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

Sebastian Munoz Segovia

31. 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** [35] bis hin zur **Qualitätskontrolle** [30] bietet Machine Learning zahlreiche Möglichkeiten zur Effizienzsteigerung und Kostensenkung. Allerdings sind viele bestehende Machine-Learning-Modelle nicht für den Einsatz auf Embedded Systemen optimiert [20]. 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 [6]. Zahlreiche Studien betonen die Vorteile von Edge-Computing in industriellen Anwendungen, insbesondere im Hinblick auf **Echtzeitanalysen und datengesteuerte Entscheidungsprozesse** [23]. 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

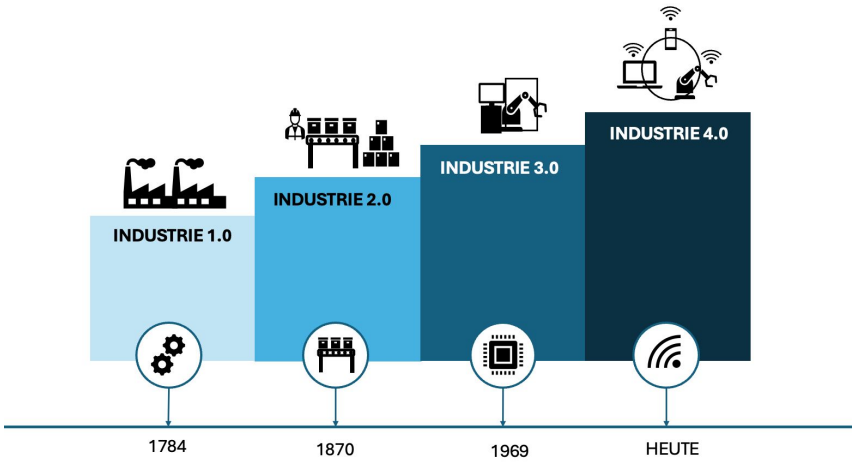

Abbildung 1.1: Die vier industriellen Revolutionen: Mechanisierung, Massenproduktion, Automatisierung, Vernetzte Systeme.

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 [16], Büchern [24] und Konferenzberichten [29] 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

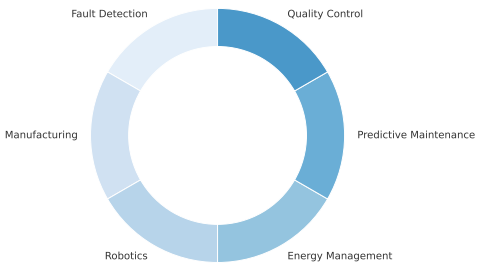

Abbildung 2.1: Anwendungen für Embedded Sytems in der Industrie 4.0

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 [21] 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.

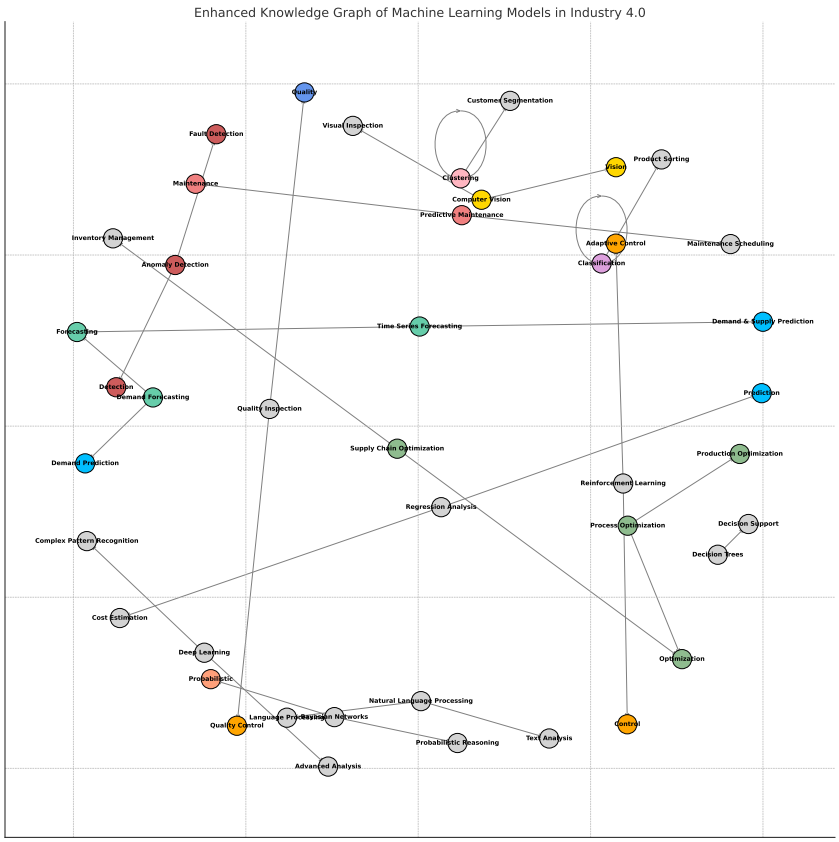

Abbildung 2.2: ML Modelle und ihre Anwendungen in der Industrie 4.0

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) [34] 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

In 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** [7] sowie die Auswahl der **Entwicklungsumgebung**, die verwendeten **Optimierungstechniken** [32] 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, sind 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 sind 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 [33].
2. Festlegung der Suchstrategie: Relevante wissenschaftliche Datenbanken wie IEEE Xplore [33], ACM Digital Library [1] und SpringerLink [25]sind 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, sind die Suchergebnisse eingegrenzt. Insgesamt sind 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**: Anforderungen sind priorisiert, um sicherzustellen, dass die wichtigsten Kriterien wie Echtzeitanforderungen und Effizienz bei der Entwicklung des Frameworks berücksichtigt sind.

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 sind in Zusammenarbeit mit Industriepartnern und durch eine umfassende Literaturrecherche zu bestehenden Lösungen festgelegt. Hierbei sind 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 sind. 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 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 sind. 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 [22] 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 [26]. TensorFlow Lite wurde speziell entwickelt, um TensorFlow-Modelle für ressourcenbeschränkte Geräte zu optimieren und diese auf Embedded-Systemen lauffähig zu machen[19].
- **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 [27].

- **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[8].
- **LightGBM**: LightGBM bietet eine ähnliche Funktionalität wie XGBoost, ist jedoch für größere Datensätze und schnellere Trainingszeiten optimiert. Es wird häufig in Echtzeitanwendungen verwendet [13].
- **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 sind[5].
- **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[31].
- **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 [17].

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[2].

- **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 sind, 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 sind, um neuronale Netze oder klassische Modelle, die in anderen Frameworks entwickelt wurden, auf Embedded-Systemen bereitzustellen.

Diese Programmiersprachen und Tools bieten wertvolle Möglichkeiten für industrielle Anwendungen im Bereich der künstlichen Intelligenz und Embedded-Systeme, mit Schwerpunkten auf Effizienz, Flexibilität und Ressourcenschonung.

3.4 Tools für Modelloptimierung, Konvertierung und Deployment

Zur Optimierung, Konvertierung und Bereitstellung von ML-Modellen auf Embedded Systems stehen verschiedene spezialisierte Tools zur Verfügung. Diese Werkzeuge unterstützen die effiziente Ausführung auf ressourcenbeschränkten Geräten und bieten flexible Optionen für die Integration in Produktionsumgebungen.

TensorFlow Lite dient als leichtgewichtige Version von TensorFlow, entwickelt für den Einsatz auf mobilen und eingebetteten Geräten. Mit Techniken wie Quantisierung und Pruning reduziert es Modellgröße und Rechenaufwand, was den Einsatz in Echtzeit- und Embedded-Anwendungen ermöglicht.

ONNX Runtime erlaubt die plattformübergreifende Ausführung von ML-Modellen, die in Frameworks wie TensorFlow, PyTorch oder scikit-learn entwickelt wurden. Es optimiert Modelle für eine Vielzahl von Hardwareplattformen und ermöglicht ein effizientes Deployment auf Edge- und Embedded-Systemen.

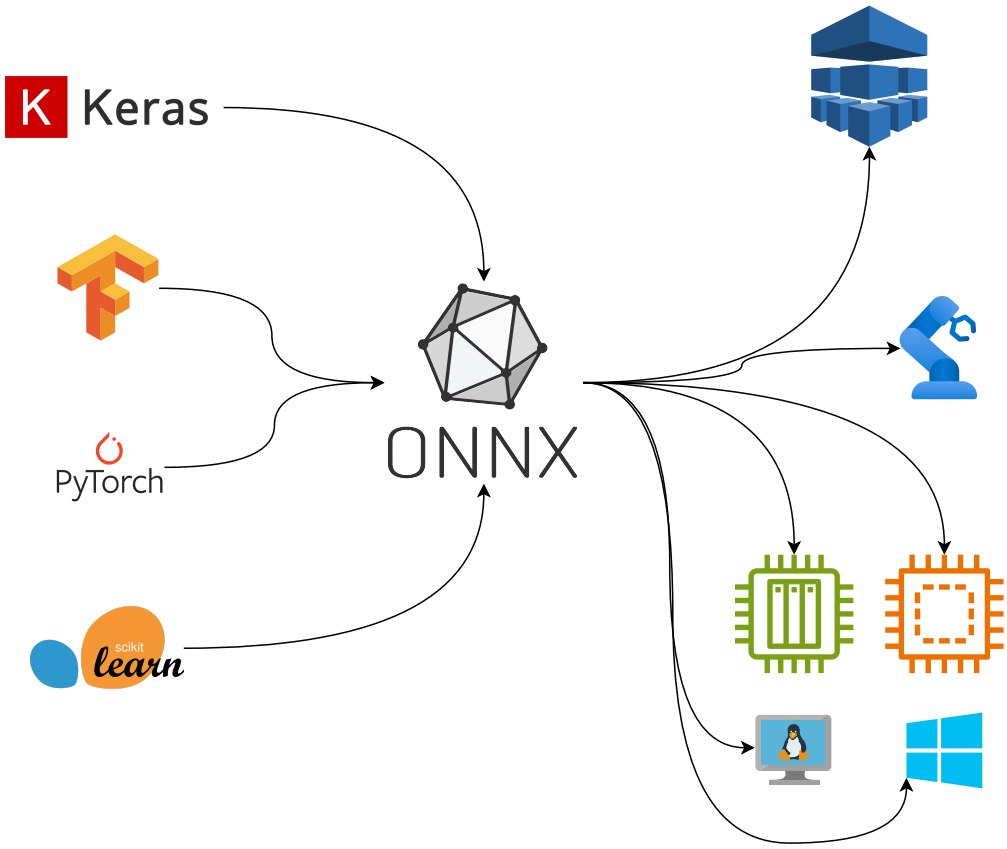

Abbildung 3.1: ONNX Runtime für plattformübergreifende ML-Modelle

Apache TVM ist ein Compiler-Framework, das ML-Modelle für diverse Hardwarearchitekturen optimiert und speziell an die Anforderungen von Embedded-Geräten anpasst. Es unterstützt die effiziente Ausführung auf Mikrocontrollern, GPUs und Edge-Computing-Plattformen.

TinyML-Frameworks wie TensorFlow Micro und uTensor sind darauf ausgelegt, ML-Modelle auf extrem ressourcenbeschränkten Geräten wie Mikrocontrollern auszuführen. Diese Frameworks ermöglichen

die Implementierung auf Kleinstgeräten ohne Überlastung der Rechenressourcen.

TensorRT von NVIDIA ist ein High-Performance-Optimizer für neuronale Netze und steigert die Inferenzleistung auf NVIDIA-Plattformen wie dem Jetson Nano. TensorRT eignet sich besonders für Echtzeitanwendungen auf leistungsstarker Edge-Hardware.

Docker und **K3s** vereinfachen das Deployment von Modellen durch Containerisierung und Orchestrierung auf Edge- und IoT-Geräten. Docker verpackt Modelle und Abhängigkeiten in Containern für ein konsistentes Deployment, während K3s als leichtgewichtige Kubernetes-Distribution die Verwaltung von ML-Diensten in Edge-Computing-Umgebungen unterstützt.

MLflow bietet eine umfassende Verwaltung des gesamten ML-Modelllebenszyklus. Es unterstützt das Tracking, die Versionierung und das Deployment, was für die Überwachung und Pflege mehrerer Modellversionen in produktiven Systemen vorteilhaft ist.

PyInstaller konvertiert Python-Anwendungen und ML-Modelle in ausführbare Dateien und ermöglicht deren Ausführung auf Geräten ohne Python-Laufzeitumgebung, was das Deployment als eigenständige Anwendung oder Windows-Dienst unterstützt.

Zephyr RTOS stellt ein Echtzeitbetriebssystem bereit, das für die Ausführung von ML-Modellen auf Mikrocontrollern optimiert ist und die Echtzeitanforderungen der industriellen Fertigung erfüllt.

Zusammen bieten **Conda** und **Pip** eine umfassende Verwaltung der Modellabhängigkeiten. Conda eignet sich für umfangreiche Bibliotheken wie TensorFlow und PyTorch, da es eine effizientere Verwaltung und Integration großer Bibliotheken ermöglicht. Pip wird für spezialisierte Bibliotheken verwendet, die nicht über Conda verfügbar sind, und ergänzt die Abhängigkeitsverwaltung flexibel.

Diese Tools und Techniken tragen dazu bei, ein Framework zu entwickeln, das ML-Modelle auf Embedded Systems effizient implementiert und die Anforderungen moderner industrieller Anwendungen erfüllt.

3.5 Optimierungstechniken

Die Optimierung von ML-Modellen für Embedded Systems stellt einen wesentlichen Aspekt dieses Projekts dar. Im Folgenden sind die implementierten Techniken beschrieben:

Quantisierung reduziert die Größe eines ML-Modells, indem die Genauigkeit der Gewichtungen von 32-Bit-Gleitkommazahlen auf 8-Bit-Ganzzahlen verringert wird [19]. Dies führt zu einer erheblichen Senkung des Speicherbedarfs und ermöglicht die Ausführung der Modelle auf Geräten mit begrenzten Ressourcen. Diese Technik wird insbesondere bei Modellen eingesetzt, die auf speicher- und rechenkapazitätsbeschränkten Systemen wie SPS und IPCs verwendet sind.

Beim **Model Pruning** sind nicht benötigte Neuronen und Verbindungen aus dem ML-Modell entfernt, um dessen Größe zu reduzieren, ohne die Genauigkeit signifikant zu beeinträchtigen. Diese Technik verbessert die Ausführungsgeschwindigkeit auf Embedded Systems, während die Reduktion der Modellgenauigkeit minimal bleibt [28].

Neben den oben genannten Techniken kommen speziell für Embedded Systems optimierte Algorithmen zur Anwendung. Diese Algorithmen sind auf minimalen Rechenaufwand ausgelegt und ermöglichen gleichzeitig Echtzeitentscheidungen. Ein Beispiel hierfür sind **binäre Entscheidungsbäume** und lineare Mo-

delle, die weniger komplex als neuronale Netzwerke sind, jedoch in vielen industriellen Anwendungen eine ausreichende Leistungsfähigkeit bieten.

3.6 Implementierungsschritte

3.6.1 Anforderungsanalyse

Zunächst erfolgt die Erfassung der Anforderungen der industriellen Partner und die Durchführung einer Anforderungsanalyse. Die zentralen Anforderungen fokussieren sich auf die Reduktion der Modellgröße, die Sicherstellung der Echtzeitfähigkeit sowie eine ressourcenschonende Ausführung auf Embedded-Devices. Besonders berücksichtigt sind die spezifischen Hardware-Einschränkungen der eingesetzten SPS- und IPC-Systeme. Zusätzlich sind Zielfunktionen und Prioritäten festgelegt, um eine Balance zwischen **Modellkomplexität**, **Laufzeitleistung** und **Speicherverbrauch** zu erreichen.

Abbildung 3.2: Beckoff IPCs - C60xx | Ultra-compact Industrial PCs[4]

3.6.2 Entwicklung des Frameworks

Das Framework wird als flexible Schnittstelle zwischen den optimierten ML-Modellen und den Embedded Systems entworfen. Die Architektur gliedert sich in verschiedene Module, die sowohl die Modelloptimierung als auch die Kommunikation zwischen den Geräten unterstützen. Die wesentlichen Entwicklungsaspekte umfassen:

- **Architekturentwurf**: Die Architektur ist modular aufgebaut, um die Integration und den Austausch verschiedener ML-Modelle zu erleichtern.
- **Modelloptimierungstechniken**: Techniken wie Quantisierung, Pruning und Kompression sind eingesetzt, um die Modellgröße zu verringern und die Ausführungseffizienz zu steigern.
- **API-Design**: Eine flexible API ermöglicht den Zugriff auf verschiedene ML-Modelle und gewährleistet eine unkomplizierte Bereitstellung auf unterschiedlichen Embedded Devices.

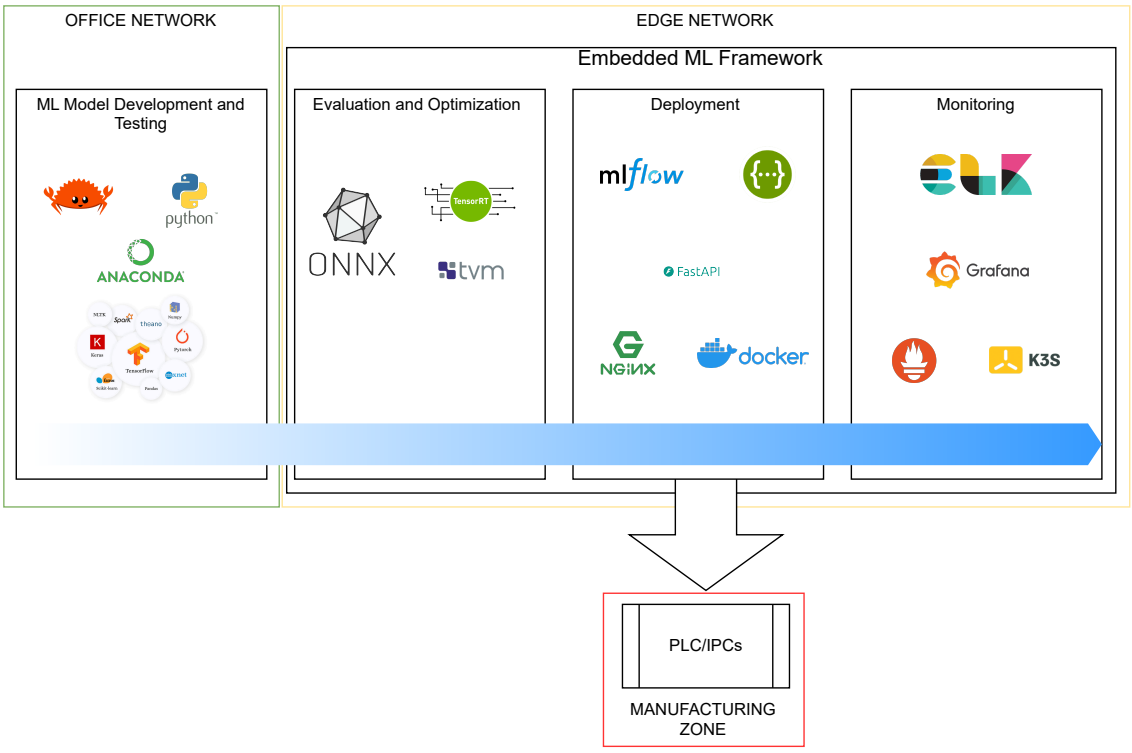

Abbildung 3.3: Architektur des Frameworks

3.6.3 Integration und Tests

Das Framework wird auf verschiedenen Embedded Devices, einschließlich SPS und IPCs, integriert und umfassend getestet. Dieser Schritt beinhaltet:

- **Hardware-Integration**: Die Anpassung des Frameworks an spezifische Hardware-Eigenschaften unterschiedlicher Embedded-Plattformen.
- **Unit-Tests**: Unit-Tests sichern die korrekte Funktionalität der zentralen Komponenten ab.
- **Hardware-in-the-Loop (HIL) Tests**: HIL-Tests [3, 15] überprüfen das Framework unter realen Bedingungen und stellen sicher, dass die Modelle in Echtzeit mit der Hardware interagieren.
- **Laufzeitleistungsprüfung**: Benchmarks zur Messung der Ausführungszeiten sind definiert, um die Erfüllung der Echtzeitanforderungen industrieller Anwendungen zu gewährleisten.

3.6.4 Evaluation der Ergebnisse

Nach der Integration wird das Framework anhand vordefinierter Leistungsmetriken evaluiert:

- **Leistungsmetriken**: Die wichtigsten Metriken umfassen die Ausführungszeit, den Speicherverbrauch und den Energieverbrauch auf den Embedded Devices.
- **Vergleich mit den Anforderungen**: Die Ergebnisse vergleichen mit den in der Anforderungsanalyse definierten Zielwerten, um die Erfüllung der Anforderungen zu validieren.

- **Robustheit und Fehlerbehandlung**: Tests bewerten die Robustheit des Frameworks und dessen Fähigkeit zur Fehlerbewältigung, beispielsweise bei Hardwareausfällen oder Modellfehlern.
- **Optimierungspotential**: Basierend auf den Evaluierungsergebnissen wird weiteres Optimierungspotenzial, wie die Reduktion des Energieverbrauchs oder die Verbesserung der Ausführungszeiten, identifiziert.

Tabelle 3.1: Beispiel evaluationsergebnisse des Frameworks

Metrik	**Gemessener Wert**	**Zielwert**	**Erfüllungsgrad**	**Opt.Potential**
Ausführungszeit	120 ms	$\leq$ 150 ms	Erfüllt	Gering
Speicherverbrauch	512 MB	$\leq$ 500 MB	Teilerfüllt	Mittel
Energieverbrauch	2.5 W	$\leq$ 2 W	Nicht erfüllt	Hoch
Robustheit	95% Fehlerfrei	$\geq$ 99%	Teilerfüllt	Hoch
Fehlerbehandlungskapazität	100% Erfolgreich	100%	Erfüllt	Gering

3.6.5 Zusätzliche Anforderungen

Weitere Themen die zu beachten sind:

- **Sicherheit und Datenschutz**: Das Framework erfüllt die Datensicherheitsanforderungen der industriellen Fertigung und stellt sicher, dass sensible Informationen geschützt und gemäß den branchenspezifischen Richtlinien verarbeitet sind.
- **Flexibilität und Erweiterbarkeit**: Das Framework ist modular und flexibel gestaltet, sodass zukünftige Erweiterungen und die Integration neuer ML-Modelle ohne umfassende Anpassungen möglich sind. Diese Struktur ermöglicht eine hohe Anpassungsfähigkeit an wechselnde industrielle Anforderungen.
- **Wartung und Updates**: Implementierte Mechanismen zur Wartung und für Modell-Updates gewährleisten die langfristige Nutzbarkeit und Anpassungsfähigkeit des Frameworks. Durch ein robustes Update-Management können Aktualisierungen und Fehlerbehebungen effizient durchgeführt sind, was die Zuverlässigkeit des Systems steigert.

Tabelle 3.2: Andere Systemvoraussetzungen

Systemvoraussetzungen	**Erläuterung**
Cloud Computing	Bereitstellung der leistungsfähigen IT-Infrastruktur für aufwendige Berechnungen, Datenspeicherung und Ausführung über das Internet bezogener Softwareanwendungen.
Sensorik	Technologien zur Wahrnehmung, Messung und Kontrolle von Veränderungen in der Umgebung oder in einem technischen System.
Edge Computing	IT-Infrastrukturen, die eine dezentralisierte Datenspeicherung und -verarbeitung realisieren.
Datenhoheit, -zugang	Rechtliche Regulierungen, die die Nutzung, Zugang und Verwertung von Daten bestimmen.

Datenqualität	Vollständigkeit der Daten, einheitliche Formatierung oder Detaillierungsgrad, unterbrechungsfreie Datenaufnahmen.
Robustheit der Algorithmen (Reaktion auf unerwartete Situationen)	Zuverlässigkeit der Ergebnisse der Datenanalyse, Reaktion auf Ausreißer und unerwartete Situationen.
Trainingsdauer und Aufwand für Algorithmen	Zeit- und Personalaufwand für eine zuverlässige Erkennung von Objekten oder Handlungen, die durch das Sammeln und Annotieren von Trainingsdaten entsteht.
Rechenaufwand/Energieeffizien	Aufwendige Berechnungen der Bewegungsabläufe während Aktionsplanung, Prozess- oder Parameteroptimierung unter Berücksichtigung von mehreren Optimierungskriterien.
IT-Sicherheit (Security)	Schutz von sensiblen Unternehmensdaten vor Cyberangriffen.
Funktionale Sicherheit (Safety)	Zuverlässiger und sicherer Betrieb von industriellen Systemen, Minimierung von Verletzungsrisiken und Personenschäden.
Interoperabilität der KI-Verfahren und Daten	Übertragbarkeit der KI-Verfahren auf andere Datenquellen, Einheitlichkeit der Datenmodelle und Formate für die Wiederverwendung der Daten und Algorithmen.

3.7 Zusammenfassung

Das Framework basiert auf einer umfassenden Anforderungsanalyse, ist modular und für den Betrieb auf Embedded Devices optimiert. Die effiziente Ausführung auf ressourcenbeschränkten Systemen wird durch den Einsatz spezifischer Optimierungstechniken, wie Quantisierung und Pruning, unterstützt. Um die Leistungsfähigkeit des Frameworks zu überprüfen, sind detaillierte Tests durchgeführt, die bestätigen, dass Echtzeitfähigkeit und Robustheit den festgelegten Anforderungen entsprechen.

Nach Evaluierung des Frameworks sind zusätzliche Optimierungsmöglichkeiten identifiziert. Diese Verbesserungsbereiche bilden die Grundlage für zukünftige Weiterentwicklungen. Zusammenfassend ermöglicht das Framework eine sichere, flexible und performante Implementierung von ML-Modellen auf industriellen Embedded Devices, wobei die Anpassungsfähigkeit an neue Anforderungen und eine einfache Wartung gewährleistet sind.

Kapitel 4

Entwicklung des Frameworks

In diesem Kapitel werden die Architektur und die Entwicklung des Frameworks beschrieben, das auf die Optimierung von ML-Modellen für ressourcenbeschränkte Umgebungen wie Embedded und Edge-Devices ausgerichtet ist. Die Konzeption des Frameworks basiert auf einer umfassenden Literaturrecherche und Anforderungsanalyse, die den wissenschaftlichen und praktischen Hintergrund zur Gestaltung des Systems liefert.

Die **Literaturrecherche** identifiziert bestehende Ansätze und Techniken zur Optimierung von ML-Modellen in Embedded Systems. Zunächst werden zentrale Forschungsfragen formuliert, die die Herausforderungen und potenziellen Lösungen für ressourceneffiziente ML-Modelle in industriellen Anwendungen adressieren. In renommierten wissenschaftlichen Datenbanken wie IEEE Xplore [14], ACM Digital Library [9] und SpringerLink [11] erfolgt eine strukturierte Suche mit Begriffen wie „Embedded Systems“, „Machine Learning Optimierung“, „Edge Computing“ und „Echtzeitanforderungen“. Die Inklusions- und Exklusionskriterien gewährleisten die Relevanz der Artikel; insgesamt werden 50 Artikel analysiert, von denen 30 als besonders geeignet für die Konzeption des Frameworks eingestuft werden. Die in der Literatur identifizierten Ansätze werden nach Relevanz bewertet und bilden die Grundlage für die zentralen Anforderungen und Architekturprinzipien des Frameworks.

Ergänzend zur Literaturrecherche erfolgt eine **Anforderungsanalyse**, um die spezifischen Bedürfnisse der industriellen Partner zu berücksichtigen. In Interviews und Workshops mit Partnern, die Embedded Systems in der Produktion einsetzen, werden Anforderungen wie Rechenleistung, Echtzeitfähigkeit und Modellflexibilität dokumentiert. Zudem werden technische Spezifikationen der eingesetzten Hardware (z. B. SPS und IPC) untersucht, um spezifische Hardwareeinschränkungen und Schnittstellenanforderungen zu berücksichtigen. Die Anforderungen werden in technische und funktionale Anforderungen kategorisiert und priorisiert, um eine Balance zwischen Modellkomplexität, Laufzeitleistung und Speicherverbrauch zu erreichen.

Auf Basis dieser Ergebnisse ist das Framework als **modulares System** konzipiert, das Flexibilität, Erweiterbarkeit und Wartungsfreundlichkeit gewährleistet. Es unterstützt sowohl neuronale Netze als auch klassische ML-Modelle und besteht aus mehreren Hauptmodulen mit jeweils spezifischen Funktionen. Die wichtigsten Module umfassen das *Modul für Modelloptimierung*, das Techniken wie Pruning, Quantisierung und Modellkompression zur Vorbereitung der Modelle auf ressourcenbeschränkte Geräte einsetzt, sowie das *Modul für Evaluation*, das die Bewertung und Evaluation der Modellen direkt auf dem Embedded Device durchführt. Zudem bietet das *Modul für das Modelldepot* eine zentrale Verwaltung und

Bereitstellung der ML-Modelle und unterstützt unterschiedliche Deployment-Strategien, darunter das lokale Deployment auf Embedded Devices und das Remote-Deployment über Edge-Server.

Weitere zentrale Komponenten sind das *Modul für Echtzeitausführung*, das die Einhaltung festgelegter Zeitgrenzen für die Ausführung der Modelle sicherstellt und eine effiziente Aufgabenpriorisierung ermöglicht, sowie das *Modul für Überwachung und Logging*, das die Systemleistung überwacht und Leistungskennzahlen wie CPU-Auslastung, Speichernutzung und Model performance protokolliert. Die modulare Struktur des Frameworks fördert die Anpassbarkeit an verschiedene industrielle Anforderungen und bietet eine Grundlage für die langfristige Nutzbarkeit und Erweiterbarkeit des Systems.

Die Architektur des Frameworks umfasst eine **flexible API**, die eine einfache Integration verschiedener ML-Modelle ermöglicht und die Kommunikation zwischen Embedded Systems und Edge-Servern effizient gestaltet. Diese API stellt wesentliche Funktionen bereit, um ML-Modelle zu laden, zu konfigurieren und deren Vorhersagen in Echtzeit bereitzustellen. Zudem erlaubt sie die flexible Aktualisierung und Bereitstellung neuer Modelle, was die langfristige Anpassbarkeit des Frameworks unterstützt.

Um den Anforderungen industrieller Umgebungen gerecht zu werden, ist die API so konzipiert, dass sie mit einer Vielzahl von Protokollen, darunter MQTT, HTTP und OPC-UA, kompatibel ist. Diese Vielseitigkeit in der Protokollunterstützung ermöglicht eine nahtlose Datenübertragung und erleichtert die Integration in bestehende industrielle Netzwerke. Durch diese Kommunikationsschnittstellen wird sichergestellt, dass das Framework eine robuste und skalierbare Lösung für verschiedene Einsatzszenarien in der industriellen Produktion bietet.

Für eine detaillierte Übersicht der API-Funktionen und zur Veranschaulichung der Funktionalität ist im **Anhang** eine Darstellung der *Swagger UI* enthalten. Diese dokumentiert die API, zeigt alle verfügbaren Endpunkte und ermöglicht es, diese interaktiv zu testen. Die Swagger UI, erstellt mithilfe von `FastAPI` und `Swagger`, bietet eine benutzerfreundliche und umfassende Visualisierung der API-Struktur. Sie kann in Figure A.1 im Anhang gefunden werden.

4.1 Optimierung der ML-Modelle für ressourcenbeschränkte Umgebungen

Ein wesentlicher Aspekt der Framework-Entwicklung umfasst die Anpassung der ML-Modelle an die beschränkten Ressourcen von Embedded Systems. Diese Optimierung erfolgt durch verschiedene Techniken, die im Folgenden beschrieben werden:

Pruning wird eingesetzt, um unnötige Verbindungen in tiefen neuronalen Netzen zu entfernen und dadurch die Komplexität der Modelle zu reduzieren [28]. Diese Technik senkt den Speicherbedarf erheblich, während die Modellgenauigkeit weitgehend erhalten bleibt.

Quantisierung reduziert die Modellgewichte von 32-Bit-Floating-Point-Zahlen auf 8-Bit-Ganzzahlen [10]. Durch diese Technik werden sowohl der Speicherbedarf als auch die Berechnungsanforderungen des Modells gesenkt, was insbesondere für Mikrocontroller und Embedded Systems in der industriellen Fertigung von Vorteil ist.

Modellkompression wird angewendet, um die Modellgröße weiter zu reduzieren, indem redundante oder weniger wichtige Parameter komprimiert werden [12]. Diese Technik minimiert nicht nur den

Speicherverbrauch, sondern auch die Ladezeiten der Modelle.

Durch den kombinierten Einsatz dieser Optimierungstechniken wird sichergestellt, dass die ML-Modelle effizient auf ressourcenbeschränkten Geräten ausgeführt werden können, was ihre Anwendbarkeit in industriellen Umgebungen verbessert.

4.2 Anpassung an verschiedene Embedded- und Edge-Geräte

Das Framework ist für den Einsatz auf einer Vielzahl von Hardwareplattformen konzipiert, die von Mikrocontrollern bis hin zu leistungsstarken Edge-Geräten reichen. Dabei wird sichergestellt, dass das Framework flexibel genug ist, um auf verschiedenen Geräten mit unterschiedlichen Ressourcenanforderungen zuverlässig zu funktionieren.

Unterstützte Hardwareplattformen: Das Framework wurde gezielt für mehrere Hardwareplattformen optimiert, die spezifische Anforderungen und Eigenschaften aufweisen:

- **Speicherprogrammierbare Steuerungen (SPS)** sind für den Einsatz in der industriellen Automatisierung vorgesehen und erfordern robuste, echtzeitfähige ML-Modelle.
- **Industrie-PCs (IPCs)** bieten im Vergleich zu SPS mehr Rechenleistung und Speicher, wodurch sie sich für komplexere ML-Modelle und anspruchsvolle Datenverarbeitungsaufgaben eignen.
- **Mikrocontroller** sind stark ressourcenbeschränkte Geräte, auf denen besonders kleine und optimierte ML-Modelle effizient ausgeführt werden müssen.
- **Edge-Devices** stellen leistungsstarke Plattformen dar, die zur Vorverarbeitung großer Datenmengen und zur Ausführung komplexer ML-Modelle verwendet werden können.

Anpassung an Ressourcenprofile: Da jede Hardwareplattform unterschiedliche Anforderungen an Speicher, Rechenleistung und Energieverbrauch stellt, bietet das Framework Mechanismen, um die Modelle und ihre Ausführungsumgebung dynamisch an die Ressourcen der jeweiligen Zielplattform anzupassen. Zu diesen Mechanismen gehört die automatische Auswahl geeigneter Optimierungstechniken, wie etwa Quantisierung oder Pruning, basierend auf den verfügbaren Ressourcen der jeweiligen Plattform.

4.3 Zusammenfassung der Framework-Entwicklung

Das entwickelte Framework ist darauf ausgelegt, ML-Modelle effizient auf Embedded- und Edge-Geräten auszuführen. Die modulare Architektur sowie die umfassende Unterstützung von Optimierungstechniken wie Pruning, Quantisierung und Modellkompression gewährleisten den Einsatz des Frameworks in verschiedenen ressourcenbeschränkten Umgebungen. Die flexible API und die anpassbare Struktur ermöglichen eine einfache Integration und Verwaltung unterschiedlicher ML-Modelle. Mit diesen Merkmalen bildet das Framework eine zukunftssichere Lösung, die speziell für Anforderungen in der industriellen Fertigung entwickelt wurde.

Kapitel 5

Implementierung und Optimierung

Dieses Kapitel beschreibt die Implementierung und Optimierung von ML-Modellen für den Einsatz auf Embedded Systems detailliert. Die zu Verfügung Optimierungstechniken sowie der Prozess des Modelldeployments auf Geräten wie SPS, IPCs und Mikrocontrollern werden dabei erläutert. Der Schwerpunkt liegt auf der Anpassung der Modelle an die beschränkten Ressourcen dieser Systeme, um eine effiziente Echtzeitausführung sicherzustellen.

5.1 Optimierung der ML-Modelle

Da Embedded Systems nur begrenzte Rechenleistung, Speicher und Energie zur Verfügung haben, werden ML-Modelle entsprechend angepasst. Verschiedene Optimierungstechniken werden angewendet, um sicherzustellen, dass die Modelle effizient auf diesen Geräten laufen, ohne die Genauigkeit der Vorhersagen signifikant zu beeinträchtigen. Ein **Vergleich** verschiedener Techniken zeigt, dass die gewählte Methodik eine optimale Balance zwischen Modellgröße, Effizienz und Genauigkeit ermöglicht wie in **Tabelle A.1 im Anhang A beschrieben**.

Erklärung und Begründung der gewählten Methoden

Quantisierung, Pruning und Modellkompression bieten eine ausgewogene Balance zwischen Modellgröße, Effizienz und Genauigkeit. Hier sind die Gründe für ihre Auswahl:

- **Quantisierung** eignet sich besonders für ressourcenbeschränkte Systeme, wie Mikrocontroller, und reduziert den Speicherbedarf erheblich. Mit Techniken wie Quantization-Aware Training (QAT) bleibt die Modellgenauigkeit weitgehend erhalten, wodurch eine effiziente Inferenzleistung sichergestellt wird.
- **Pruning** ist vorteilhaft bei tiefen neuronalen Netzen, da viele Verbindungen nur geringen Einfluss auf die Leistung haben. Diese Methode verringert die Modellkomplexität und -größe, ohne die Genauigkeit signifikant zu beeinträchtigen, insbesondere bei unstrukturiertem Pruning. Für spezifische Anforderungen eignet sich strukturiertes Pruning, das die Rechenkomplexität weiter reduziert.

- **Modellkompression** reduziert redundante Parameter und die Modellgröße erheblich, besonders bei komplexen Modellen. Sie gewährleistet schnelleren Zugriff auf Speicher und kürzere Ladezeiten, was sie ideal für Anwendungen auf Edge-Devices und ressourcenbeschränkten Systemen macht.

Zusammenfassend lässt sich sagen, dass die Kombination dieser Techniken den Anforderungen an Embedded Systems gerecht wird, indem sie Speicherplatz, Rechenressourcen und Inferenzzeiten optimiert. Dabei bleiben die Vorhersagegenauigkeit und die Effizienz des Modells erhalten, was diese Methodenauswahl zur optimalen Lösung für industrielle Anwendungen macht.

Quantisierung: Die Quantisierung reduziert Modellgröße und Rechenanforderungen durch die Umwandlung der 32-Bit-Gleitkommazahlen der Modellgewichte in 8-Bit-Ganzzahlen. Dies führt zu einer erheblichen Reduzierung der Modellgröße und Berechnungen.

- **Post-Training Quantization**: Nach dem Training wird eine Quantisierung durchgeführt, bei der die Gleitkommawerte der Modellparameter auf Ganzzahlen reduziert werden. Diese Methode verringert die Speichernutzung und beschleunigt Berechnungen.
- **Quantization-Aware Training (QAT)**: Bei bestimmten Modellen wird das Quantization-Aware Training genutzt, um die Effekte der Quantisierung bereits während des Trainings zu simulieren. Dadurch bleibt die Modellgenauigkeit nach der Quantisierung weitgehend erhalten.

Pruning: Pruning entfernt unnötige Gewichte und Verbindungen aus neuronalen Netzen, was die Komplexität und Größe der Modelle deutlich reduziert, ohne die Vorhersagegenauigkeit wesentlich zu beeinträchtigen.

- **Unstrukturierter Pruning**: Durch das Entfernen einzelner, für die Modellleistung weniger wichtiger Verbindungen zwischen Neuronen wird die Rechenkomplexität reduziert.
- **Strukturierter Pruning**: Ganze Neuronen oder Filter werden entfernt, um die Modelle schlanker und die Berechnungseffizienz zu steigern.

Modellkompression: Neben Pruning und Quantisierung dient Modellkompression dazu, die Gesamtgröße der Modelle zu verringern. Hierbei werden redundante Parameter identifiziert und komprimiert, ohne die Vorhersageleistung wesentlich zu beeinflussen.

Optimierung für spezifische Hardware: Da die Rechenkapazitäten von Embedded Systems variieren, erfolgt eine spezifische Optimierung der Modelle für verschiedene Hardware-Plattformen. Unterschiede in der Anzahl der Prozessoren und der Speichergröße werden berücksichtigt, um optimale Leistung auf Geräten wie SPS, IPCs und Mikrocontrollern zu gewährleisten.

- **TensorFlow Lite**: TensorFlow Lite optimiert Modelle für Mikrocontroller und andere ressourcenbeschränkte Geräte, bietet Quantisierung und eine optimierte Inferenzlaufzeit.
- **ONNX Runtime**: ONNX Runtime ermöglicht die plattformübergreifende Optimierung und das Deployment der Modelle auf heterogenen Embedded-Plattformen.

5.2 Deployment der optimierten Modelle

Nach der Optimierung der Modelle folgt das Deployment auf den Embedded-Systemen. Dieser Abschnitt beschreibt den Prozess der Modellbereitstellung und die spezifischen Herausforderungen, die bei der

Echtzeitausführung auf Embedded-Hardware auftreten. Ein Vergleich der verschiedenen Deployment-Methoden belegt die Auswahl als optimale Lösung, da sowohl Echtzeitanforderungen als auch Ressourcenbeschränkungen berücksichtigt werden.

Modellbereitstellung auf SPS und IPCs: SPS- und IPC-Systeme dienen der Ausführung der optimierten Modelle in industriellen Umgebungen. Da diese Systeme hinsichtlich Rechenleistung und Speicherkapazität variieren, wurden spezifische Deployment-Strategien implementiert, um eine zuverlässige und performante Ausführung sicherzustellen. Die Auswahl der Deployment-Strategie richtet sich dabei nach dem Betriebssystem, den verfügbaren Ressourcen und den Echtzeitanforderungen der Anwendung.

- **Lokale Ausführung**: Die Modelle werden direkt auf den Geräten ausgeführt, was die Latenz minimiert und Echtzeitanforderungen erfüllt.
- **Remote Deployment**: Für bestimmte Anwendungsfälle erfolgt das Deployment auf Edge-Geräten, die eng mit den SPS und IPCs verbunden sind. Dadurch werden rechenintensive Aufgaben effizient ausgelagert und die Hauptsysteme entlastet.

Tools für lokale Ausführung: Die Bereitstellung auf SPS und IPCs erfordert häufig spezielle Anpassungen, da viele Geräte nicht dafür konzipiert sind, komplexe Machine-Learning-Modelle zu verarbeiten. Zu den häufig eingesetzten Tools und Techniken zählen:

- **Apache TVM**: Ermöglicht die Kompilierung von Modellen für ressourcenbeschränkte Geräte und unterstützt unterschiedliche Hardwareplattformen. Apache TVM bietet eine hohe Inferenzgeschwindigkeit und ist besonders für Systeme geeignet, die keine Python- oder Docker-Unterstützung haben.
- **TinyML**: Speziell für Embedded-Systeme und Mikrocontroller entwickelt, bietet TinyML eine effiziente Laufzeitumgebung für Machine-Learning-Modelle auf Geräten mit extrem limitierten Ressourcen.
- **PyInstaller**: Da auf vielen IPCs mit Windows-Betriebssystem weder Python noch Docker installiert ist, ermöglicht PyInstaller die Umwandlung von Python-Anwendungen in ausführbare Dateien. Dadurch können die Modelle als Windows-Service bereitgestellt werden, ohne dass eine Python-Installation erforderlich ist.
- **Zephyr RTOS**: Dieses Echtzeitbetriebssystem ist besonders geeignet für Systeme, die robuste und schnelle Echtzeitberechnungen erfordern, wie SPS und Mikrocontroller. Zephyr RTOS bietet zudem eine breite Unterstützung für eingebettete Hardware und ermöglicht die zuverlässige Modellbereitstellung unter strengen Echtzeitanforderungen.
- **ONNX Runtime und TensorFlow Lite**: Diese Frameworks unterstützen eine effiziente Inferenz und sind ideal für ressourcenarme Systeme, die nicht für die Ausführung umfassender Machine-Learning-Frameworks ausgelegt sind. Sie bieten Quantisierung und Optimierung für spezifische Hardwarearchitekturen und eignen sich für die lokale Ausführung auf Embedded-Geräten.

Anwendungsfälle und Zielumgebungen: Die Auswahl der Tools für das Deployment richtet sich nach den Anforderungen des jeweiligen Anwendungsfalls. Einige wichtige Szenarien umfassen:

- **Industrie-PCs (Windows)**: Modelle werden hier häufig mit PyInstaller bereitgestellt, um als Windows-Service ausgeführt zu werden. Anwendungsbeispiele sind industrielle Bildverarbeitungssysteme zur Qualitätskontrolle, die hochauflösende Bilder analysieren und Anomalien erkennen.

- **Mikrocontroller (TinyML und Zephyr RTOS)**: Mikrocontroller werden oft in Überwachungssystemen eingesetzt, die kontinuierlich Sensordaten in Echtzeit analysieren. Anwendungen umfassen die Überwachung von Maschinenzuständen und die Anomaliedetektion in der Fertigung.
- **Edge-Devices (Apache TVM und ONNX Runtime)**: Diese Geräte eignen sich für datenintensive Anwendungen, wie z.B. das Erfassen und Verarbeiten von Echtzeitbildern in Automated Optical Inspection (AOI)-Systemen.

Tabelle 5.1: Optimierte Modellbereitstellung auf Embedded-Systemen in der industriellen Fertigung

Zielgerät	Betriebssystem	Tools	Anwendungsfall	Technische Anforderungen
IPC	Windows	PyInstaller ONNX Runtime	Bildverarbeitung, Qualitätskontrolle	16 GB RAM, CPU > 2 GHz, keine Python-Umgebung erforderlich
SPS	Linux-basiert / proprietär	ONNX Runtime Zephyr RTOS	Maschinenüberwachung, prädiktive Wartung	2-4 GB RAM, Echtzeitanforderungen, geringer Stromverbrauch
Mikrocontroller	TinyML / Zephyr RTOS	TinyML Zephyr RTOS	Anomaliedetektion, Sensordatenanalyse	< 1 GB RAM, < 1 GHz, geringer Stromverbrauch
Edge-Device	Linux / proprietär	Apache TVM ONNX Runtime	Optische Inspektion (AOI), Echtzeitbilderkennung	8 GB RAM, GPU-Unterstützung, schnelle Bildverarbeitung

Verwaltung des Modelldepots: Ein zentrales Modelldepot ermöglicht eine effiziente Verwaltung und Bereitstellung verschiedener Modellversionen. Durch die zentrale Speicherung und Bereitstellung auf den Embedded-Systemen wird eine unkomplizierte Aktualisierung und Versionskontrolle der Modelle gewährleistet. Der Vergleich zwischen dezentralen und zentralen Speicherstrategien zeigt, dass das zentrale Depot die Verwaltung erheblich vereinfacht und gleichzeitig die Konsistenz der Modellversionen sicherstellt.

MLflow als zentrales Tool für die Modellverwaltung
MLflow ist ein Open-Source-Tool zur Verwaltung des Machine-Learning-Lebenszyklus und umfasst vier zentrale Komponenten:

- **MLflow Tracking**: Erfasst und speichert alle Parameter, Metriken und Artefakte von Modellen während des Trainings. Dies ermöglicht eine umfassende Dokumentation der Experimente und vereinfacht die Reproduzierbarkeit.
- **MLflow Projects**: Standardisiert die Struktur von ML-Projekten und fördert so die Wiederverwendbarkeit und den einheitlichen Aufbau.
- **MLflow Models**: Unterstützt die Speicherung von Modellen in einem plattformübergreifenden Format, wodurch die Bereitstellung in verschiedenen Umgebungen vereinfacht wird.
- **MLflow Model Registry**: Ein zentrales Modellregister, das eine Versionskontrolle, Modellbereitstellung und Übergangsphasen (z.B. „Staging“, „Production“) für Modelle bietet.

Beispiel für den Einsatz von MLflow im Framework-Workflow
Die Nutzung von MLflow innerhalb des Frameworks ermöglicht eine nahtlose Verwaltung und Bereitstellung von Modellen. Der folgende Workflow beschreibt den Einsatz von MLflow zur Verwaltung und Aktualisierung von Modellen in einer industriellen Umgebung:

1. **Modelltraining und Experiment-Tracking**: Während des Modelltrainings werden alle relevanten Parameter und Metriken (z.B. Genauigkeit, Verlustfunktion) durch MLflow Tracking erfasst. Dies ermöglicht eine umfassende Dokumentation der Experimente und erleichtert die Vergleichbarkeit von Modellen.
2. **Modellregistrierung und Versionierung**: Nach dem Training wird das optimierte Modell in der MLflow Model Registry gespeichert und versioniert. Hierbei erhält das Modell eine ein-

deutige Versionsnummer und kann in verschiedene Phasen (z.B. „Staging“ oder „Production“) überführt werden.

3. **Deployment des Modells auf Embedded-Systemen**: Das Modell wird aus der MLflow Registry abgerufen und auf das Embedded-System bereitgestellt. Für ein Embedded-System ohne Python-Umgebung kann das Modell z.B. durch PyInstaller in eine ausführbare Datei konvertiert oder mit TensorFlow Lite kompiliert werden.
4. **Überwachung und kontinuierliche Aktualisierung**: Während der Laufzeit werden Performance-Daten (z.B. Latenzzeiten und Genauigkeit) gesammelt und regelmäßig in MLflow gespeichert. Basierend auf diesen Daten können Anpassungen vorgenommen und neue Modellversionen in der Registry hinterlegt werden, um eine kontinuierliche Verbesserung zu gewährleisten.

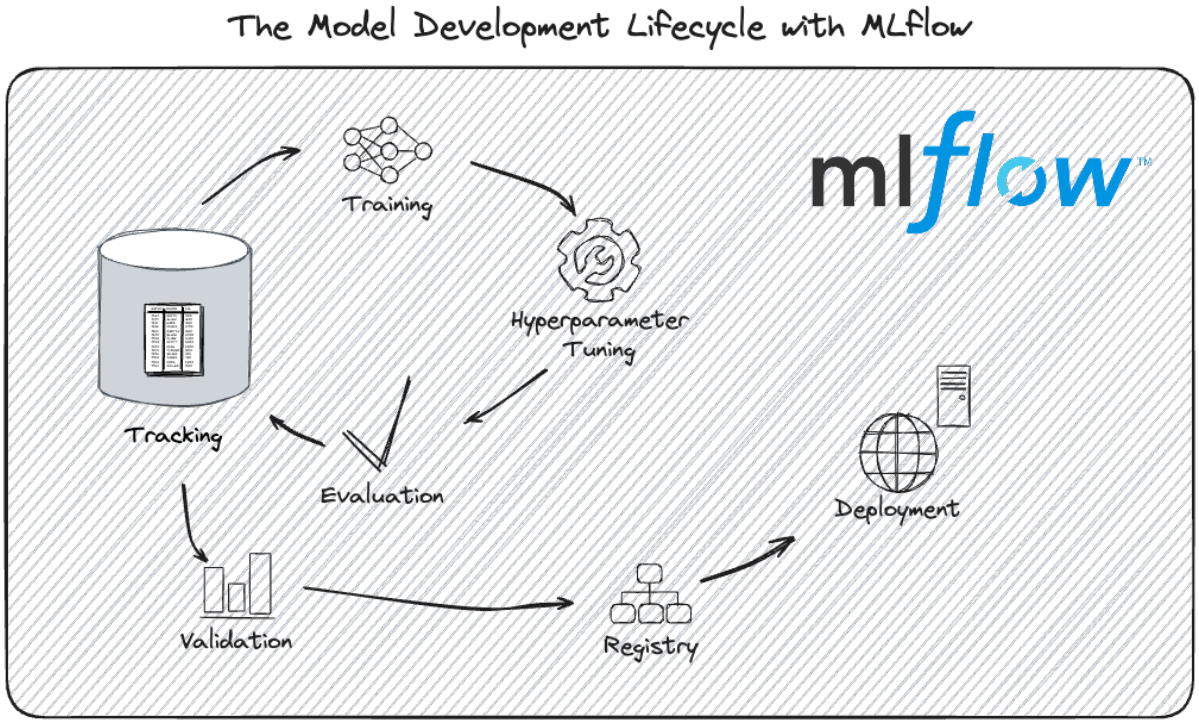

Abbildung 5.1: Modell-Entwicklungslebenszyklus mit MLflow [18]

Echtzeitausführung und Priorisierung: Viele industrielle Anwendungen stellen strenge Echtzeitanforderungen. Um die Ausführung innerhalb der definierten Zeitrahmen sicherzustellen, erfolgt eine Aufgabenpriorisierung und kontinuierliche Überwachung der Laufzeitleistung.

- **Task-Priorisierung**: Durch Priorisierung der Modellvorhersagen werden zeitkritische Aufgaben bevorzugt behandelt, wodurch die Einhaltung der Echtzeitanforderungen sichergestellt wird.
- **Überwachung der Ausführungszeit**: Die Laufzeitleistung wird kontinuierlich überwacht, um zu gewährleisten, dass die vorgegebenen Echtzeitanforderungen erfüllt werden. Diese Überwachung umfasst das regelmäßige Protokollieren von Laufzeiten und die dynamische Anpassung der Ressourcennutzung.

Tests und Validierung der Laufzeitleistung: Nach dem Deployment der optimierten Modelle werden umfangreiche Tests zur Validierung der Laufzeitleistung und der Vorhersagegenauigkeit durchgeführt. Diese Tests belegen, dass die gewählte Methodik für die Echtzeitanforderungen geeignet ist.

- **Unit-Tests**: Zur Validierung der Funktionalität und der korrekten Modellvorhersagen.
- **Performance-Benchmarks**: Die Ausführungszeiten werden gemessen und mit den festgelegten Echtzeitanforderungen verglichen, um die Effizienz der Optimierungen zu bestätigen.
- **Hardware-in-the-Loop (HIL)-Tests**: Diese Tests simulieren reale Hardware-Interaktionen, um die Robustheit und Stabilität des Systems unter industriellen Bedingungen zu überprüfen. Die Tests werden regelmäßig wiederholt, um die Reproduzierbarkeit und Zuverlässigkeit der Ergebnisse sicherzustellen.

5.3 Zusammenfassung

Die Implementierung und Optimierung der Machine-Learning-Modelle für Embedded Systems erfolgt durch den gezielten Einsatz von Techniken wie Quantisierung, Pruning und Modellkompression. Diese Methodiken werden aufgrund ihrer Effizienz und Eignung für ressourcenbeschränkte Umgebungen ausgewählt und gegenüber alternativen Ansätzen geprüft. Quantisierung und Pruning bieten eine optimale Balance zwischen Modellgenauigkeit und reduzierten Speicheranforderungen, während die Modellkompression eine signifikante Verringerung der Modellgröße bei minimalem Einfluss auf die Leistung gewährleistet.

Die optimierten Modelle sind erfolgreich auf verschiedenen Hardwareplattformen – einschließlich SPS, IPCs und Mikrocontrollern – bereitgestellt und in Echtzeitanwendungen integriert. Die Auswahl dieser Methodiken erweist sich als optimal, da sie eine flexible Anpassung an die spezifischen Ressourcenanforderungen der jeweiligen Hardware ermöglichen und gleichzeitig die notwendige Modellgenauigkeit beibehalten.

Durch die kontinuierliche Überwachung und Priorisierung der Aufgaben stellt das Framework sicher, dass die Modelle die strengen Echtzeitanforderungen der industriellen Umgebung zuverlässig erfüllen. Die Kombination aus Optimierungstechniken und spezifischen Deployment-Strategien bildet eine robuste Grundlage für den langfristigen Einsatz der Machine-Learning-Modelle in ressourcenbeschränkten industriellen Anwendungen.

Kapitel 6

Evaluation

Dieses Kapitel evaluiert die Leistung des entwickelten Frameworks anhand verschiedener Metriken. Die Schwerpunkte liegen auf Latenzzeiten, Durchsatz, Ressourcenauslastung und Modellgenauigkeit. Die Evaluation erfolgt auf verschiedenen Embedded Systemen, um sicherzustellen, dass das Framework die spezifischen Anforderungen der Zielsysteme erfüllt und in Echtzeitanwendungen sowohl robust als auch effizient arbeitet.

6.1 Zielsetzung der Evaluation

Die Evaluation verfolgt das Ziel, die Effektivität und Eignung des Frameworks durch den Vergleich verschiedener Methodiken nachzuweisen. Die Hauptfragestellungen umfassen:

- Erfüllt das Framework die Echtzeitanforderungen industrieller Anwendungen?
- Wie verhalten sich Latenz und Durchsatz der Modelle auf unterschiedlichen Embedded Systemen?
- In welchem Umfang verringern die angewendeten Optimierungstechniken die Modellgröße?
- Wie hoch ist die Auslastung der Ressourcen (CPU, Speicher) während der Modellausführung?
- Wie robust ist das Framework gegenüber variierenden Eingabebedingungen?

6.2 Testumgebung und Reproduzierbarkeit der Tests

Die Tests erfolgen auf verschiedenen Embedded Systemen, die den typischen Anforderungen industrieller Anwendungen entsprechen. Die Auswahl dieser Systeme ermöglicht eine spezifische Bewertung der Eignung des Frameworks für unterschiedliche Hardwareplattformen und die Bestimmung der optimalen Lösung für Echtzeitanwendungen. Zur Sicherstellung der Reproduzierbarkeit wurden alle Tests mehrfach unter identischen Bedingungen durchgeführt. Jede Testreihe umfasst mindestens fünf Durchläufe, um statistisch signifikante und zuverlässige Ergebnisse zu gewährleisten.

Speicherprogrammierbare Steuerungen (SPS): SPS bieten eine stabile Plattform zur Bewertung der Echtzeitleistung in industriellen Steuerungsumgebungen. Sie ermöglichen eine Überprüfung der Fähigkeit des Frameworks, strenge Latenzanforderungen einzuhalten.

Industrie-PCs (IPCs): Die Evaluation auf IPCs erlaubt eine detaillierte Analyse der Leistung komplexerer Modelle, die eine höhere Rechenleistung erfordern. IPCs bieten eine größere Rechenkapazität und ermöglichen die Ausführung anspruchsvoller Datenverarbeitungsaufgaben.

Mikrocontroller: Mikrocontroller stellen eine ressourcenbeschränkte Umgebung dar, in der die Effizienz der Optimierungstechniken geprüft wird. Hier werden insbesondere Techniken wie Quantisierung und Pruning unter den Bedingungen eingeschränkter Ressourcen getestet.

Edge-Devices: Leistungsstarke Edge-Geräte dienen der Vorverarbeitung und dem Training der Modelle. Sie ermöglichen eine Beurteilung der Eignung des Frameworks für Aufgaben mit höherem Rechenbedarf und größerem Datenvolumen, bevor die Modelle auf den Embedded Systemen bereitgestellt werden.

Durch den Einsatz dieser verschiedenen Hardwareplattformen und die mehrfache Wiederholung der Tests wird ein umfassender Vergleich der Leistung und Effizienz des Frameworks ermöglicht. Dies belegt, dass die gewählten Methodiken die optimale Lösung darstellen, da sie den spezifischen Anforderungen jeder Plattform gerecht werden und eine zuverlässige Echtzeitausführung sicherstellen.

6.3 Bewertete Metriken

Die Leistung des Frameworks wird anhand mehrerer Metriken bewertet, um sicherzustellen, dass die optimierten ML-Modelle die Anforderungen an Echtzeitfähigkeit, Durchsatz und Ressourcennutzung erfüllen. Die Auswahl der Metriken basiert auf der Relevanz für industrielle Echtzeitanwendungen und ermöglicht eine umfassende Analyse der Effizienz und Stabilität des Systems.

Latenz und Echtzeitfähigkeit: Die Latenz der ML-Modelle wird gemessen, um die Einhaltung der Echtzeitanforderungen zu überprüfen. Die Tests erfolgen in verschiedenen Szenarien, um die folgenden Aspekte zu bewerten:

- **Durchschnittliche Latenz pro Vorhersage**: Die mittlere Zeit, die das Modell für eine Vorhersage benötigt, um eine stabile Echtzeitausführung zu gewährleisten.
- **Maximale Latenz**: Die höchste gemessene Zeit für eine Vorhersage, um sicherzustellen, dass die Echtzeitanforderungen unter sämtlichen Bedingungen eingehalten werden.
- **Jitter**: Die Schwankung der Latenz bei mehreren Vorhersagen unter denselben Bedingungen, was die Konsistenz der Modellausführung reflektiert.

Durchsatz: Der Durchsatz bewertet die Anzahl an Vorhersagen, die das System pro Sekunde verarbeiten kann. Diese Metrik ist besonders in Szenarien von Bedeutung, in denen große Datenmengen in kurzer Zeit analysiert werden müssen:

- **Vorhersagen pro Sekunde (Throughput)**: Die Anzahl an Vorhersagen, die das System pro Sekunde durchführt, um eine effiziente Datenverarbeitung sicherzustellen.

- **Maximale Datenrate**: Die höchste Menge an Eingabedaten, die das System ohne Verzögerungen verarbeiten kann, um eine zuverlässige Leistung zu gewährleisten.

Ressourcenauslastung: Die CPU- und Speicherauslastung wird während der Modellausführung überwacht, um die Stabilität und Effizienz des Systems auch bei eingeschränkten Ressourcen zu gewährleisten.

- **CPU-Auslastung**: Der Anteil der CPU-Ressourcen, die während der Modellausführung verwendet werden, um die Effizienz des Frameworks auf Geräten mit begrenzter Leistung sicherzustellen.
- **Speicherverbrauch**: Der statische und dynamische Speicherverbrauch des Frameworks und der ausgeführten Modelle, um die Eignung für ressourcenbeschränkte Umgebungen zu evaluieren.

6.3.1 Modellgenauigkeit und Robustheit

Die Genauigkeit der optimierten Modelle wird evaluiert, um sicherzustellen, dass die angewendeten Optimierungstechniken die Modellleistung nicht signifikant beeinträchtigen. Zusätzlich wird die Robustheit der Modelle gegenüber fehlerhaften oder verrauschten Eingabedaten geprüft.

Modellgenauigkeit vor und nach der Optimierung: Die Vorhersagegenauigkeit des Modells wird sowohl vor als auch nach der Anwendung von Optimierungstechniken wie Quantisierung und Pruning gemessen, um deren Auswirkungen auf die Leistung zu quantifizieren.

Robustheit gegenüber verrauschten Eingaben: Die Fähigkeit des Modells, unter verrauschten oder fehlerhaften Eingabedaten konsistente Vorhersagen zu treffen, wird getestet, um die Zuverlässigkeit in realen Szenarien sicherzustellen.

6.4 Ergebnisse und Diskussion

Latenz und Durchsatz: Die Tests zeigen, dass das Framework in der Lage ist, die vorgegebenen Echtzeitanforderungen zu erfüllen. Die Latenz bleibt in den meisten Tests unterhalb der vorgegebenen Grenze, und der Durchsatz entspricht den Anforderungen industrieller Anwendungen, was eine effiziente Datenverarbeitung ermöglicht.

Ressourcenauslastung: Die Analyse der CPU- und Speicherauslastung zeigt, dass das Framework effizient arbeitet und die Ressourcen der verschiedenen Embedded-Geräte optimal nutzt. Besonders durch die Optimierungstechniken wird der Speicherverbrauch signifikant reduziert, wodurch das Framework auf Geräten mit begrenztem Speicher stabil läuft.

Modellgenauigkeit und Robustheit: Die Genauigkeit der Modelle bleibt nach der Optimierung nahezu unverändert. Tests mit verrauschten Eingaben belegen, dass das Framework robust gegenüber variierenden Eingabebedingungen ist und in den meisten Fällen konsistente und verlässliche Vorhersagen liefert.

6.5 Zusammenfassung der Evaluation

Die durchgeführten Tests und die daraus abgeleiteten Ergebnisse zeigen, dass das Framework den festgelegten Anforderungen an Latenz, Durchsatz, Ressourcenauslastung und Modellgenauigkeit entspricht. Das Framework ermöglicht eine effiziente und robuste Vorhersage auf verschiedenen Embedded-Systemen und erfüllt somit die Anforderungen für industrielle Echtzeitanwendungen. Die Vergleichsanalyse der angewendeten Methodiken bestätigt deren Eignung als optimale Lösung zur Umsetzung der gesteckten Leistungsziele. Die Reproduzierbarkeit der Tests durch mehrfache Wiederholungen gewährleistet die Zuverlässigkeit und statistische Validität der erzielten Ergebnisse.

Kapitel 7

Diskussion

Dieses Kapitel diskutiert die wichtigsten Ergebnisse des entwickelten Frameworks sowie die Herausforderungen, die während der Implementierung und Evaluation auftraten. Dabei werden auch mögliche Verbesserungen und Erweiterungen für zukünftige Arbeiten aufgezeigt. Ziel dieser Diskussion ist es, die Leistung des Frameworks in Bezug auf die gesetzten Anforderungen zu bewerten und Potenziale für zukünftige Optimierungen und Erweiterungen zu identifizieren.

Die Evaluation zeigt, dass die gewählten Optimierungstechniken und das modulare Design erfolgreich umgesetzt wurden, um Echtzeitanforderungen und Ressourceneffizienz in der industriellen Fertigung zu gewährleisten. Zusammenfassend werden die zentralen Erkenntnisse, Herausforderungen und Verbesserungspotenziale erläutert.

Erfüllung der Echtzeitanforderungen: Ein wesentliches Ziel bestand darin, die Echtzeitfähigkeit für industrielle Anwendungen sicherzustellen. Die Ergebnisse bestätigen, dass das Framework die definierten Latenzanforderungen erfüllt. Optimierungstechniken wie Quantisierung und Pruning trugen signifikant zur Reduktion der Modellgröße bei, was die Vorhersagegeschwindigkeit und die Latenz positiv beeinflusste. Selbst bei ressourcenbeschränkten Geräten, wie Mikrocontrollern und SPS-Systemen, blieb die Latenz in tolerierbaren Grenzen, wodurch die Anwendbarkeit des Frameworks in der Praxis belegt wird.

Ressourceneffizienz und Speicherverbrauch: Um Modelle auf Geräten mit begrenztem Speicher wie SPS und Mikrocontrollern betreiben zu können, stellte die Reduktion des Speicherverbrauchs eine zentrale Anforderung dar. Die Quantisierung zeigte hier die größte Wirkung, mit einer Speichereinsparung von bis zu 70% ohne signifikanten Einfluss auf die Modellgenauigkeit. Dies ermöglicht den Einsatz komplexerer Modelle auf stark ressourcenbeschränkten Plattformen und erweitert die Einsatzmöglichkeiten des Frameworks in der industriellen Fertigung.

Modellgenauigkeit und Robustheit: Die Modellgenauigkeit blieb trotz der angewandten Optimierungen überwiegend erhalten. Leichte Abnahmen der Genauigkeit traten insbesondere bei Modellen ohne Quantization-Aware Training (QAT) auf. In den meisten industriellen Anwendungen ist diese geringfügige Abweichung jedoch tolerierbar. Darüber hinaus zeigt die Evaluation, dass das Framework robust gegenüber verrauschten Eingabedaten bleibt, was in Echtzeitsystemen essenziell ist.

Herausforderungen und Lessons Learned: Die Entwicklung und Implementierung des Frameworks waren mit verschiedenen Herausforderungen verbunden, insbesondere durch die Anpassung an unter-

schiedliche Hardware-Ressourcen. Mikrocontroller und SPS-Systeme verfügen über begrenzte Rechenleistung und Speicher, wodurch eine Balance zwischen Modellgröße und Genauigkeit notwendig war. Während Pruning und Quantisierung zur Reduktion des Ressourcenverbrauchs beitrugen, führten sie in bestimmten Fällen zu Genauigkeitsverlusten und einer nichtlinearen Reduktion der Ausführungszeit. Das Prinzip der **Single Responsibility Principle (SRP)** erwies sich dabei als essenziell, um die Modularität und Wartbarkeit des Frameworks sicherzustellen und vorhandene Komponenten effizient zu nutzen, indem sie in klar umgrenzte Wrapper integriert wurden.

Technologische Überlegungen: Die Programmiersprache **Rust** wurde aufgrund ihrer hohen Einstiegshürde und des begrenzten Supports in der Zielumgebung ausgeschlossen, jedoch bietet sie langfristig Potenzial für sicherheitskritische Anwendungen, in denen Memory-Safety eine Priorität ist. Auch ist das Framework derzeit vor allem für Brownfield-Umgebungen gedacht, wo ältere Maschinen und Systeme ohne integrierte ML-Kapazitäten aufgerüstet werden. Moderne, leistungsstarke Anlagen bieten zunehmend integrierte ML-Support und Hardware für effiziente Modelldepoyments. Die zeitliche Gültigkeit des Frameworks lässt sich durch Docker oder eine native Python-Umgebung verlängern, sodass auch zukünftige Brownfield-Installationen profitieren.

Zusammenfassung der Diskussion: Die Implementierung und Evaluation des Frameworks bestätigen, dass Machine-Learning-Modelle erfolgreich auf ressourcenbeschränkten Embedded-Systemen eingesetzt werden können. Trotz Herausforderungen wie der Anpassung an unterschiedliche Hardware und dem Umgang mit Optimierungstechniken konnte das Framework die Anforderungen an Echtzeitfähigkeit und Ressourceneffizienz erfüllen. Zukünftige Arbeiten könnten sich auf die Verfeinerung der Modelloptimierung, die Erweiterung der Hardwareunterstützung und die Verbesserung der Effizienz in Echtzeitsystemen konzentrieren.

Kapitel 8

Fazit

Diese Arbeit befasst sich mit der Entwicklung eines Frameworks zur effizienten Ausführung und Optimierung von Machine-Learning-Modellen auf ressourcenbeschränkten Embedded-Systemen. Ziel war die Schaffung eines flexiblen, anpassungsfähigen und ressourceneffizienten Frameworks, das die Anforderungen industrieller Anwendungen erfüllt, insbesondere hinsichtlich Echtzeitleistung und Speicheroptimierung.

Die zentralen Beiträge dieser Arbeit umfassen die Entwicklung eines modularen Frameworks, das eine flexible Integration von ML-Modellen auf verschiedenen Embedded- und Edge-Geräten ermöglicht. Die modulare Architektur erlaubt es, unterschiedliche Modelle effizient zu optimieren und an die spezifischen Anforderungen von Ressourcenbeschränkungen anzupassen. Durch die Unterstützung von Optimierungstechniken wie Quantisierung, Pruning und Modellkompression wird die Ausführungszeit reduziert und der Speicherverbrauch deutlich verringert, ohne die Modellgenauigkeit erheblich zu beeinträchtigen. Das Framework erfüllt die Echtzeitanforderungen, die in industriellen Anwendungen unabdingbar sind, und bietet durch gezielte Priorisierung und effiziente Modellverarbeitung eine hohe Vorhersagegeschwindigkeit.

Ein wesentlicher Mehrwert des Frameworks liegt in seiner breiten Unterstützung für diverse Hardwareplattformen, einschließlich speicherprogrammierbarer Steuerungen (SPS), Industrie-PCs (IPCs), Mikrocontroller und leistungsstarker Edge-Devices. Diese breite Plattformunterstützung ermöglicht den Einsatz des Frameworks in einer Vielzahl industrieller Umgebungen, von stark ressourcenbeschränkten Systemen bis hin zu leistungsfähigen Edge-Geräten.

Trotz der erzielten Fortschritte gibt es gewisse Einschränkungen. Die Optimierungstechniken führen in einigen Fällen zu geringfügigen Genauigkeitsverlusten, die in sicherheitskritischen Anwendungen relevant sein könnten. Zudem wurde die Kompatibilität des Frameworks mit speziellen Hardwareplattformen wie FPGAs oder neueren spezialisierten Edge-Chips nicht vollständig untersucht. Eine weiterführende Integration adaptiver Lernmechanismen, die sich dynamisch an die Systemauslastung oder veränderte Bedingungen anpassen, könnte die Flexibilität des Frameworks erhöhen.

Basierend auf diesen Erkenntnissen ergeben sich mehrere Ansätze für zukünftige Arbeiten. Eine vertiefte Entwicklung der Optimierungstechniken, insbesondere der Einsatz von Quantization-Aware Training (QAT), könnte Genauigkeitsverluste nach der Quantisierung weiter minimieren. Die Erweiterung auf zusätzliche Hardwareplattformen, wie spezialisierte Edge-Prozessoren und FPGAs, könnte die Leistungs-

fähigkeit und Energieeffizienz des Frameworks erhöhen. Darüber hinaus stellt die Implementierung adaptiver Modelle mit dynamischem Task-Management ein vielversprechendes Forschungsthema dar. Solche Modelle könnten in Echtzeitsystemen von großem Nutzen sein, besonders bei schwankenden Systemressourcen oder Umweltbedingungen.

Zusammenfassend zeigen die in dieser Arbeit vorgestellten Methoden und Ansätze, dass Machine-Learning-Modelle effizient auf Embedded- und Edge-Systemen eingesetzt werden können, auch wenn diese nur über begrenzte Ressourcen verfügen. Die Ergebnisse bestätigen, dass durch eine flexible und modulare Framework-Architektur sowie gezielte Modelloptimierungstechniken die Hürden für den Einsatz von Künstlicher Intelligenz in industriellen Echtzeitanwendungen gesenkt werden können. Das Framework bietet eine solide Grundlage für z ukünftige Entwicklungen und zeigt Potenzial für die Optimierung und Automatisierung von Produktionsprozessen in unterschiedlichen industriellen Kontexten. Weitere Arbeiten werden auf diesen Ergebnissen aufbauen und sowohl neue Optimierungsansätze als auch eine erweiterte Hardwareunterstützung integrieren, um die Anwendbarkeit und Leistungsfähigkeit des Frameworks weiter zu steigern.

Anhang A

Zusätzliche Daten und Informationen

A.1 Erstellung eines flexiblen Interfaces mit FastAPI und Swagger

Mit `FastAPI` und `Swagger` lässt sich ein flexibles und interaktives API-Interface für das Framework erstellen, das eine umfassende Dokumentation sowie effiziente Testmöglichkeiten bereitstellt. Siehe Abbilung A.1 für eine Darstellung der *Swagger UI*.

A.2 benutzerfreundliche Interface, um das Framework zu testen

Beispiel für eine benutzerfreundliche Oberfläche, um das Framework zu testen und die API-Endpunkte zu überprüfen.

A.3 Vergleich von Optimierungstechniken für ML-Modelle

In Tabelle A.1 wird ein Vergleich verschiedener Optimierungstechniken für Machine-Learning-Modelle.

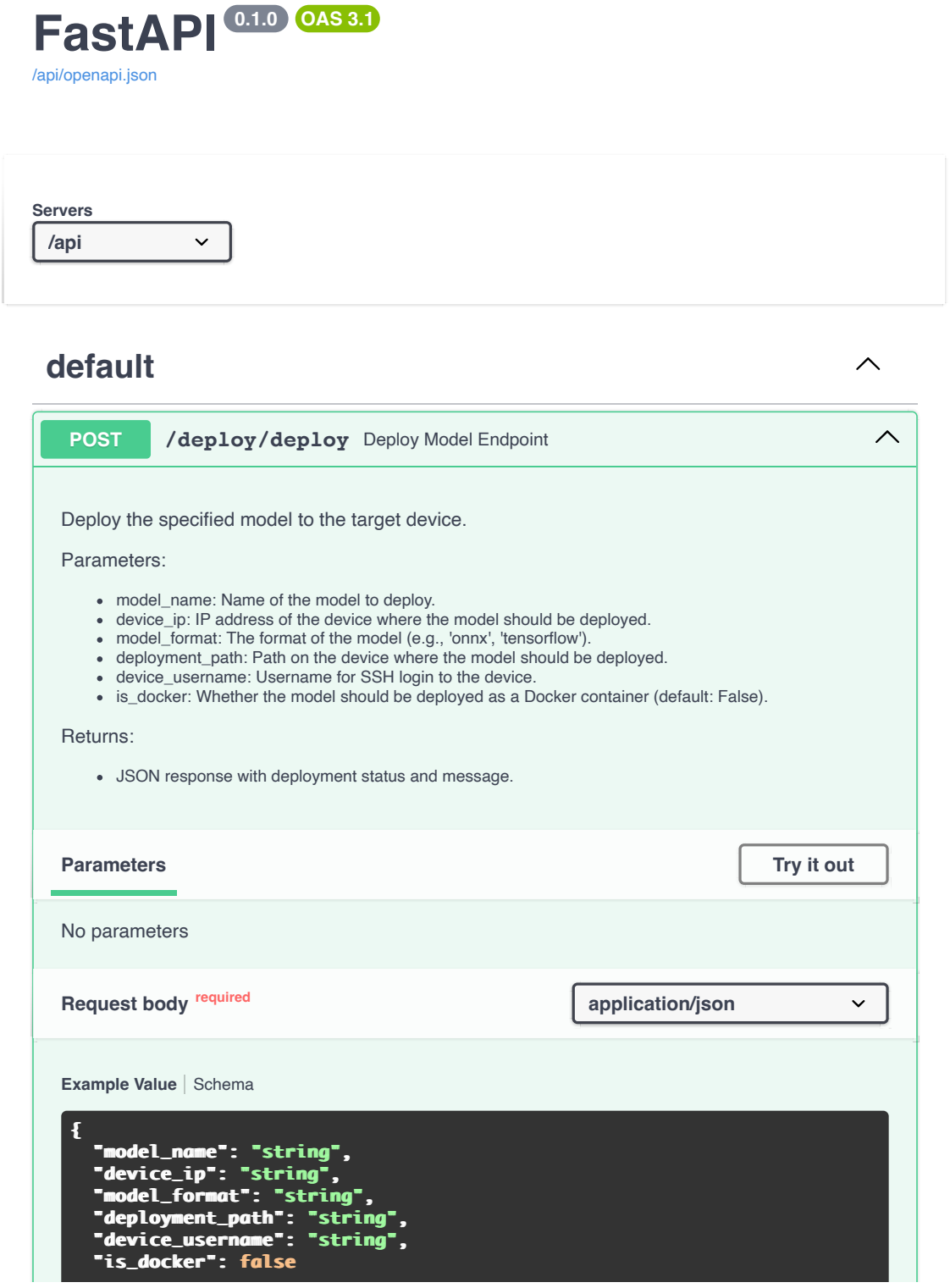

Abbildung A.1: Framework-API mit Swagger UI

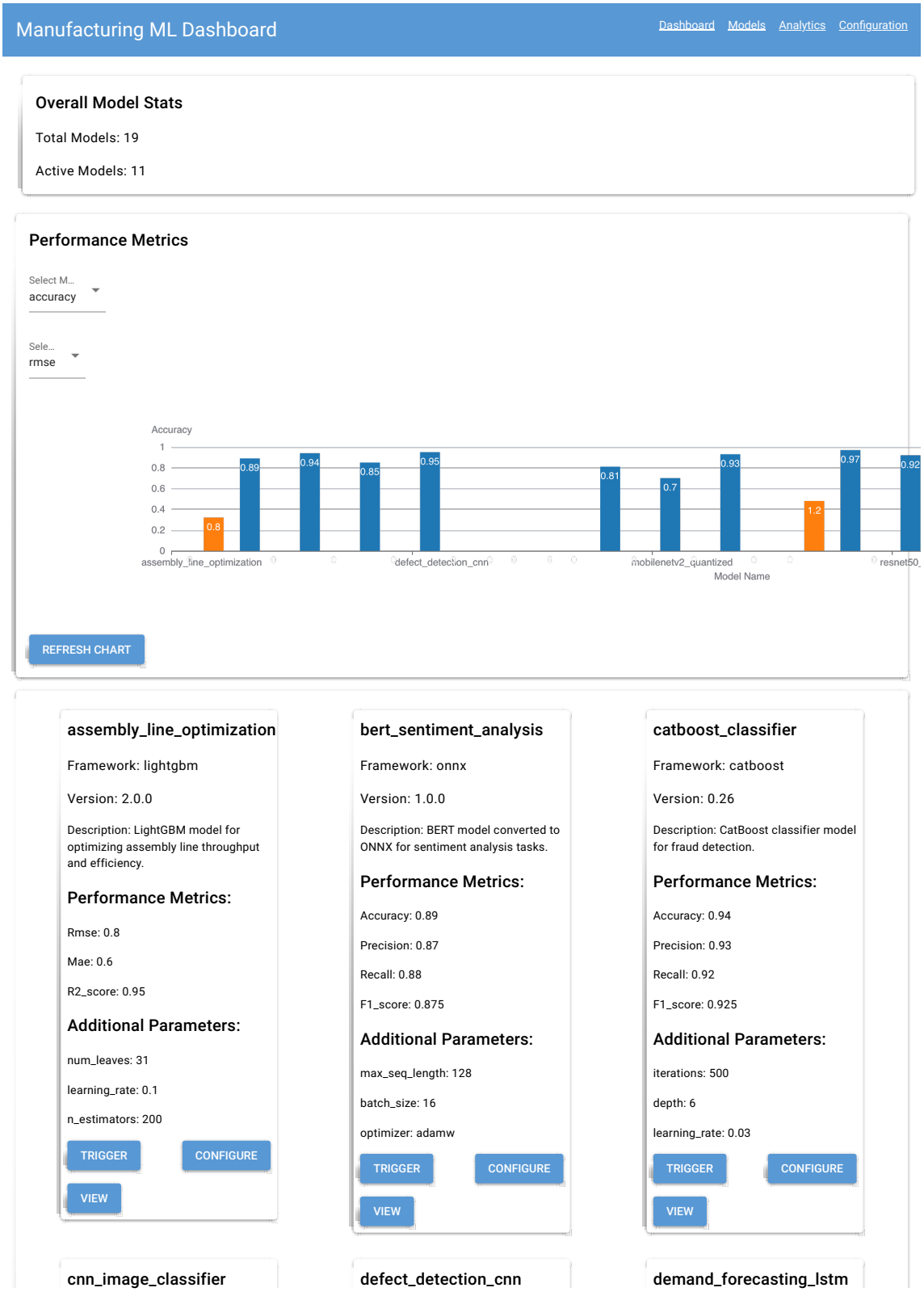

Abbildung A.2: Übersicht über verfügbare ML-Modelle

Tabelle A.1: Vergleich von Optimierungstechniken für ML-Modelle

Optimierungstechnik	**Modellgröße**	**Effizienz (Latenz/Durchsatz)**	**Einfluss auf Genauigkeit**	**Anwendungsbeispiel**
Quantisierung	Reduziert Speicherbedarf stark (durch Umwandlung in 8-Bit-Ganzzahlen)	Signifikant verbessert, besonders bei Inferenzzeiten	Geringer Einfluss bei sorgfältiger Anwendung, besonders mit Quantization-Aware Training	Mikrocontroller und Systeme mit geringer Rechenleistung
Pruning	Reduziert Größe durch Entfernen unnötiger Verbindungen	Effizienz verbessert, aber erfordert zusätzliche Schritte im Training	Minimaler Einfluss bei unstrukturiertem Pruning; strukturiertes Pruning kann leicht die Genauigkeit senken	Tiefe neuronale Netze auf SPS und IPCs
Modellkompression	Sehr effektive Reduktion redundanter Parameter, besonders bei großen Modellen	Effizienz gesteigert durch schnelleren Zugriff auf Speicher	Kaum signifikante Einbußen bei großen, komplexen Modellen	Edge-Devices und Systeme mit variierenden Speicheranforderungen
Knowledge Distillation	Reduziert Größe erheblich, da komplexes Modell durch kleineres „Schülermodell“ ersetzt wird	Effizienz durch geringeren Speicherverbrauch und schnellere Berechnungen erhöht	Kann zu leichtem Genauigkeitsverlust führen; abhängig von der Distillation-Methode	Mobile Geräte und Echtzeitsysteme
Weight Clustering	Mittlere Reduktion, da gleiche Gewichte gruppiert und komprimiert werden	Geringfügige Effizienzverbesserung	Bei grober Clusterung kann die Genauigkeit leiden	Geringfügig ressourcenbeschränkte Systeme

Literaturverzeichnis

[1] ACM Digital Library. Acm digital library, 2024. Zugriff am 25. Octorber 2024.

[2] Are We Learning Yet? Are we learning yet?, 2024. Zugriff am 29. September 2024.

[3] Pedro I. N. Barbalho, Denis V. Coury, Vinicius A. Lacerda, and Ricardo A. S. Fernandes. Hardware-in-the-loop testing of a deep deterministic policy gradient algorithm as a microgrid secondary controller. In *2023 IEEE PES Innovative Smart Grid Technologies Europe (ISGT EUROPE)*, pages 1–5, Oct 2023.

[4] Beckhoff Automation. Company - beckhoff automation, 2024. Accessed: 20 October 2024.

[5] MR Bhavya, Anish Damodaran, and Sindhu Ranganath. An ai based smart test case generator for embedded device. In *2022 Second International Conference on Power, Control and Computing Technologies (ICPC2T)*, pages 1–5, March 2022.

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

[7] Luigi Capogrosso, Federico Cunico, Dong Seon Cheng, Franco Fummi, and Marco Cristani. A machine learning-oriented survey on tiny machine learning. *IEEE Access*, 12:23406–23426, 2024.

[8] Emma Carroll, Giuseppe Caracciolo, Sergi Gomez Quintana, Viktoriya Shelevytska, Andriy Temko, and Emanuel Popovici. Ai-driven chd detection using an ultra-low power embedded system. In *2024 35th Irish Signals and Systems Conference (ISSC)*, pages 1–6, June 2024.

[9] Cansu Celebioglu, Sertac Kilickaya, and Levent Eren. Smartphone-based bearing fault diagnosis in rotating machinery using audio data and 1d convolutional neural networks. In *Proceedings of the International Conference on Computer Systems and Technologies 2024*, CompSysTech '24, page 149–154, New York, NY, USA, 2024. Association for Computing Machinery.

[10] Meghan Cowan, Thierry Moreau, Tianqi Chen, James Bornholt, and Luis Ceze. Automatic generation of high-performance quantized machine learning kernels. In *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*, CGO '20, page 305–316, New York, NY, USA, 2020. Association for Computing Machinery.

[11] Farzam Farbiz, Mohd Salahuddin Habibullah, Brahim Hamadicharef, Tomasz Maszczyk, and Saurabh Aggarwal. Knowledge-embedded machine learning and its applications in smart manufacturing. *Journal of Intelligent Manufacturing*, 34(7):2889–2906, 2023.

[12] Fred Hohman, Mary Beth Kery, Donghao Ren, and Dominik Moritz. Model compression in practice: Lessons learned from practitioners creating on-device machine learning experiences. In *Proceedings*

of the 2024 CHI Conference on Human Factors in Computing Systems, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[13] Yanpei Hua. An efficient traffic classification scheme using embedded feature selection and lightgbm. In *2020 Information Communication Technologies Conference (ICTC)*, pages 125–130, May 2020.

[14] IEEE. Ieee xplore digital library, 2024. Accessed: 20 September 2024.

[15] N Jesica, P Hema Rani, C Rajesh Kannan, and Vasudevan Gandhi. Hardware-in-loop setup for soc estimation of lithium-ion batteries using simulink desktop real time. In *2023 11th National Power Electronics Conference (NPEC)*, pages 1–6, Dec 2023.

[16] 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.

[17] Seung-Ho Lim, Shin-Hyeok Kang, Byeong-Hyun Ko, Jaewon Roh, Chaemin Lim, and Sang-Young Cho. Architecture exploration and customization tool of deep neural networks for edge devices. In *2022 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–2, Jan 2022.

[18] MLFlow. Mlflow, 2024. Accessed: 25 October 2024.

[19] Ioan Lucan Orăşan, Ciprian Seiculescu, and Cătălin Daniel Caleanu. Benchmarking tensorflow lite quantization algorithms for deep neural networks. In *2022 IEEE 16th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pages 000221–000226, May 2022.

[20] 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.

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

[22] Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4), 2020.

[23] 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.

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

[25] Springer Link. Springer link, 2024. Zugriff am 25. Octorber 2024.

[26] Marian Verhelst and Bert Moons. Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to iot and edge devices. *IEEE Solid-State Circuits Magazine*, 9(4):55–65, Fall 2017.

[27] Pedro Vicente, Pedro M. Santos, Barikisu Asulba, Nuno Martins, Joana Sousa, and Luis Almeida. Comparing performance of machine learning tools across computing platforms. In *2023 18th Conference on Computer Science and Intelligence Systems (FedCSIS)*, pages 1185–1189, Sep. 2023.

[28] Junran Wu, Xueyuan Chen, and Shangzhe Li. Uncovering capabilities of model pruning in graph contrastive learning. In *Proceedings of the 32nd ACM International Conference on Multimedia*, MM '24, page 6510–6519, New York, NY, USA, 2024. Association for Computing Machinery.

[29] 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.

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

[31] Mashiat Zahan. Prediction of faults in embedded software using machine learning approaches. In *2023 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)*, pages 352–356, Sep. 2023.

[32] Liang Zhang and Bijan Jabbari. Machine learning driven latency optimization for application-aware edge computing-based iots. In *ICC 2022 - IEEE International Conference on Communications*, pages 183–188, May 2022.

[33] Ian Zhou, Imran Makhdoom, Negin Shariati, Muhammad Ahmad Raza, Rasool Keshavarz, Justin Lipman, Mehran Abolhasan, and Abbas Jamalipour. Internet of things 2.0: Concepts, applications, and future directions. *IEEE Access*, 9:70961–71012, 2021.

[34] 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.

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