird die Implementierung und Optimierung von Machine Learning Modellen für den Einsatz auf Embedded Systems detailliert beschrieben. Es werden die angewendeten Optimierungstechniken erläutert sowie der Prozess des Modelldeployments auf Geräten wie SPS, IPCs und Mikrocontrollern. Der Schwerpunkt liegt auf der Anpassung der Modelle an die beschränkten Ressourcen dieser Systeme, um eine effiziente Echtzeitausführung zu gewährleisten.

## 5.1 Optimierung der ML-Modelle

Da Embedded Systems nur begrenzte Rechenleistung, Speicher und Energie zur Verfügung haben, mussten die Machine Learning Modelle entsprechend angepasst werden. Verschiedene Optimierungstechniken wurden angewendet, um sicherzustellen, dass die Modelle effizient auf diesen Geräten laufen, ohne die Genauigkeit der Vorhersagen signifikant zu beeinträchtigen.

### 5.1.1 Quantisierung

Die Quantisierung wurde verwendet, um die Modellgrößen und die Rechenanforderungen zu reduzieren. Dabei wurden die 32-Bit-Gleitkommazahlen der Modellgewichte in 8-Bit-Ganzzahlen umgewandelt, was die Modellgröße und die Anzahl der Berechnungen erheblich reduziert hat.

- **Post-Training Quantization**: Nach dem Training des Modells wurde die Quantisierung durchgeführt, indem die Gleitkommawerte der Modellparameter auf Ganzzahlen reduziert wurden. Dies ermöglichte eine geringere Speichernutzung und schnellere Berechnungen.
- **Quantization-Aware Training (QAT)**: Bei einigen Modellen wurde das Quantization-Aware Training verwendet, um die Auswirkungen der Quantisierung während des Trainingsprozesses zu simulieren. Dadurch konnte die Modellgenauigkeit nach der Quantisierung besser erhalten bleiben.

### 5.1.2 Pruning

Pruning wurde angewendet, um unnötige Gewichte und Verbindungen aus neuronalen Netzen zu entfernen. Dies reduzierte die Komplexität und Größe der Modelle erheblich, ohne die Genauigkeit der Vorhersagen zu stark zu beeinträchtigen.

- **Unstrukturierter Pruning**: Durch das Entfernen einzelner Verbindungen zwischen Neuronen, die als weniger wichtig für die Modellleistung eingestuft wurden, konnte die Rechenkomplexität reduziert werden.
- **Strukturierter Pruning**: Hierbei wurden ganze Neuronen oder Filter entfernt, um die Modelle schlanker zu machen und gleichzeitig die Berechnungseffizienz zu steigern.

### 5.1.3 Modellkompression

Neben Pruning und Quantisierung wurde Modellkompression verwendet, um die Gesamtgröße der Modelle zu verringern. Hierbei wurden spezielle Algorithmen eingesetzt, die redundante Parameter im Modell identifizierten und diese komprimierten, ohne die Vorhersageleistung wesentlich zu beeinflussen.

### 5.1.4 Optimierung für spezifische Hardware

Da die Rechenkapazitäten von Embedded Systems variieren, wurden die Modelle für unterschiedliche Hardware-Plattformen optimiert. Verschiedene Hardware-Spezifika, wie die Anzahl der verfügbaren Prozessoren oder die Größe des Speichers, wurden berücksichtigt, um die optimale Leistung auf Geräten wie SPS, IPCs und Mikrocontrollern zu gewährleisten.

- **TensorFlow Lite**: TensorFlow Lite wurde verwendet, um Modelle für Mikrocontroller und andere ressourcenbeschränkte Geräte zu optimieren. Es bot Funktionen wie Quantisierung und eine optimierte Inferenzlaufzeit.
- **ONNX Runtime**: ONNX Runtime wurde eingesetzt, um die Modelle auf verschiedenen Embedded-Plattformen auszuführen. Es ermöglichte eine plattformübergreifende Optimierung und das Deployment auf heterogenen Hardware-Umgebungen.

## 5.2 Deployment der optimierten Modelle

Nachdem die Modelle optimiert waren, wurde der nächste Schritt das Deployment auf den Embedded Systems . Dieser Abschnitt beschreibt den Prozess der Modellbereitstellung und die Herausforderungen, die bei der Ausführung der Modelle in Echtzeit auf Embedded-Hardware auftraten.

### 5.2.1 Modellbereitstellung auf SPS und IPCs

SPS- und IPC-Systeme wurden verwendet, um die optimierten Modelle in einer industriellen Umgebung auszuführen. Aufgrund der unterschiedlichen Rechenleistung und Speicherkapazität dieser Systeme war

es notwendig, spezifische Deployment-Strategien zu implementieren.

- **Lokale Ausführung**: Modelle wurden direkt auf den Geräten ausgeführt, um Verzögerungen zu minimieren und Echtzeitanforderungen zu erfüllen.
- **Remote Deployment**: In einigen Fällen wurden Modelle auf Edge-Geräten bereitgestellt, die eng mit den SPS und IPCs verbunden waren, um eine effizientere Berechnung durchzuführen und gleichzeitig die Hauptsysteme zu entlasten.

### 5.2.2 Verwaltung des Modelldepots

Ein zentrales Modelldepot wurde implementiert, um die Verwaltung und Bereitstellung verschiedener Modellversionen zu erleichtern. Dieses Depot ermöglichte es, Modelle zentral zu speichern und von dort aus auf den Embedded-Systemen bereitzustellen. Dies ermöglichte eine einfache Aktualisierung der Modelle und eine bessere Verwaltung der Modellversionen.

### 5.2.3 Echtzeitausführung und Priorisierung

Da viele industrielle Anwendungen strenge Echtzeitanforderungen haben, musste sichergestellt werden, dass die Modelle in vorgegebenen Zeitrahmen ausgeführt werden. Dies erforderte eine Priorisierung der Aufgaben und eine enge Überwachung der Laufzeitleistung. Dabei wurden folgende Maßnahmen ergriffen:

- **Task-Priorisierung**: Die Modellvorhersagen wurden priorisiert, um sicherzustellen, dass zeitkritische Aufgaben zuerst ausgeführt werden.
- **Überwachung der Ausführungszeit**: Die Laufzeitleistung wurde kontinuierlich überwacht, um sicherzustellen, dass die Echtzeitanforderungen eingehalten wurden.

### 5.2.4 Tests und Validierung der Laufzeitleistung

Nach der Bereitstellung der optimierten Modelle wurden umfangreiche Tests durchgeführt, um die Laufzeitleistung und die Genauigkeit der Vorhersagen zu überprüfen. Diese Tests beinhalteten:

- **Unit-Tests**: Zur Validierung der Funktionalität des Frameworks und der korrekt ausgeführten Vorhersagen.
- **Performance-Benchmarks**: Die Ausführungszeiten wurden gemessen und mit den zuvor festgelegten Echtzeitanforderungen verglichen.
- **Hardware-in-the-Loop (HIL)-Tests**: Diese Tests simulierten reale Hardware-Interaktionen, um die Robustheit und Leistung des Systems zu überprüfen.

## 5.3 Zusammenfassung

Die Implementierung und Optimierung der Machine-Learning-Modelle für Embedded Systems erfolgte durch den Einsatz von Techniken wie Quantisierung, Pruning und Modellkompression. Die optimierten Modelle wurden erfolgreich auf verschiedenen Hardwareplattformen, einschließlich SPS, IPCs und Mikrocontrollern, bereitgestellt und in Echtzeitanwendungen integriert. Durch die laufende Überwachung und Priorisierung der Aufgaben konnte sichergestellt werden, dass die Modelle den strengen Echtzeitanforderungen der industriellen Umgebung entsprachen.

# Kapitel 6

# Evaluation

In diesem Kapitel wird die Leistung des entwickelten Frameworks anhand verschiedener Metriken evaluiert. Dabei stehen die Latenzzeiten, der Durchsatz, die Ressourcenauslastung sowie die Modellgenauigkeit im Vordergrund. Die Evaluation wurde auf verschiedenen Embedded Systems durchgeführt, um sicherzustellen, dass das Framework den spezifischen Anforderungen der Zielsysteme entspricht und in Echtzeitanwendungen robust und effizient funktioniert.

## 6.1 Zielsetzung der Evaluation

Die Evaluation verfolgt das Ziel, die folgenden Fragen zu beantworten:

- Wie gut erfüllt das Framework die Echtzeitanforderungen industrieller Anwendungen?
- Wie hoch ist die Latenz und der Durchsatz der Modelle auf verschiedenen Embedded Systems ?
- Inwieweit wurde die Modellgröße durch die angewendeten Optimierungstechniken reduziert?
- Wie hoch ist die Auslastung der Ressourcen (CPU, Speicher) während der Modellausführung?
- Wie robust ist das Framework gegenüber variierenden Eingabebedingungen?

## 6.2 Testumgebung

Die Tests wurden auf verschiedenen Embedded Systems durchgeführt, die in typischen industriellen Anwendungen verwendet werden:

- **Speicherprogrammierbare Steuerungen (SPS)**: Diese Systeme wurden verwendet, um die Echtzeitleistung in industriellen Steuerungsumgebungen zu bewerten.
- **Industrie-PCs (IPCs)**: Die Evaluation auf IPCs ermöglichte die Analyse komplexerer Modelle mit höherer Rechenleistung.

- **Mikrocontroller**: Mikrocontroller wurden verwendet, um die Leistung des Frameworks auf Geräten mit stark eingeschränkten Ressourcen zu testen.
- **Edge-Devices**: Leistungsstarke Edge-Geräte wurden für das Training und die Vorverarbeitung der Modelle eingesetzt.

## 6.3 Bewertete Metriken

### 6.3.1 Latenz und Echtzeitfähigkeit

Die Latenz der ML-Modelle wurde gemessen, um zu überprüfen, ob die Echtzeitanforderungen erfüllt werden. Die Tests wurden für verschiedene Szenarien durchgeführt, um die folgenden Aspekte zu bewerten:

- **Durchschnittliche Latenz pro Vorhersage**: Die durchschnittliche Zeit, die das Modell benötigt, um eine Vorhersage zu treffen.
- **Maximale Latenz**: Die maximale Zeit, die für eine Vorhersage benötigt wurde, um die Einhaltung der Echtzeitanforderungen zu garantieren.
- **Jitter**: Die Varianz der Latenz bei mehreren Vorhersagen unter denselben Bedingungen.

### 6.3.2 Durchsatz

Der Durchsatz beschreibt die Anzahl der Vorhersagen, die das System pro Sekunde durchführen kann. Diese Metrik ist besonders wichtig in Szenarien, in denen das System in kurzer Zeit eine große Anzahl von Sensordaten verarbeiten muss:

- **Vorhersagen pro Sekunde (Throughput)**: Die Anzahl der pro Sekunde durchgeführten Vorhersagen.
- **Maximale Datenrate**: Die maximale Anzahl von Eingabedaten, die das System verarbeiten kann, ohne Verzögerungen zu verursachen.

### 6.3.3 Ressourcenauslastung

Die Auslastung der CPU und des Speichers wurde während der Modellausführung gemessen. Dies ist besonders wichtig, um sicherzustellen, dass das System auch bei begrenzten Ressourcen stabil und effizient arbeitet.

- **CPU-Auslastung**: Der Prozentsatz der CPU-Ressourcen, die während der Modellausführung genutzt werden.
- **Speicherverbrauch**: Der statische und dynamische Speicherverbrauch des Frameworks und der ausgeführten Modelle.

### 6.3.4 Modellgenauigkeit und Robustheit

Die Genauigkeit der optimierten Modelle wurde evaluiert, um sicherzustellen, dass die angewendeten Optimierungstechniken die Leistung nicht signifikant beeinträchtigen. Zusätzlich wurde die Robustheit des Modells gegenüber fehlerhaften oder verrauschten Eingabedaten getestet.

- **Modellgenauigkeit vor und nach der Optimierung**: Die Vorhersagegenauigkeit des Modells wurde sowohl vor als auch nach der Anwendung von Optimierungstechniken wie Quantisierung und Pruning gemessen.
- **Robustheit gegen verrauschte Eingaben**: Die Fähigkeit des Modells, auch bei verrauschten oder fehlerhaften Daten konsistente Vorhersagen zu treffen.

## 6.4 Ergebnisse und Diskussion

### 6.4.1 Latenz und Durchsatz

Die Tests ergaben, dass das Framework in der Lage ist, die vorgegebenen Echtzeitanforderungen zu erfüllen. Die Latenz blieb in den meisten Tests unter der vorgegebenen Grenze, während der Durchsatz den Anforderungen industrieller Anwendungen entsprach.

### 6.4.2 Ressourcenauslastung

Die Analyse der CPU- und Speicherauslastung zeigte, dass das Framework effizient arbeitet und die Ressourcen der verschiedenen Embedded-Geräte optimal nutzt. Insbesondere durch die Optimierungstechniken konnte der Speicherverbrauch signifikant reduziert werden.

### 6.4.3 Modellgenauigkeit und Robustheit

Die Genauigkeit der Modelle blieb nach der Optimierung weitgehend unverändert. Tests mit verrauschten Eingaben zeigten, dass das Framework robust gegenüber variierenden Eingabebedingungen ist und in den meisten Fällen konsistente Vorhersagen liefern konnte.

## 6.5 Zusammenfassung der Evaluation

Die durchgeführten Tests und die darauf basierenden Ergebnisse zeigen, dass das Framework den festgelegten Anforderungen an Latenz, Durchsatz, Ressourcenauslastung und Modellgenauigkeit entspricht. Das Framework ist in der Lage, effiziente und robuste Vorhersagen auf verschiedenen Embedded Systems zu liefern und erfüllt die Anforderungen für industrielle Echtzeitanwendungen.

# Kapitel 7

# Diskussion

In diesem Kapitel werden die wichtigsten Ergebnisse des entwickelten Frameworks sowie die Herausforderungen, die während der Implementierung und Evaluation auftraten, diskutiert. Zudem werden mögliche Verbesserungen und Erweiterungen für zukünftige Arbeiten aufgezeigt. Das Ziel der Diskussion ist es, die Leistung des Frameworks in Bezug auf die gesetzten Anforderungen zu bewerten und Schwachstellen aufzuzeigen, die in zukünftigen Iterationen adressiert werden können.

## 7.1 Wichtigste Erkenntnisse

Die Evaluation des Frameworks hat gezeigt, dass die gewählten Optimierungstechniken und das Designprinzip einer modularen Architektur erfolgreich umgesetzt wurden. Die wesentlichen Erkenntnisse lassen sich wie folgt zusammenfassen:

### 7.1.1 Erfüllung der Echtzeitanforderungen

Einer der zentralen Anforderungen war die Sicherstellung der Echtzeitfähigkeit, insbesondere für Anwendungen in der industriellen Fertigung. Die Evaluation hat gezeigt, dass das Framework in der Lage ist, die vorgegebenen Latenzanforderungen zu erfüllen. Durch den Einsatz von Optimierungstechniken wie Quantisierung und Pruning konnte die Modellgröße signifikant reduziert werden, was zu einer Verbesserung der Vorhersagegeschwindigkeit führte. Auch bei ressourcenbeschränkten Geräten, wie Mikrocontrollern und SPS-Systemen, blieb die Latenz innerhalb der tolerierbaren Grenzen.

### 7.1.2 Ressourceneffizienz und Speicherverbrauch

Ein weiteres wichtiges Ziel war die Reduktion des Speicherverbrauchs, um die Modelle auf Geräten mit begrenztem Speicher, wie SPS und Mikrocontrollern, betreiben zu können. Die eingesetzten Optimierungen, insbesondere die Quantisierung, zeigten deutliche Verbesserungen. Die Speichernutzung konnte um bis zu 70% reduziert werden, ohne die Modellgenauigkeit signifikant zu beeinträchtigen. Dies ermöglicht den Einsatz von komplexeren Modellen selbst auf stark ressourcenbeschränkten Geräten.

### 7.1.3 Modellgenauigkeit und Robustheit

Die Vorhersagegenauigkeit der Modelle blieb trotz der angewandten Optimierungen weitgehend erhalten. Es wurde jedoch festgestellt, dass einige Modelle nach der Quantisierung eine leichte Abnahme der Genauigkeit zeigten. Dies war insbesondere bei Modellen der Fall, die nicht für Quantization-Aware Training (QAT) vorbereitet waren. Dennoch war die Abnahme der Genauigkeit in den meisten Fällen akzeptabel, insbesondere im Kontext von industriellen Anwendungen, in denen eine geringfügige Abweichung oft toleriert werden kann. Die Robustheit des Frameworks gegen verrauschte Eingabedaten wurde ebenfalls bestätigt, was für den Einsatz in Echtzeitsystemen von entscheidender Bedeutung ist.

## 7.2 Herausforderungen und Lessons Learned

Während der Entwicklung und Implementierung des Frameworks traten verschiedene Herausforderungen auf, die im Folgenden diskutiert werden:

### 7.2.1 Hardware-Einschränkungen

Eine der größten Herausforderungen war die Anpassung des Frameworks an die unterschiedlichen Ressourcenanforderungen der Zielhardware. Mikrocontroller und SPS-Systeme haben im Vergleich zu IPCs und Edge-Geräten stark eingeschränkte Rechenleistung und Speicher. Insbesondere die Modellgröße und die Laufzeitleistung mussten durch Optimierungstechniken drastisch verbessert werden, um die Modelle auf diesen Geräten lauffähig zu machen. Obwohl Techniken wie Quantisierung und Pruning hilfreich waren, war es eine Herausforderung, das richtige Gleichgewicht zwischen Modellgröße und Genauigkeit zu finden.

### 7.2.2 Modelloptimierung ohne Genauigkeitsverlust

Obwohl Techniken wie Pruning und Quantisierung den Speicher- und Rechenaufwand erheblich reduzierten, führten sie in einigen Fällen zu einem gewissen Genauigkeitsverlust. Vor allem Modelle, die nicht für Quantization-Aware Training trainiert wurden, zeigten eine Verschlechterung der Vorhersagequalität nach der Quantisierung. Ein weiteres Problem war, dass Pruning zwar die Modellgröße reduziert, aber in einigen Fällen die Ausführungszeit durch unstrukturierte Gewichtsverteilung nicht wie erwartet verkürzte. Dies zeigt, dass es entscheidend ist, Optimierungstechniken gezielt einzusetzen, je nach Modell und Anwendungsfall.

### 7.2.3 Integration in Echtzeitsysteme

Die Integration des Frameworks in industrielle Echtzeitsysteme stellte ebenfalls eine Herausforderung dar. Echtzeitsysteme erfordern eine zuverlässige und deterministische Ausführung der ML-Modelle. Das Erreichen dieser deterministischen Ausführung bei ressourcenbeschränkten Geräten war insbesondere bei der Anpassung von nicht-zeitkritischen Modellen eine Herausforderung. Die Priorisierung von Aufgaben und die Überwachung der Laufzeit waren notwendig, um sicherzustellen, dass die Echtzeitanforderungen erfüllt wurden.

## 7.3 Mögliche Verbesserungen und zukünftige Arbeiten

Obwohl das Framework erfolgreich implementiert und evaluiert wurde, gibt es mehrere Bereiche, in denen zukünftige Arbeiten Verbesserungen oder Erweiterungen vornehmen könnten:

### 7.3.1 Optimierte Unterstützung für unterschiedliche Hardware

Während das Framework bereits auf verschiedenen Hardwareplattformen erfolgreich ausgeführt werden konnte, könnten zukünftige Versionen eine noch gezieltere Optimierung für spezifische Hardware bieten. Beispielsweise könnten benutzerdefinierte Optimierungsstrategien für Mikrocontroller oder SPS-Systeme implementiert werden, um die Leistung weiter zu verbessern. Zudem könnte die Unterstützung für neue Plattformen, wie spezialisierte Edge-Geräte oder FPGAs, ausgebaut werden.

### 7.3.2 Weiterentwicklungen bei der Modelloptimierung

Ein weiterer Bereich für zukünftige Verbesserungen ist die Verfeinerung der Modelloptimierungstechniken. Der Einsatz von Quantization-Aware Training (QAT) könnte ausgeweitet werden, um sicherzustellen, dass Modelle bei der Quantisierung weniger Genauigkeitsverluste erleiden. Auch neue Techniken zur Modellkompression und -optimierung, wie distillierte Modelle oder neuronale Architektur-Suche (NAS), könnten untersucht und implementiert werden.

### 7.3.3 Verbesserte Task-Priorisierung und Scheduling

Die Priorisierung von Aufgaben und die Verwaltung von Ressourcen in Echtzeitsystemen könnte weiter verbessert werden, um eine effizientere Nutzung der Hardware zu gewährleisten. Eine dynamische Anpassung der Aufgabenprioritäten in Abhängigkeit von der aktuellen Systemauslastung oder externen Faktoren könnte zu einer besseren Einhaltung der Echtzeitanforderungen führen. Auch die Integration von Machine-Learning-Modellen zur Vorhersage von Systemlasten könnte untersucht werden.

### 7.3.4 Erweiterung der Modellunterstützung

Während das Framework derzeit eine breite Palette von ML-Modellen unterstützt, könnte die Erweiterung auf neue Modelltypen oder Frameworks die Flexibilität erhöhen. Dies könnte beispielsweise die Integration von Modellen beinhalten, die für spezielle industrielle Anwendungen wie Anomalieerkennung oder Vorhersagemodellierung optimiert sind.

## 7.4 Zusammenfassung der Diskussion

Die Implementierung und Evaluation des Frameworks hat gezeigt, dass es möglich ist, Machine-Learning-Modelle erfolgreich auf ressourcenbeschränkten Embedded-Systemen einzusetzen. Obwohl verschiedene Herausforderungen, wie die Anpassung an unterschiedliche Hardware und der Umgang mit Optimierungstechniken, zu bewältigen waren, konnte das Framework die wichtigsten Anforderungen an Echtzeitfähig-

keit und Ressourceneffizienz erfüllen. Zukünftige Arbeiten könnten darauf abzielen, die Modelloptimierung weiter zu verfeinern, die Hardwareunterstützung zu erweitern und die Effizienz in Echtzeitsystemen weiter zu verbessern.

# Kapitel 8

# Fazit

Die vorliegende Arbeit befasst sich mit der Entwicklung eines Frameworks zur effizienten Ausführung und Optimierung von Machine Learning Modellen auf ressourcenbeschränkten Embedded Systems . Ziel war es, ein flexibles, anpassungsfähiges und ressourceneffizientes Framework zu entwickeln, das die Anforderungen industrieller Anwendungen erfüllt, insbesondere im Hinblick auf Echtzeitleistung und Speicheroptimierung.

## 8.1 Zusammenfassung der Beiträge

Im Rahmen dieser Thesis wurden mehrere zentrale Beiträge geleistet:

### 8.1.1 Entwicklung eines modularen Frameworks

Ein modular aufgebautes Framework wurde entwickelt, das es ermöglicht, ML-Modelle flexibel auf Embedded- und Edge-Geräten einzusetzen. Die Architektur des Frameworks wurde so gestaltet, dass verschiedene ML-Modelle integriert und optimiert werden können, um spezifische Anforderungen ressourcenbeschränkter Umgebungen zu erfüllen. Die modulare Struktur erleichtert die Erweiterbarkeit und Wartbarkeit des Frameworks.

### 8.1.2 Anwendung von Optimierungstechniken für Embedded-ML

Das Framework unterstützt fortgeschrittene Optimierungstechniken wie Quantisierung, Pruning und Modellkompression, um ML-Modelle für Embedded-Geräte mit begrenzten Ressourcen zu adaptieren. Diese Techniken haben sich als effektiv erwiesen, um den Speicherbedarf und die Ausführungszeit der Modelle zu reduzieren, während die Genauigkeit der Vorhersagen weitgehend erhalten bleibt. Dies stellt sicher, dass auch ressourcenintensive Modelle auf Geräten mit eingeschränkten Rechenkapazitäten eingesetzt werden können.

### 8.1.3 Erfüllung von Echtzeitanforderungen in industriellen Anwendungen

Ein wesentlicher Schwerpunkt dieser Arbeit lag auf der Sicherstellung der Echtzeitleistung der ML-Modelle. Die durchgeführte Evaluation hat gezeigt, dass die entwickelten Optimierungen die Echtzeitanforderungen in verschiedenen industriellen Szenarien erfüllen können. Durch gezielte Priorisierung und effiziente Ausführung der Modelle wurde eine geringe Latenz und eine hohe Vorhersagegeschwindigkeit erreicht, was das Framework für den Einsatz in industriellen Echtzeitsystemen qualifiziert.

### 8.1.4 Breite Unterstützung von Embedded- und Edge-Plattformen

Das Framework wurde für eine Vielzahl von Embedded- und Edge-Geräten optimiert, darunter Speicherprogrammierbare Steuerung (SPS), Industrie-PC (IPCs), Mikrocontroller und leistungsstarke Edge-Devices. Diese breite Unterstützung ermöglicht es, das Framework in verschiedenen industriellen Umgebungen einzusetzen, von stark ressourcenbeschränkten Geräten bis hin zu leistungsfähigeren Systemen.

## 8.2 Einschränkungen der Arbeit

Trotz der erzielten Erfolge gibt es einige Einschränkungen, die in zukünftigen Arbeiten adressiert werden können. Dazu gehören unter anderem:

- Die Modelloptimierungstechniken wie Quantisierung und Pruning führten in einigen Fällen zu einem geringen Genauigkeitsverlust, der in besonders sensiblen Anwendungen problematisch sein könnte.
- Die Integration des Frameworks auf sehr spezifische oder hochgradig spezialisierte Hardwareplattformen (z.B. FPGAs oder neuere spezialisierte Edge-Chips) wurde nicht im vollen Umfang untersucht.
- Eine umfassende Integration von adaptiven Lerntechniken, die Modelle automatisch basierend auf der Systemauslastung oder Umweltbedingungen optimieren, könnte zukünftige Arbeiten bereichern.

## 8.3 Ausblick und zukünftige Arbeiten

Basierend auf den Erkenntnissen dieser Arbeit gibt es mehrere interessante Ansätze für zukünftige Forschungs- und Entwicklungsarbeiten:

### 8.3.1 Weiterentwicklung der Optimierungstechniken

Zukünftige Arbeiten könnten sich auf die Verbesserung der Optimierungstechniken konzentrieren, insbesondere auf die Anwendung von Techniken wie Quantization-Aware Training (QAT), um die Genauigkeitsverluste nach der Quantisierung zu minimieren. Zudem könnten neue Ansätze zur Modellkompression und -optimierung untersucht werden, um noch bessere Ergebnisse auf extrem ressourcenbeschränkten Geräten zu erzielen.

### 8.3.2 Erweiterung auf neue Hardwareplattformen

Die Unterstützung für zusätzliche Hardwareplattformen, wie spezialisierte Edge-Prozessoren oder FPGAs, könnte die Anwendbarkeit des Frameworks weiter erhöhen. Besonders die Möglichkeit, ML-Modelle auf energieeffizienteren und leistungsfähigeren spezialisierten Prozessoren auszuführen, könnte die Reichweite des Frameworks vergrößern.

### 8.3.3 Adaptive Modelle und dynamisches Task-Management

Ein weiteres spannendes Forschungsfeld ist die Implementierung von adaptiven Modellen, die ihre Ressourcenanforderungen dynamisch an die verfügbaren Systemressourcen anpassen. Solche Modelle könnten in Echtzeitsystemen von großem Nutzen sein, insbesondere in Situationen, in denen sich die Systemauslastung oder Umweltbedingungen ändern.

## 8.4 Abschließende Bemerkungen

Die in dieser Arbeit vorgestellten Methoden und Ansätze haben gezeigt, dass es möglich ist, Machine-Learning-Modelle effizient auf Embedded- und Edge-Systemen einzusetzen, die nur über begrenzte Ressourcen verfügen. Die Ergebnisse bestätigen, dass durch den Einsatz von Modelloptimierungstechniken und einer flexiblen Framework-Architektur die Hürden für den Einsatz von KI in industriellen Echtzeitanwendungen gesenkt werden können. Das Framework bietet eine solide Grundlage für zukünftige Entwicklungen und kann in verschiedenen industriellen Umgebungen zur Optimierung von Produktionsprozessen und zur Automatisierung eingesetzt werden. Zukünftige Arbeiten werden auf den hier vorgestellten Ergebnissen aufbauen und neue Optimierungsansätze sowie eine erweiterte Hardwareunterstützung integrieren.

# Anhang A

# Zusätzliche Daten und Informationen

In diesem Anhang werden zusätzliche Informationen bereitgestellt, die für die Ausarbeitung relevant sind, aber den Lesefluss im Hauptteil unterbrechen würden.

## A.1 Tabellen und Diagramme

Hier könnten detaillierte Tabellen oder Diagramme stehen, die im Haupttext nur kurz erwähnt wurden.

| Kriterium 1 | Kriterium 2 | Kriterium 3 |
|---|---|---|
| Wert 1 | Wert 2 | Wert 3 |
| Wert 4 | Wert 5 | Wert 6 |

Tabelle A.1: Eine detaillierte Tabelle mit zusätzlichen Daten.

## A.2 Codebeispiele

Hier könntest du Codebeispiele hinzufügen, die im Haupttext aus Platzgründen nur kurz erwähnt wurden.

```
# Beispielcode in Python
def beispiel_funktion():
    print("Dies ist ein Beispielcode.")
```

## A.3 Zusätzliche Erläuterungen

Falls nötig, kannst du hier zusätzliche Erklärungen oder Hintergrundinformationen bereitstellen.

# Literaturverzeichnis


[1] Amin Biglari and Wei Tang. A review of embedded machine learning based on hardware, application, and sensing scheme. *Sensors*, 23(4), 2023.

[2] Nerea Gómez Larrakoetxea, Joseba Eskubi Astobiza, Iker Pastor López, Borja Sanz Urquijo, Jon García Barruetabeña, and Agustin Zubillaga Rego. Efficient machine learning on edge computing through data compression techniques. *IEEE Access*, 11:31676–31685, 2023.

[3] Jungme Park, Pawan Aryal, Sai Mandumula, and Ritwik Asolkar. An optimized dnn model for real-time inferencing on an embedded device. *Sensors*, 23:3992, 04 2023.

[4] Itilekha Podder, Tamas Fischl, and Udo Bub. Artificial intelligence applications for mems-based sensors and manufacturing process optimization. *Telecom*, 4(1):165–197, 2023.

[5] Satyam, P. Geetha, K.S. Shashikala, and N. Ashok Kumar. Ai-enabled edge computing models: Trends, developments, and future implications. In *2023 2nd International Conference on Edge Computing and Applications (ICECAA)*, pages 63–67, July 2023.

[6] John Soldatos, editor. *Artificial Intelligence in Manufacturing: Enabling Intelligent, Flexible and Cost-Effective Production Through AI*. Springer Cham, 1 edition, 2024.

[7] Xifan Yao, Jiajun Zhou, Jiangming Zhang, and Claudio R. Boër. From intelligent manufacturing to smart manufacturing for industry 4.0 driven by next generation artificial intelligence and further on. In *2017 5th International Conference on Enterprise Systems (ES)*, pages 311–318, 2017.

[8] Dilara Yesilbas. *Deep Learning in Automatic Optical Inspection - Selected Studies in the Context of an Automotive Supplier*. PhD thesis, 01 2022.

[9] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. Adaptive quantization for deep neural network. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32, 12 2017.

[10] Ramazan Ünlü and İsmet Söylemez. *AI-Driven Predictive Maintenance*, pages 207–233. 09 2024.