HAW HAMBURG

Masterarbeit

Michel Jacobsen

Imputation Strategies in Time Series based on Language Models

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

Michel Jacobsen

Imputation Strategies in Time Series based on Language Models

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuende Prüferin: Prof. Dr. Marina Tropmann-Frick
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 25.04.2024

Michel Jacobsen

Thema der Arbeit

Imputation Strategies in Time Series based on Language Models

Stichworte

Imputation, Fehlende Daten, Zeitreihen, Language Models, Fine-Tuning

Kurzzusammenfassung

Diese Arbeit untersucht die Eignung von Large Language Models zur Imputation von Zeitreihen. Innerhalb eines Versuchsaufbaus werden Open-Source-Modelle miteinander verglichen und mit PEFT-Methoden an die Imputation angepasst. Die Ergebnisse der Versuche zeigen, dass die Leistungsfähigkeit der Modelle von der Anzahl der Modellparameter sowie der Art des Pre-Trainings eines Modells abhängig ist. Dies hat zur Folge, dass die großen Sprachmodelle zwar auf einem Teil der Datensätze führende Ergebnisse erzielen, kleinere Modelle aber je nach Art des Pre-Trainings gleichwertig sind.

Michel Jacobsen

Title of Thesis

Imputation Strategies in Time Series based on Language Models

Keywords

Imputation, Missing Values, Time Series, Language Models, Fine-Tuning

Abstract

This paper investigates the suitability of large language models for time series imputation. Within an experimental setup, open-source models are compared and adapted to the imputation task using PEFT methods. The results of the experiments show that the performance of the models depends on the number of model parameters and the nature of a model's pre-training. As a result, while the large language models achieve leading results on part of the datasets, smaller models are equally effective depending on their type of pre-training.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

BRITS Bidirectional Recurrent Imputation for Time Series.

DAE Denoising Autoencoding.

DL Deep Learning.

DMSA Diagonally-Masked Self-Attention.

ETT Electricity Transformer Temperature.

FFN Feedforward Neural Networks.

GRU Gated Recurrent Unit.

IoT Internet of Things.

LLM Large Language Model.

LoRA Low-Rank Adaptation.

LSTM Long-Short Term Memory.

MAR Missing at random.

MCAR Missing completely at random.

MIM Masked Imputation Modeling.

MIT Masked Imputation Task.

ML Machine Learning.

MNAR Missing not at random.

NLP Natural Language Processing.

NRL Non-missing Reconstruction Loss.

ORT Observed Reconstruction Task.

PE Positional Encoding.

PEFT Parameter-Efficient Fine-Tuning.

PLM Pretrained Language Model.

RNN Recurrent Neural Network.

SVM Support Vector Machine.

1 Einleitung

1.1 Motivation

In der modernen Datenwelt sind Zeitreihendaten allgegenwärtig und weit verbreitet. Eine Zeitreihe umfasst eine Sequenz von Beobachtungen, die in regelmäßigen zeitlichen Intervallen gesammelt werden [33]. Diese Art der Daten spielt eine entscheidende Rolle für viele Entscheidungsprozesse in verschiedensten Sektoren, darunter das Finanzwesen, die Wettervorhersage und die Nachfrage nach Gütern. Doch nicht immer sind die Datensätze vollständig und können durch Fehler in der Erhebung unzureichend dokumentiert sein. Die Rekonstruktion dieser unvollständigen Daten wird als Imputation bezeichnet. Die Imputation ist in allen Datendomänen, einschließlich der Zeitreihenanalyse, von großer Bedeutung, da sie für die Zuverlässigkeit nachgelagerter Analyse- und Prognoseaufgaben essenziell ist [3].

Traditionelle Ansätze zur Behandlung fehlender Werte sind bspw. das Löschen unvollständiger Datensätze oder das Auffüllen mit Mittelwerten. Diese Methoden führen häufig zu Verzerrungen in den Daten und erschweren die nachgelagerten Aufgaben im Rahmen der Zeitreihenanalyse [3]. Daher besteht ein Bedarf an validen Imputationsmethoden, die in der Lage sind, die Vollständigkeit und Integrität der Daten zu bewahren. In diesem Kontext haben neuronale Netze und im Besonderen die Transformer-Architektur weitreichende Fortschritte erzielt. Neuartige Ansätze verwenden diese Transformer-Architekturen zur Imputation von Zeitreihendaten und bieten State-of-the-Art Ergebnisse. Dieselbe Architektur bietet im Bereich des Natural Language Processing (NLP) aktuell bemerkenswerte Fortschritte. Modelle in dieser Domäne werden Large Language Model (LLM) genannt und sind vor allem aufgrund ihrer Modellgröße sehr leistungsfähig [72]. In diesem Zuge gibt es eine Reihe von Versuchen die Fortschritte im Bereich des NLPs auf die Zeitreihendomäne zu übertragen und die dort bereits erzielten Ergebnisse für die Zeitreihendaten zu nutzen. Ein Transfer der LLMs auf die Zeitreihendomäne ist dabei möglich,

da die LLMs darauf trainiert sind, das nächste Token eines Satzes vorherzusagen, unvollständige Texte zu komplettieren oder Klassifikationen durchzuführen. Im abstrakteren Sinne sind die Modelle also darauf trainiert, Sequenzen einer gewissen Länge zu verarbeiten [23].

Das Ziel der vorliegenden Arbeit ist es, die Anwendbarkeit von LLMs im Kontext der Zeitreihenanalyse eingehend zu untersuchen. Hierzu werden diverse LLMs spezifisch für die Imputationsaufgabe modifiziert und optimiert. Dieser methodische Ansatz beabsichtigt das Gebiet der Zeitreihenverarbeitung zu erweitern und zu demonstrieren, inwiefern die Imputation von Zeitreihendaten durch den Einsatz von LLMs verbessert werden kann. Im weiteren Verlauf sollen die nachstehend formulierten Forschungsfragen einer detaillierten Untersuchung unterzogen werden.

1.2 Forschungsfragen

Im Zentrum der vorliegenden Arbeit soll die Frage beantwortet werden, inwieweit vortrainierte Sprachmodelle im Vergleich zu aktuellen State-of-the-Art Verfahren der Zeitreihenimputation performen und möglicherweise überlegen sind. In diesem Zusammenhang wird ein bereits vorhandenes Framework verwendet und entsprechend der Anforderungen angepasst. Innerhalb dieses Rahmens werden unterschiedliche LLMs ausgetauscht und miteinander verglichen. Weiterhin soll im Kontext der Leistung dieser Modelle geklärt werden, ob größere Sprachmodelle, gemessen an der Parameteranzahl, eine bessere Imputationsleistung erbringen und somit eine Abhängigkeit zwischen diesen Faktoren besteht. Die entsprechende Forschungsfrage (Research Question (RQ)) lautet wie folgt.

RQ1:
Inwieweit übertreffen vortrainierte Sprachmodelle die aktuellen State-of-the-Art Verfahren zur Imputation von Zeitreihendaten und spielt in diesem Kontext die Größe der ausgewählten Modelle eine entscheidende Rolle?

Neben diesen Aspekten soll des Weiteren untersucht werden, welche Komponenten der Sprachmodelle für die Anpassung an die Aufgabe der Imputation geeignet sind. Dies begründet sich darin, dass die großen Sprachmodelle aus unterschiedlichen Transformer-Komponenten bestehen [62] und eine vollständige oder nur teilweise Anpassung bereits gute Ergebnisse erbringen kann. Zudem wird angenommen, dass bestimmte Schichten der Transformer-Architektur den Großteil des Sprachverständnisses beinhalten. Hier gilt zu

untersuchen, ob eine Optimierung dieser Schichten für die Anpassung auf die Imputation notwendig ist. Es ergibt sich folgende Forschungsfrage.

RQ2:
Welche Modellkomponenten der Sprachmodelle eignen sich für ein Fine-Tuning im Kontext der Imputation von Zeitreihendaten?

Eine weiterführende offene Frage stellt die Wahl der Optimierungsmethode dar. Für das Fine-Tuning der LLMs haben sich unterschiedliche Methoden entwickelt, die sowohl ein ressourcensparendes sowie effizientes Fine-Tuning ermöglichen und im gleichen Zuge einen minimalen Performance-Verlust zeigen. Die Optimierungsmethoden wurden allerdings für die Anpassung der LLMs auf andere NLP-Tasks entwickelt [72]. Daher muss evaluiert werden, ob diese Methoden für die Anpassung auf Zeitreihenaufgaben geeignet sind.

RQ3:
Welchen Einfluss hat die Wahl der Optimierungsmethode, die für die Anpassung auf nachgelagerte NLP-Aufgaben entwickelt wurden, auf die Anwendbarkeit von LLMs bei der Imputation?

1.3 Gliederung

Zur Beantwortung der Forschungsfragen ist die vorliegende Arbeit in aufeinander aufbauende Kapitel unterteilt. Im Anschluss an dieses Kapitel folgt die Darstellung der notwendigen theoretischen Grundlagen. Hierzu gehören Inhalte bzgl. der Zeitreihenanalyse, der Datenimputation sowie den grundlegenden und weiterführenden Methoden zur Imputation von Zeitreihendaten. Darauf aufbauend werden die Transformer-Architektur sowie die LLMs vorgestellt. Nach den theoretischen Grundlagen folgt die Darstellung verwandter Arbeiten, welche die Transformer Architektur oder LLMs für die Vorhersage oder Imputation von Zeitreihen verwenden. Diese verwandten Arbeiten stellen den Ausgangspunkt für die in dieser Arbeit konzeptionierten Methodik dar, welche in Kapitel 4 dargestellt wird. Aufbauend auf dieser Methodik, die es ermöglicht unterschiedliche LLMs für die Imputation zu vergleichen, werden Experimente durchgeführt. Die Ergebnisdarstellung einschließlich des Versuchsaufbaus und der Diskussion der Ergebnisse folgt in Kapitel 5. Die Arbeit schließt in Kapitel 6 mit einem Fazit und Ausblick ab.

2 Theoretische Grundlagen

2.1 Zeitreihenanalyse

Das folgende Kapitel stellt die grundlegenden Konzepte der Zeitreihenanalyse dar und definiert zu Beginn die wichtigen Begrifflichkeiten. Im Anschluss folgen die Herausforderungen, von denen die Thematik fehlender Daten gesondert aufbereitet wird.

2.1.1 Definitionen

Definition Zeitreihe. Eine Zeitreihe umfasst eine Sequenz von Beobachtungen, die in regelmäßigen Zeitabständen gesammelt oder aufgezeichnet werden [33]. Jeder Datenpunkt einer Zeitreihe ist mit einem spezifischen Zeitpunkt verknüpft, sodass die Zeit eine Schlüsselvariable in diesen Daten ist. Zeitreihendaten entstehen häufig in großer Menge innerhalb von dynamischen Systemen. Die Daten werden in den Systemen durch physische oder virtuelle Sensoren (z.B. bei Online-Prozessen) gemessen [33]. Diese Art von Daten kommen in nahezu allen Bereichen und Domänen vor. Dazu gehören bspw. die Finanzwirtschaft (Verlauf von Aktienkursen), die Ökonomie (Käuferzahl, Absatzmenge) oder die Messung der örtlichen Temperatur [7].

Formal beschreiben lässt sich eine Zeitreihe als eine Serie von Datenpunkten, von denen jeder einem spezifischen Zeitpunkt zugeordnet ist [20]. Durch die Zuordnung eines Datenpunktes zu einem Zeitpunkt ist eine Zeitreihe

$$X = \{x_1, x_2, x_3, \ldots, x_N\}, \tag{2.1}$$

eine Sequenz von Datenpunkten in zeitlich geordneter Reihenfolge, wobei N die Länge der beobachteten Zeitreihe ist und $x_n \in \mathbb{R}^D$ ist definiert als

$$\{x_n^1, x_n^2, x_n^3, \ldots, x_n^D\}, \tag{2.2}$$

mit D, dass die Angabe der Feature Dimension darstellt [20]. Zu beachten ist, dass die Zeitunterschiede zwischen den Zeitschritten x_n und x_{n+1} nicht zwingend gleich sein müssen. Darüber hinaus können Zeitreihen univariat ($D = 1$) oder multivariat ($D > 1$) sein. Zum Beispiel stellt die Erfassung der Temperatur eines Ortes eine univariate Zeitreihe dar. Wird neben der Temperatur noch die Luftfeuchtigkeit erfasst und werden diese beiden Messgrößen kombiniert, liegt eine multivariate Zeitreihe vor [35]. Ein weiteres Beispiel für eine univariate Zeitreihe ist der Schlusskurs des DAX in Punkten (siehe Abb. 2.1).

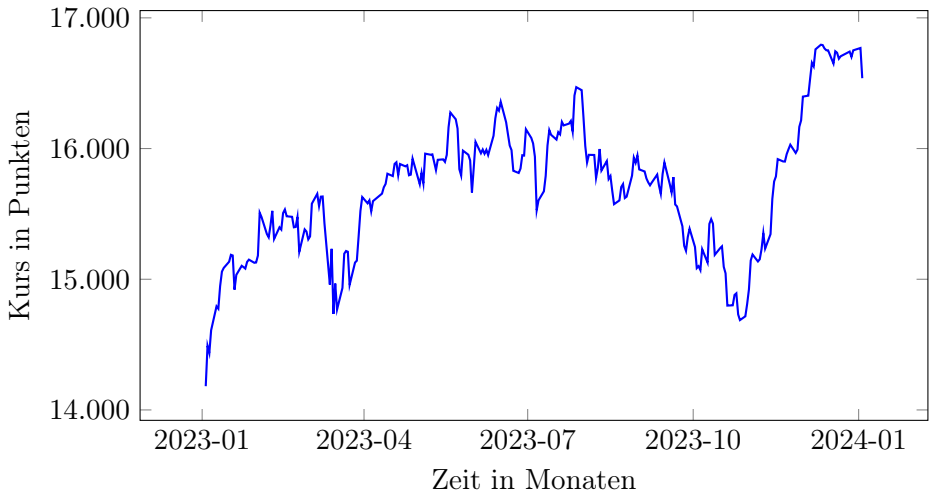

Abbildung 2.1: Zeitreihe des deutschen Aktienindex. Darstellung des Tagesschlusskurses in Punkten [9].

Definition Zeitreihenanalyse. Die Zeitreihenanalyse ist ein Bereich der Statistik und Datenanalyse, welcher sich mit der Untersuchung von Zeitreihen befasst. Ziel ist es, die zugrunde liegenden Strukturen und die dynamischen Beziehungen innerhalb der Daten zu verstehen, um Muster erkennen und Prognosen durchführen zu können, welche zu fundierten Entscheidungen führen [7, 33].

Die Teilbereiche der Zeitreihenanalyse sind vielfältig und umfassen folgende Bereiche: Vorhersage, Anomalieerkennung, Klassifikation und Imputation. Der erste Teilbereich befasst sich basierend auf historischen Beobachtungen mit der Vorhersage von zukünftigen Werten einer Zeitreihe [33]. Hier wird zwischen Single-Step-Ahead und Multi-Step-Ahead Vorhersagen unterschieden. Single-Step-Ahead Verfahren prognostizieren nur einen weiteren Wert einer Zeitreihe. Multi-Step-Ahead Vorhersagen dagegen prognostizieren eine Serie an Werten [33]. Neben der Vorhersage ist die Anomalieerkennung ein Teilbereich

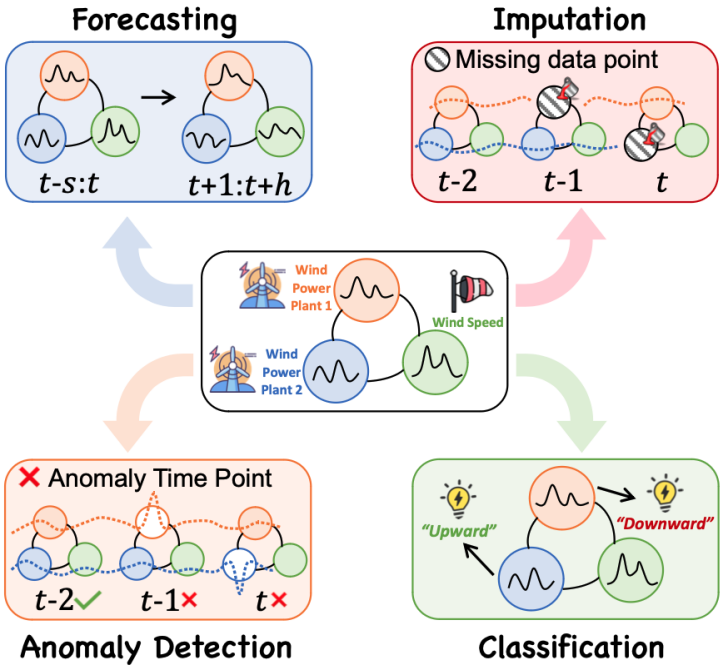

Abbildung 2.2: Teilbereiche der Zeitreihenanalyse. Zu den Bereichen gehören die Vorhersage (Forecasting), die Imputation, die Anomalieerkennung (Anomaly Detection) und die Klassifikation [33].

der Zeitreihenanalyse. Diese fokussiert sich auf die Erkennung von nicht normalen und unerwarteten Beobachtungen. Dazu gehört zum einen den Zeitpunkt des Auftretens zu bestimmen und zum anderen einen Grund für das Auftreten zu erheben. Aufgrund der allgemeinen Komplexität dieses Problems werden Modelle entwickelt, welche normales und anormales Verhalten voneinander abgrenzen können [33]. Das Hauptziel der Klassifikation von Zeitreihen ist die Zuordnung von Zeitreihen zu bestimmten Kategorien und Klassen auf Grundlage von Mustern und Charakteristiken. Als Beispiel könnten Verläufe von Aktienkursen betrachtet werden. Weisen die Zeitreihen spezifische Muster auf, können diese bspw. in die Kategorien *Kurs steigt* und *Kurs fällt* eingeteilt werden [33]. Eine weitere Teilaufgabe der Zeitreihenanalyse ist die Imputation [33]. Hierbei werden für unvollständige oder fehlende Daten Werte geschätzt. Das Ziel ist somit das Vervollständigen der Daten für nachgelagerte Aufgaben [35]. Abb. 2.2 fasst die einzelnen Teilbereiche der Zeitreihenanalyse zusammen.

Anschließend an die Definitionen einer Zeitreihe und der Analyse dieser folgen die Probleme und Herausforderungen, die mit der Analyse von Zeitreihen einhergehen. Im Mit-

telpunkt stehen die fehlenden Daten in Zeitreihen, welche durch die Imputation wieder vervollständigt werden.

2.1.2 Herausforderungen

Die Zeitreihenanalyse ist mit verschiedenen Herausforderungen konfrontiert, welche die Aufgaben der einzelnen Teilbereiche erschweren. Eine Herausforderung ist das Rauschen. Verrauschte Daten beziehen sich auf unerwünschte oder irrelevante Datenpunkte, die in den Daten verstreut sind. Dadurch wird es schwierig, klare Muster oder Trends in den Daten zu erkennen [63]. Eine weitere Herausforderung stellen Sprünge in Zeitreihen dar. Als Sprünge werden abrupte und unerwartete Veränderungen in den Zeitreihen bezeichnet. Diese werden bspw. durch Marktänderungen, Naturkatastrophen oder technologische Anpassungen verursacht. Für ein Vorhersagemodell oder Modell zur Entdeckung von Anomalien ist das Vorkommen von Sprüngen problematisch, da diese einen direkten Einfluss auf die Stabilität und Vorhersagbarkeit der Zeitreihen haben [63]. Als weitere Herausforderung werden Trendveränderungen beschrieben. Ein Trend innerhalb einer Zeitreihe kann sich über einen gewissen Zeitraum verändern. Für die exakte Vorhersage von Zeitreihen ist es daher wichtig, Trendveränderungen zu erkennen und entsprechend zu interpretieren. Eine Nichtbeachtung dieser Komponente führt zu fehlerhaften Prognosen [63]. Des Weiteren spielt das Fehlen von Daten bzw. unvollständige Datensätze eine große Rolle in Zeitreihen. Die Ursachen für das Fehlen von Datenpunkten können sehr unterschiedlich sein [58]. Die Herausforderung der fehlenden Werte soll im nächsten Abschnitt gesondert behandelt und auch die Arten von fehlenden Daten aufgrund der Relevanz für die Datenimputation beschrieben werden.

2.1.3 Fehlende Daten in Zeitreihen

Fehlende Daten in Zeitreihen stellen eine Herausforderung dar und werden meist durch unerwartete Ereignisse wie den Ausfall eines Sensors oder Übertragungsfehler in Internet of Things (IoT) Prozessen verursacht [20]. Diese Lücken beeinträchtigen die Qualität der Zeitreihendaten und erhöhen die Komplexität ihrer Nutzung für nachgelagerte Analysen und Aufgaben [58].

In der Statistik werden drei unterschiedliche Arten von fehlenden Werten unterschieden. Der erste Typ ist *Missing completely at random (MCAR)*, welcher das völlig zufällige

Fehlen von Werten beschreibt [3]. MCAR tritt auf, wenn die fehlenden Daten durch zufällige Ereignisse entstehen. Beispiele dafür sind das zufällige Zerbrechen von Blutproben, der Ausfall eines Sensors oder der Verlust von Fragebögen [18]. Der zweite Typ fehlender Daten wird als *Missing not at random (MNAR)* bezeichnet. MNAR bezieht sich auf Situationen, in denen die Wahrscheinlichkeit des Fehlens einer Beobachtung von der nicht erfassten Beobachtung selbst abhängt [49]. Diese Art von fehlenden Daten ist zu betrachten, wenn bspw. ein Sensor Messungen vornimmt und Werte auftreten, die außerhalb des Messbereichs liegen. Dann kann der Wert nicht gemessen werden, obwohl ein Wert existiert [48]. Die letzte Art fehlender Daten wird als *Missing at random (MAR)* bezeichnet. MAR bedeutet, dass die Wahrscheinlichkeit, dass die Beobachtung fehlt, von Informationen abhängig ist, die beobachtet worden sind. Das Fehlen ist daher in anderen beobachteten Werten begründet [36]. Diese unterschiedlichen Arten von fehlenden Daten erfordern spezifische Ansätze in der Datenanalyse, um die Genauigkeit und Zuverlässigkeit der Ergebnisse zu gewährleisten. Im Kontext dieser Arbeit beziehen sich fehlende Daten im Folgenden speziell auf den Typ MCAR.

In [58] wird eine weitere Einteilung fehlender Daten beschrieben, die sich auf die Anzahl fehlender Datenpunkte in aufeinanderfolgenden Sequenzen beziehen. Danach werden fehlende Instanzen basierend auf den benachbarten Datenpunkten in zwei Arten unterteilt. Die zwei Arten unterscheiden sich wie folgt:

- Instanz t_{i-1} und Instanz t_{i+1} fehlen nicht
- Als minimale Anforderung fehlen mindestens t_{i-1} oder t_{i+1}

Im ersten Fall ist die Instanz eine isolierte fehlende Instanz. Im zweiten Fall ist die Instanz Teil einer Sequenz von fehlenden Werten. Grafisch bildet dies Abb. 2.3 ab. Das linke Diagramm der Abbildung zeigt isolierte Instanzen, bei denen die benachbarten Instanzen nicht fehlen. Auf der rechten Seite der Abbildung zeigt das Diagramm Sequenzen fehlender Daten.

Nachdem die Definitionen und die analytischen Ansätze von Zeitreihen sowie deren spezifischen Teilbereiche dargelegt wurden, erfolgte eine eingehende Diskussion der verschiedenen Typen fehlender Daten in Zeitreihen. Das nun folgende Kapitel widmet sich der Imputation dieser fehlenden Werte.

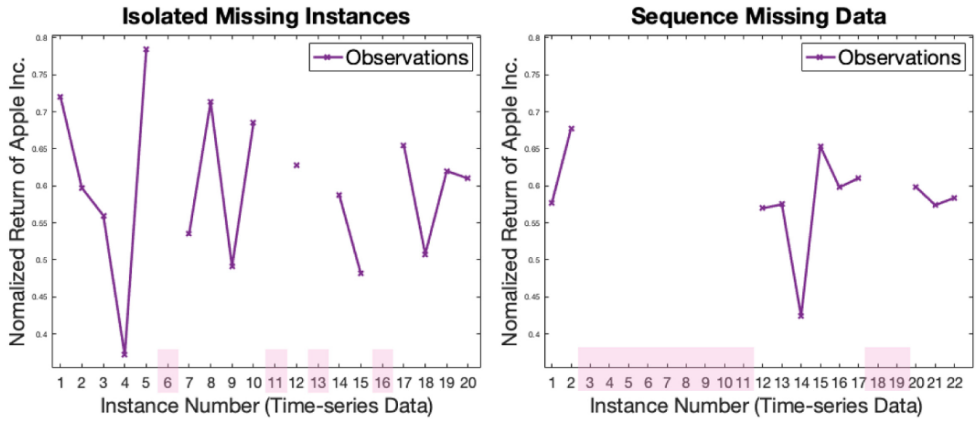

Abbildung 2.3: Beispiele für isolierte und sequenziell fehlende Daten. Die Nummern der fehlenden Instanzen sind Lila gekennzeichnet. Links fehlen nur einzelne Instanzen (siehe 6, 11 oder 13). In der rechten Abbildung fehlen Sequenzen von Datenpunkten (siehe 3-11 oder 18-19) [58].

2.2 Imputation

Am Anfang dieses Kapitels wird die Definition sowie Relevanz der Imputation herausgestellt. Anschließend erfolgt eine detaillierte Betrachtung klassischer Methoden der Datenimputation. Zum Abschluss wird eine kritische Beschreibung der Grenzen dieser herkömmlichen Verfahren vorgenommen, um die Basis für weiterführende Methoden und Ansätze zu bilden.

2.2.1 Definition und Bedeutung

Die Imputation beschreibt das Auffüllen bzw. Ersetzen von fehlenden Datenpunkten [33]. Wie bereits beschrieben, ist die Imputation ein Teilbereich der Zeitreihenanalyse. Allerdings ist die Imputation auch in anderen Anwendungsbereichen zu finden. In der Zeitreihendomäne stellen die fehlenden Daten eine besondere Herausforderung für nachgelagerte Aufgaben dar, da fehlende Daten einer der Hauptgründe für verzerrte Ergebnisse sind [3]. Daher ist die Imputation aufgrund ihrer Relevanz ein weiterhin aktives Forschungsthema [58].

Die Anwendung einer Imputationsmethode als Preprocessing-Schritt hat eine hohe Bedeutung, da dies einen entscheidenden Einfluss auf die Qualität der Daten hat [36]. Ohne die Vorverarbeitung müssen die Daten mit fehlenden Datenpunkten aus dem Datensatz

entfernt werden. Ein Entfernen der Daten würde zum einen die Größe des Datensatzes und die Anzahl an Trainingsbeispielen für Machine Learning (ML) oder Deep Learning (DL) Modelle reduzieren. Zum anderen resultiert das Entfernen in der Einführung eines Bias. Die Datenreduktion hat daher direkten Einfluss auf die Güte zu trainierender Modelle [3]. Darüber hinaus geht das Entfernen von Datensätzen mit einem Verlust von Informationen einher [58]. Die Behandlung von fehlenden Datenpunkten einer Zeitreihe ist somit notwendig, sodass eine Vielzahl an Methoden entstanden sind.

2.2.2 Klassische Methoden

Methoden der Imputation werden je nach vorliegender Literatur unterschiedlich kategorisiert. In dieser Arbeit wird eine Unterteilung in Anlehnung an [20] vorgenommen und diese entsprechend der Zielsetzung angepasst. Zu den Kategorien zählen naive Methoden, statistische Ansätze sowie ML basierte Verfahren. DL Methoden werden in Kapitel 2.3 gesondert behandelt.

Naive Methoden. In diesen Bereich der Methoden fallen primär Löschmethoden (engl. Deletion based Methods). Bei diesen Verfahren werden die Zeitreihen mit fehlenden Beobachtungen aus dem Datensatz entfernt [20]. Neben dem Löschen der Datensätze können die fehlenden Beobachtungen innerhalb der Zeitreihen auch ignoriert und nicht behandelt werden [49]. Diese Ansätze sind allerdings nur anwendbar, wenn die Rate fehlender Werte nicht mehr als fünf Prozent beträgt und somit die nachgelagerten Methoden nicht beeinflusst [20]. Ab einer höheren Rate sind die naiven Methoden nicht mehr anzuwenden. Trotz der einfachen Anwendung dieser Ansätze wird das Löschen der Daten meist vermieden und gilt als eine der am wenigsten effektiven Methoden [48].

Statistische Methoden. Neben den naiven Ansätzen können Werte auf Grundlage von statistischen Methoden imputiert werden [42]. Zu den elementaren Techniken in diesem Bereich gehören beispielsweise die Imputation durch Mittelwert oder Median, wie in [53] beschrieben. Ein zusätzliches Verfahren stellt die *lineare Interpolation* dar. Hier wird eine Funktion zwischen zwei Punkten derart optimiert, dass fehlende Werte innerhalb dieses Intervalls approximiert werden können [58]. Im Gegensatz zu anderen Verfahren berücksichtigen statistische Modelle vergangene sowie zukünftige Beobachtungen bei der Berechnung der imputierten Werte. Dies ermöglicht es, Informationen und Zusammenhänge auch aus übergreifenden Datenpunkten zu extrahieren und nicht nur auf vergangene Beobachtungen angewiesen zu sein [20].

ML basierte Methoden. Weitere Methoden der Imputation entstammen dem Bereich des ML. Diese Methoden übertreffen naive und traditionelle statistische Imputationstechniken, indem sie komplexe Muster und Beziehungen innerhalb der Daten lernen und nutzen, um präzise Werte zu generieren [48]. Einer dieser Ansätze verwendet Zusammenhänge zu den nächsten Nachbarn. Diese *Neighbor based Methods* wenden Clustering Algorithmen, wie z.B. kNN oder DBSCAN, an, um die ähnlichsten Beobachtungen zu erfassen. Auf Grundlage der n nächsten Nachbarn wird dann ein geschätzter Wert angenommen und als imputierter Wert eingesetzt [20]. Zu den ML basierten Methoden zählt auch die Support Vector Machine (SVM), welche zur Imputation von fehlenden Werten verwendet wird. Die SVM wird durch Supervised Training darauf trainiert, die optimale Annäherung an einen fehlenden Wert auszugeben. Dafür nutzt diese Methode die temporalen Informationen einer Zeitreihe. Aufgrund ihrer Funktionsweise, die hier nicht im Detail vorgestellt werden soll, ist die SVM dabei ressourcensparend und effizient in der Berechnung [58].

2.2.3 Limitationen klassischer Imputationsmethoden

Die elementarsten Verfahren klassischer Imputationsmethoden zeichnen sich durch eine einfache Implementierung aus, resultieren jedoch in einer Reduktion des Datenvolumens. Dies kann für nachgelagerte Aufgaben der Zeitreihenanalyse problematisch sein. Zudem kann die Anwendung dieser Methoden zu einer Verzerrung der Daten führen, falls die ersetzten Werte nicht adäquat berechnet werden [48]. Die fortschrittlicheren Methoden haben den Vorteil, dass sie die Größe des Datensatzes erhalten sowie die statistische Aussagekraft zu einem höheren Grad bewahren können. Die Anwendung der ML basierten Methoden gestaltet sich ebenfalls einfach [48]. Das größte Problem auch dieser Methoden ist allerdings die Erfassung langfristiger temporaler Abhängigkeiten. Die fortgeschrittenen Techniken berücksichtigen zwar Zusammenhänge innerhalb der Daten, sind aber nicht ausreichend performant in der Abstraktion wieder aufkommender Muster und anderer Abhängigkeiten [36]. Daher sind diese Methoden für die Imputation langer und komplexer Zeitreihen nur bis zu einem gewissen Grad geeignet.

Die in diesem Abschnitt erörterten Methoden unterstreichen die Notwendigkeit einer Weiterentwicklung der Imputationsmethoden. Vor diesem Hintergrund untersucht das nachfolgende Kapitel Techniken, welche auf Deep Learning basieren.

2.3 Deep Learning Methoden zur Imputation von Zeitreihen

Statistische und herkömmliche maschinelle Lernverfahren können komplexe temporale Abhängigkeiten und nicht-lineare Beziehungen innerhalb von univariaten oder multivariaten Zeitreihen nicht abbilden [33]. Einige der Verfahren basieren zudem auf bestimmten Annahmen bezüglich fehlender Werte, was zu Verzerrungen oder systematischem Bias führen kann [11]. Neuronale Netze dagegen sind in ihrer erweiterten Form dazu in der Lage, diese Abhängigkeiten und Beziehungen innerhalb der Daten zu abstrahieren und übertreffen die vorherigen Methoden in der Genauigkeit der Imputation [36]. Aufgrund dieser Überlegenheit werden die relevanten Modelle im nachfolgenden Abschnitt behandelt. Um ein fundiertes Verständnis dieser Modelle zu ermöglichen, wird zunächst grundlegendes Wissen über neuronale Netze vermittelt, gefolgt von einer detaillierten Vorstellung verschiedener Modellarchitekturen.

2.3.1 Künstliche Neuronale Netze

Ein künstliches neuronales Netz ist ein lernfähiges System, dass aus einer Reihe von miteinander verbundenen Einheiten, den Neuronen, besteht [25]. Diese Neuronen sind in Schichten angeordnet und können über einen Trainingsprozess komplexe Muster in Daten erkennen. Typischerweise sind alle Neuronen des Netzwerkes miteinander verbunden. Diese klassische Art des neuronalen Netzes wird auch *Fully connected Neural Network* genannt. Besitzt das Netzwerk mehr als eine Hidden Layer ist es ein *Deep Neural Network* [25].

Die Schichten eines neuronalen Netzes sind die Input- und Output-Layer sowie eine versteckte Schicht, die Hidden Layer [54]. Die Input-Schicht empfängt die zu verarbeitenden Daten. Die Hidden Layer verarbeitet die Eingaben x durch gewichtete Verbindungen. Die Funktionsweise eines einzelnen Neuronen ist in Abb. 2.4 abgebildet. Der Input x der vorherigen Schichten wird über die Gewichte w summiert. Anschließend entscheidet eine Aktivierungsfunktion f darüber, ob und wie stark das Neuron aktiviert wird. Das Ergebnis der Aktivierungsfunktion y ist dann wiederum der Input für die nächste Schicht oder das Ergebnis der Output-Layer [54]. Damit das neuronale Netz als System trainiert werden kann, wird der Backpropagation Algorithmus angewendet. Dieser Algorithmus verwendet einen berechneten Fehler zwischen der Ausgabe des Netzes und der wahren

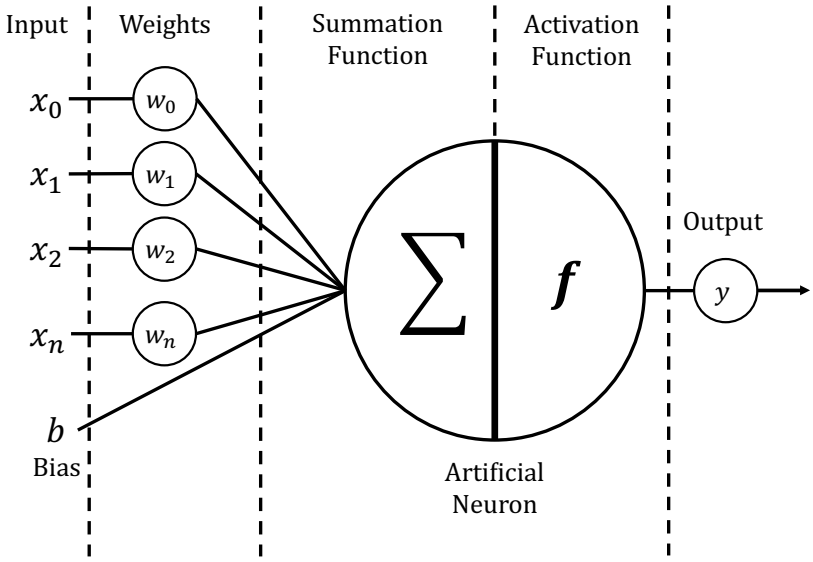

Abbildung 2.4: Darstellung eines künstlichen Neurons [54]. Die Inputfeatures werden über eine Summenfunktion unter Berücksichtigung der Gewichte aggregiert. Anschließend entscheidet die Anwendung einer Aktivierungsfunktion darüber, ob das Neuron schaltet.

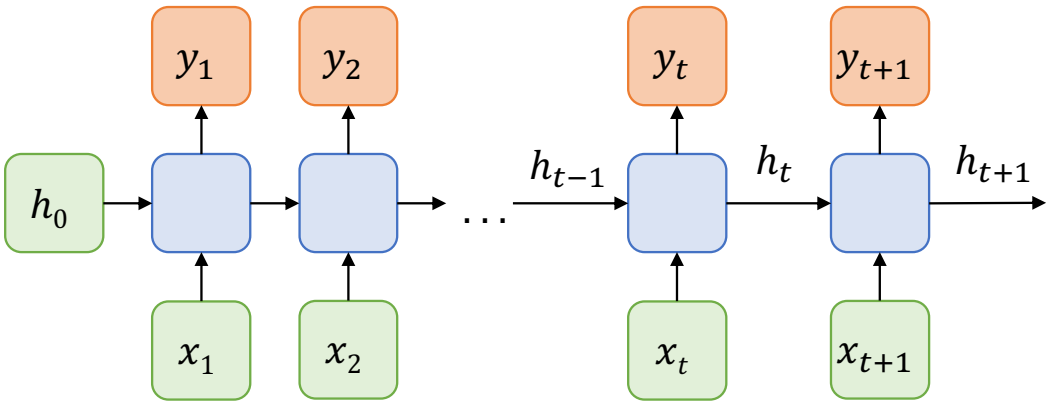

Abbildung 2.5: Struktur eines RNNs. Durch die Weitergabe von verdeckten Zuständen (h_t) kann das RNN Informationen vergangener Zustände nutzen, um die Vorhersagen folgender Tokens zu verbessern. In Anlehnung an [5].

Ausgabe, um die Gewichte im Netzwerk anzupassen. Der Fehler wird dabei durch eine Loss-Funktion berechnet [54].

2.3.2 Rekurrente Neuronale Netze

Rekurrente neuronale Netze (engl. Recurrent Neural Network (RNN)) repräsentieren eine Art künstlicher neuronaler Netze, welche sich von den normalen tiefen Netzen durch die Integration eines inneren Zustands unterscheiden. Diese Eigenschaft ermöglicht es den Netzen, sequenzielle Daten mit temporalen Abhängigkeiten zu verarbeiten [40]. Diese Fähigkeit hat RNNs nicht nur in NLP-Szenarien zu einer verbreiteten Anwendung verholfen, sondern auch ihre Einsatzmöglichkeiten in der Zeitreihenanalyse erweitert [27]. Anfangs primär zur Vorhersage von Zeitreihendaten eingesetzt, finden die RNN Modelle zunehmend auch Anwendung in der Imputation von Zeitreihen [40, 53]. In Abb. 2.5 wird die Struktur eines RNNs dargestellt.

Architektur eines RNNs. Ein RNN hat im Vergleich zu einem klassischen neuronalen Netz einen inneren Zustand, welcher auch *Hidden State* genannt wird. Der Hidden State kann als kompakte Zusammenfassung vergangener Informationen verstanden werden [40]. Dieser Zustand wird zu jedem Zeitpunkt rekursiv mit neuen Beobachtungen aktualisiert und zwischen den aufeinanderfolgenden Zeitschritten der Zeitreihe ausgetauscht (siehe Abb. 2.5 h_t, h_{t+1}, ...) [22]. Dementsprechend ist der Wert des vorherigen Zustands h_{t-1} sowohl in der Berechnung des nächsten versteckten Zustands als auch in der Berechnung

der Ausgabe y_t enthalten [40]. Der Hidden State h_t sowie der Output y_t wird wie folgt berechnet:

$$h_t = \sigma(x_t \cdot W_i + h_{t-1} \cdot U_i + b_i), \tag{2.3}$$

$$y_t = f(V_i \cdot h_t + c_i), \tag{2.4}$$

mit den Parametern W_i, U_i und V_i als Gewichtsmatrizen sowie b_i und c_i als Bias-Vektoren. σ und f sind Aktivierungsfunktionen, wie z.B. die Sigmoid-Funktion [11].

Eine konkrete Anwendung eines RNNs zur Imputation von Zeitreihen wird in [11] beschrieben. Hier wird die Architektur des Bidirectional Recurrent Imputation for Time Series (BRITS) vorgestellt. BRITS basiert auf bidirektionalen RNNs, welche zum Zeitpunkt der Veröffentlichung des Papers (2018) State-of-the-Art Ergebnisse bei der Imputation von Werten erreichen. In der praktischen Implementierung verwendet BRITS allerdings Long-Short Term Memory (LSTM) Zellen, die eine Weiterentwicklung der RNN-Komponenten sind. Dies begründen die Autoren mit den Schwächen der klassischen Struktur eines RNNs. Insbesondere die Modellierung langer Sequenzen stellt eine Herausforderung dar, da der inhärente Aktualisierungsprozess dazu führen kann, dass Verbindungen und Abhängigkeiten in den Daten über einen langen Zeitraum nicht mehr adäquat wiedergegeben werden [53]. Zusätzlich tritt bei der Anwendung von RNNs das Problem von explodierenden und verschwindenden Gradienten (engl. Exploding, Vanishing Gradients) auf, welches signifikante Einflüsse auf die Kapazität dieser Modelle hat [61].

2.3.3 Long-Short Term Memory Modell

LSTMs sind eine Unterart von RNNs und wurden entwickelt, um lange Sequenzen und die inhärenten Abhängigkeiten einer Sequenz verarbeiten zu können [61]. Dies wird dem Modell dadurch ermöglicht, dass es aktiv regulieren kann, welche Informationen weitergegeben werden sollen und welche nicht. Eine typische LSTM Zelle besteht aus einem Input Gate, einem Forget Gate und einem Output Gate (siehe Abb. 2.6). Zudem besitzt die Zelle einen Zustand (Cell State), welcher durch die drei Gates reguliert wird und zusätzlich zum Hidden State des Netzes jeder LSTM Zelle innewohnt. Dieser Cell State ist daher für die Speicherung von Informationen über ein längeres Zeitintervall zuständig [40]. Das Forget Gate war anfangs nicht Bestandteil des Aufbaus, wurde aber von Gers et al. [21] integriert, um es der Zelle zu ermöglichen, den inneren State zurückzusetzen [61].

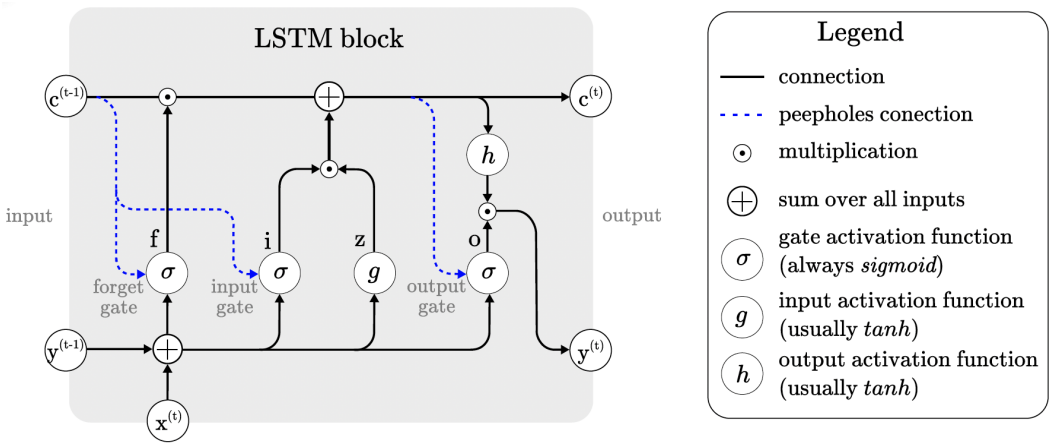

Abbildung 2.6: Architektur einer typischen LSTM Zelle [61].

Die drei Gates bestimmen somit über den Zustand der Zelle und werden wie folgt berechnet:

$$i_t = \sigma(W_i \cdot x_t + R_i \cdot y_{t-1} + P_i \odot c_{t-1} + b_i), \tag{2.5}$$

$$f_t = \sigma(W_f \cdot x_t + R_f \cdot y_{t-1} + P_f \odot c_{t-1} + b_f), \tag{2.6}$$

$$o_t = \sigma(W_o \cdot x_t + R_o \cdot y_{t-1} + P_o \odot c_{t-1} + b_o), \tag{2.7}$$

wobei $\odot$ die elementweise Multiplikation zweier Vektoren bezeichnet, W_x, R_x, P_x die mit x_t, y_{t-1}, c_{t-1} assoziierten Gewichte sind und b_x den Bias darstellt.

Der Zellzustand c_t wird dann auf Grundlage des Inputgates i_t und des Forgetgates f_t unter Berücksichtigung des Block-Inputs z_t kalkuliert:

$$c_t = z_t \odot i_t + c_{t-1} \odot f_t \tag{2.8}$$

Abschließend wird der Output der LSTM Zelle sowie der Hidden State berechnet und dabei der aktuelle Zellstatus sowie der Wert für das Output Gate miteinander kombiniert:

$$y_t = h_t = tanh(c_t) \odot o_t \tag{2.9}$$

Durch die hier dargestellte Architektur und die Berechnung der Zellzustände mit Hilfe von Gates wird das Problem der Exploding, Vanishing Gradients gelöst. Dies führte zu weiteren Modellarchitekturen, die neue, fortschrittlichere Gate-Ansätze implementieren. Dazu gehört unter anderem die Gated Recurrent Unit (GRU) Architektur [15]. Auch

die Architektur des GRUs eignet sich besonders zur Verarbeitung von Zeitreihendaten und die Imputation [53]. Im Vergleich zu einem LSTM besitzt ein GRU nur zwei Gates, das Update und Reset Gate, und vereinfacht die im Vorwege beschriebene Struktur [15]. Durch diese Anpassungen sind die GRUs effizienter in der Berechnung [53]. Ein auf GRU basierender Ansatz zur Zeitreihen-Imputation ist GRU-D [20]. Die Autoren führen einen Zerfallsmechanismus (Decay-Mechanism) ein, um fehlende Daten zu imputieren. Dieser Mechanismus beruht auf der Annahme, dass Werte von fehlenden Variablen dazu neigen, sich einem Standardwert zu nähern, wenn die letzte Beobachtung lange zurückliegt. Durch die eingeführte Rate wird der Einfluss des aktuellen Wertes auf die Vorhersage gesteuert und das Modell erreicht vergleichbare Performance mit anderen zu dieser Zeit veröffentlichten Ansätzen [14].

Nach der Betrachtung von GRUs und deren spezifischen Anpassungen zur effizienten Imputation von Zeitreihendaten, leitet dieser Abschnitt nun zu einem noch fortschrittlicheren Bereich der künstlichen Intelligenz über: den Large Language Models, insbesondere der Transformer-Architektur und dem Aufmerksamkeitsmechanismus. Konventionelle sequenzielle Modelle, wie die beschriebenen RNNs, LSTMs und GRUs, stoßen bei sehr langen und komplexen Sequenzen auf Herausforderungen, die mit dem vorgestellten Design nicht zu lösen sind. Aufgrund der Verwendung von Zuständen verlieren Informationen über eine lange Sequenz an Relevanz. Trotz dessen können die Informationen für ein zu generierendes Token von Bedeutung sein. Diese Problematik hat sich zudem in unterschiedlichen Versuchen gezeigt, sodass eine Weiterentwicklung notwendig ist [46]. Der Attention-Mechanismus adressiert genau diese Herausforderungen, indem er die Modellierung langer Abhängigkeiten verbessert und eine dynamische Gewichtung der Relevanz einzelner Token ermöglicht. Diese innovative Technologie bietet neue Perspektiven für anspruchsvolle NLP-Aufgaben aber auch für die Verarbeitung und Imputation von Zeitreihendaten. Um ein fundiertes Verständnis für diese fortschrittlichen Modelle zu entwickeln, ist es zunächst notwendig, sich mit den zugrunde liegenden Technologien und Konzepten zu befassen. Dazu zählen insbesondere der Self-Attention-Mechanismus und die Transformer-Architektur, welche die Basis für die meisten modernen Language Models bilden. Die Auseinandersetzung mit diesen Grundlagen bildet den Ausgangspunkt für die anschließende Erörterung von Language Models in der Anwendung auf die Zeitreihen-Imputation, wobei ein besonderes Augenmerk auf die spezifischen Eigenschaften und Vorteile dieser Modelle in diesem Anwendungsbereich gelegt wird.

2.4 Language Models

Language Models sind Modelle zur Abbildung von Konzepten natürlicher Sprache und werden im Bereich des NLPs entwickelt und angewendet. Typischerweise sind die Sprachmodelle darauf trainiert, das nächste Wort eines Satzes vorherzusagen und Sätze oder ganze Absätze zu bilden. Abstrahiert bedeutet dies, dass die Modelle das nächste Token einer Sequenz vorhersagen können. Dieses Muster kommt dem Prognostizieren von nächsten Werten innerhalb einer Zeitreihe sehr nahe. Neben der Prognose können die Modelle somit auch für die Imputation von Daten verwendet werden. Verschiedene Arbeiten [10, 12, 34] verwenden große Sprachmodelle bereits zur Prognose von Zeitreihen, wenige aber erst für die Imputation der Daten [74]. Ein Grundverständnis der Basiskonzepte von Sprachmodellen ist essenziell, um deren Anwendung auf den Bereich der Zeitreihen nachvollziehen zu können. Daher folgen die Grundlagen von Sprachmodellen, der Transformer-Architektur und dem Aufmerksamkeitsmechanismus als Ausgangspunkt für die Entwicklungen im NLP-Bereich.

2.4.1 Transformer-Modelle

Attention Mechanism

Der Aufmerksamkeitsmechanismus (engl. Attention Mechanism) wurde erstmals von Bahdanau et al. [8] eingeführt und als neuartiger Ansatz für maschinelle Übersetzungen im NLP Umfeld entwickelt. Auf Grundlage des Attention Mechanism sind Weiterentwicklungen entstanden, die auch heute noch einen nachhaltigen Einfluss auf die Entwicklungen der Architekturen haben [46]. Der Mechanismus ermöglicht neuronalen Netzen einen Fokus oder auch *Aufmerksamkeit* auf bestimmte Elemente des Inputs zu legen und damit Informationen entsprechend ihrer Relevanz zu gewichten [46]. Vor Einführung des Attention Mechanism waren Encoder-Decoder basierte Ansätze zur Modellierung von Sprachsequenzen rein auf den Hidden States des Encoders angewiesen. Encoder-Decoder Architekturen strukturieren den Prozess der Übersetzung in zwei Hauptkomponenten: dem Encoder und dem Decoder [57]. Der Encoder ist dafür verantwortlich, die Eingabesequenz (bspw. ein Satz oder auch eine Zeitreihe) in eine Zwischenrepräsentation als Vektordarstellung zu überführen. Diese Zwischenrepräsentation beinhaltet die wesentlichen Informationen der Eingabesequenz und dient damit als Kontext für den Decoder (siehe Abb. 2.7, Vanilla Encoder Decoder). Der Decoder nutzt diese Informationen, um

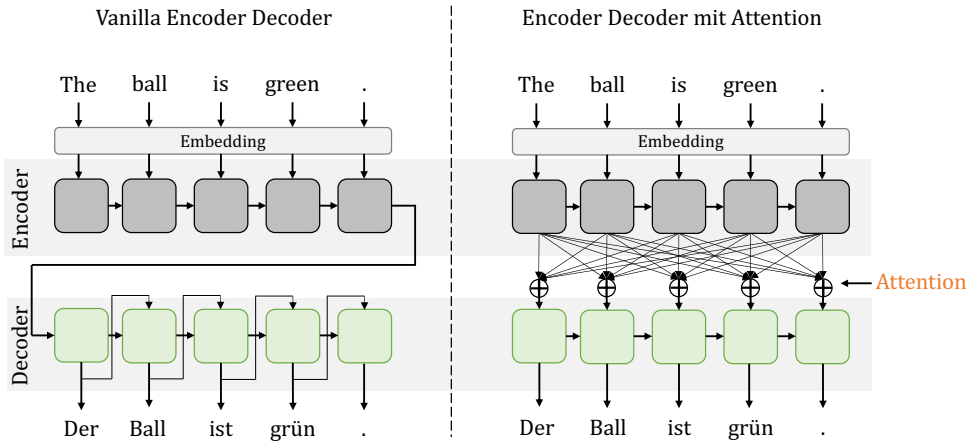

Abbildung 2.7: Encoder-Decoder-Architektur mit (rechts) und ohne (links) Aufmerksamkeitsmechanismus. Der Vanilla Encoder Decoder Ansatz verwendet eine innere Repräsentation beim Übergang von Encoder zu Decoder. Mit dem Attention Mechanism ist es dem Modell möglich, auf alle Tokens gleichermaßen zuzugreifen. In Anlehnung an [31, 8].

die Ausgabesequenz, hier eine deutsche Übersetzung, zu generieren [57]. Die Erweiterung um die Aufmerksamkeit ermöglicht es, bei der Generierung jedes Tokens spezifischen Fokus auf unterschiedliche Teile des Inputs zu legen. Darüber hinaus können auch zukünftige Tokens bei der Vorhersage des aktuellen Tokens durch den Decoder mit berücksichtigt werden (siehe Abb. 2.7, Encoder Decoder mit Attention) [8]. Der vollständige Prozess des Attention Mechanism ist in Abb. 2.8 abgebildet und die einzelnen Schritte des Prozesses werden anhand dieser Abbildung erläutert.

Encoding. Der Prozess beginnt mit dem Encoding der Input-Sequenz. Die Input-Sequenz $(X_1, X_2, ..., X_T)$ wird von einem bidirektionalen RNN verarbeitet, sodass Hidden States für jeden Input zum einen vorwärts-gerichtet $\overrightarrow{h_t}$ sowie rückwärts-gerichtet $\overleftarrow{h_t}$ berechnet werden. Der Encoder erhält für jeden Input zu Zeitpunkt t einen Hidden State, welcher aus der Verkettung beider Hidden States besteht [8]. Auf diese Weise enthält h_j sowohl Informationen der vorangehenden sowie nachfolgenden Wörter [46]. Dies ist in Abb. 2.8 im unteren Bereich abgebildet.

$$h_j = \left[\overrightarrow{h}_j^T; \overleftarrow{h}_j^T\right]^T \tag{2.10}$$

Attention Weights. An die Berechnung der Hidden States schließt die Kalkulation der Aufmerksamkeitsgewichte $a_{t,j}$ an. Dafür wird zunächst das Gewicht für jeden Zeitschritt

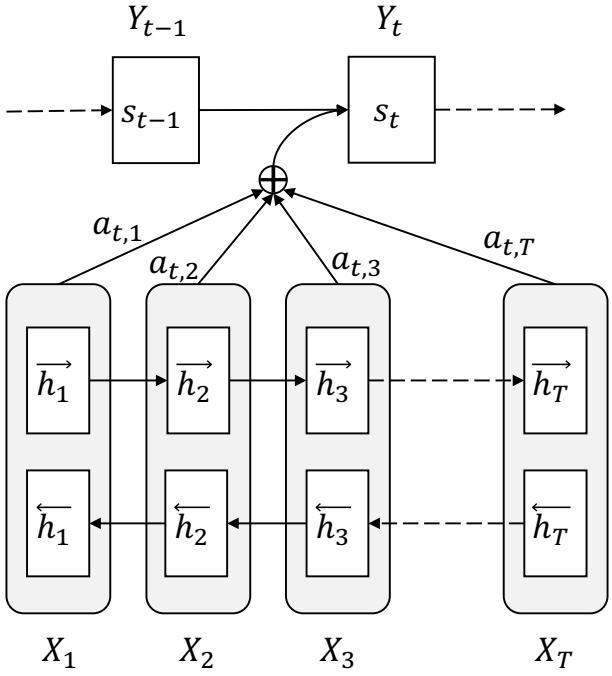

Abbildung 2.8: Berechnung der Attention Weights [8].

t des Decoders und alle Hidden States j des Encoders iterativ verarbeitet:

$$e_{tj} = a(s_{t-1}, h_j), \tag{2.11}$$

mit a als beliebige Funktion. Folgend werden auf alle Hidden States h_j des Encoders und alle vorangegangenen Hidden States des Decoders s_{t-1} eine Funktion zur Berechnung des Einflusses angewendet. Die Attention Weights werden anschließend durch eine Softmax-Funktion normalisiert, sodass höhere Werte auf einen höheren Einfluss Rückschluss geben und $0 \leq \alpha_{t,j} \leq 1$ [8].

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})} \tag{2.12}$$

Decoding. Der Schritt des Dekodierens (engl. Decoding) benötigt neben den berechneten Attention Weights darüber hinaus noch die Berechnung eines Kontextvektors. Zu jedem Zeitpunkt t wird der Kontextvektor als eine gewichtete Summe der codierten Hidden States h_j berechnet:

$$c_t = \sum_{j=1}^{T} a_{tj} h_j \tag{2.13}$$

Der Kontextvektor c_t fasst die für das aktuelle Ausgabetoken relevanten Informationen der gesamten Eingabesequenz zusammen und dient als Eingabe für den Decoder, um das nächste Token vorherzusagen. Unter Verwendung des Kontextvektors c_t, des vorherigen Zustands des Decoders s_{t-1} und des zuletzt generierten Tokens y_{t-1} berechnet der Decoder den nächsten Zustand s_t und anschließend die Wahrscheinlichkeit für das nächste Wort y_t [8].

Zusammenfassend lässt sich die Berechnung der Attention Weights als Prozess verstehen, welcher die Input Sequenz verarbeitet und anschließend für jedes einzelne Input Token eine Gewichtung festlegt. Diese Gewichtung drückt eine Art Einfluss der Tokens auf das vorhergesagte Wort aus. Durch diese Vorgehensweise können neuronale Netze Schlüsselwörter in Sequenzen erlernen und komplexe Muster abstrahieren [8].

Transformer Architektur

Die ursprüngliche Transformer Architektur wurde 2017 von einem Google Research Team in dem Paper *Attention Is All You Need* vorgestellt. Der Transformer ist ein neuronales Netz mit einer neuartigen Architektur, welches für NLP-Aufgaben entwickelt wurde [62]. Vorherige Ansätze beruhten auf rekurrenten Netzen, wie dem RNN, LSTM oder GRU. Diese Netze wurden zu diesem Zeitpunkt auch weiterhin verwendet und in einer Encoder-Decoder Struktur angeordnet, sowie unter Anwendung des Attention Mechanism erweitert. Trotz der Vorteile der Integration des neuen Aufmerksamkeitsmechanismus bestehen weiterhin die Nachteile der rekurrenten Netzarchitekturen, sodass die Modellierung von langen Sequenzen und auch die Berücksichtigung temporaler Abhängigkeiten ein Problem bleiben [56]. Den Nachteilen der vorherigen Modellarchitekturen begegnet die Transformer-Architektur dahingehend, dass alle RNN-Bestandteile verworfen werden und nur noch der Attention Mechanism zentraler Gegenstand der Architektur ist. Für eine zielführende Architektur zur Lösung moderner NLP-Aufgaben wird der Aufmerksamkeitsmechanismus in eine neue Struktur mit zusätzlichen Komponenten überführt. Im Folgenden werden diese Komponenten vorgestellt.

Die Transformer Architektur besteht aus einem Encoder (links) sowie einem Decoder (rechts) (siehe Abb. 2.9). Der Encoder bildet eine Input Sequenz $(x_1, ..., x_n)$ auf eine Sequenz kontinuierlicher Darstellungen $\mathbf{z} = (z_1, ..., z_n)$ ab. x_n ist dabei bspw. ein Wort eines Satzes oder ein Wert innerhalb einer Zeitreihe. Der Decoder verwendet diese interne Darstellung $\mathbf{z}$ und bildet eine Output Sequenz $(y_1, ..., y_n)$. Dabei werden die Elemente

schrittweise, also Element für Element, erstellt. Zu jedem Zeitschritt generiert der Decoder ein Element unter Berücksichtigung von $\mathbf{z}$ und dem im Vorwege generierten Output. Dieses Vorgehen ist auto-regressiv und von dem Encoder unabhängig [62].

Beide Komponenten des Transformers bestehen aus N identischen Schichten (engl. Layers). Der Encoder hat jeweils zwei Sub-Layer. Die erste Layer stellt eine Multi-Head Self-Attention Layer dar. Die zweite Layer ist ein positionsweise vollständig verbundenes Feedforward Netzwerk (engl. positionwise fully connected feed-forward network). Im Decoder ist zusätzlich noch eine dritte Layer integriert, welche den Output des Encoders verarbeitet [62]. Eine detaillierte Beschreibung dieser beschriebenen Komponenten folgt im nächsten Abschnitt. Die Komponenten werden entsprechend ihres Vorkommens in der Architektur geordnet, sodass zuerst die Embeddings und das Positional Encoding beschrieben werden (siehe Abb. 2.9, Betrachtung von unten nach oben).

Embeddings. In der Struktur von Encoder und Decoder stellen das Input-Embedding sowie das Output-Embedding die jeweils ersten Schichten dar und sind in Form von Feedforward Netzwerken konzipiert. Diese Embeddings werden dahingehend trainiert, die Input- und Output-Token in Vektordarstellung der Dimension d_{model} zu transformieren. In der Architektur teilen sich beide Embedding-Layers dieselbe Gewichtsmatrix [62].

Positional Encoding. Einer der wichtigsten Aspekte bei der Verarbeitung von sequenziellen Daten ist die Information über die Ordnung der Sequenz. Dies ermöglicht es dem Modell erst, sequenzielle und temporale Abhängigkeiten der Sequenzen zu erlernen [2]. Da die Transformer Architektur auf rekurrente Bestandteile verzichtet und somit nicht automatisch auf vergangene Informationen zurückgreifen kann, muss dem Modell über einen anderen Mechanismus die Information zur relativen und absoluten Position des Tokens mitgegeben werden. Dieser Mechanismus ist in der Transformer Architektur das Positional Encoding (PE). Das PE kann mit unterschiedlichen Methoden umgesetzt werden. In der ursprünglichen Architektur verwenden die Autoren Sinus- und Kosinus-Funktionen, welche die Position des Tokens innerhalb der Sequenz wie folgt berechnen:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \tag{2.14}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \tag{2.15}$$

wobei pos die Position (Zeitstempel) des Tokens, i die Position innerhalb des Embedding-Vektors und d_{model} die Dimension des Modells ist. Das PE wird dem Embedding im

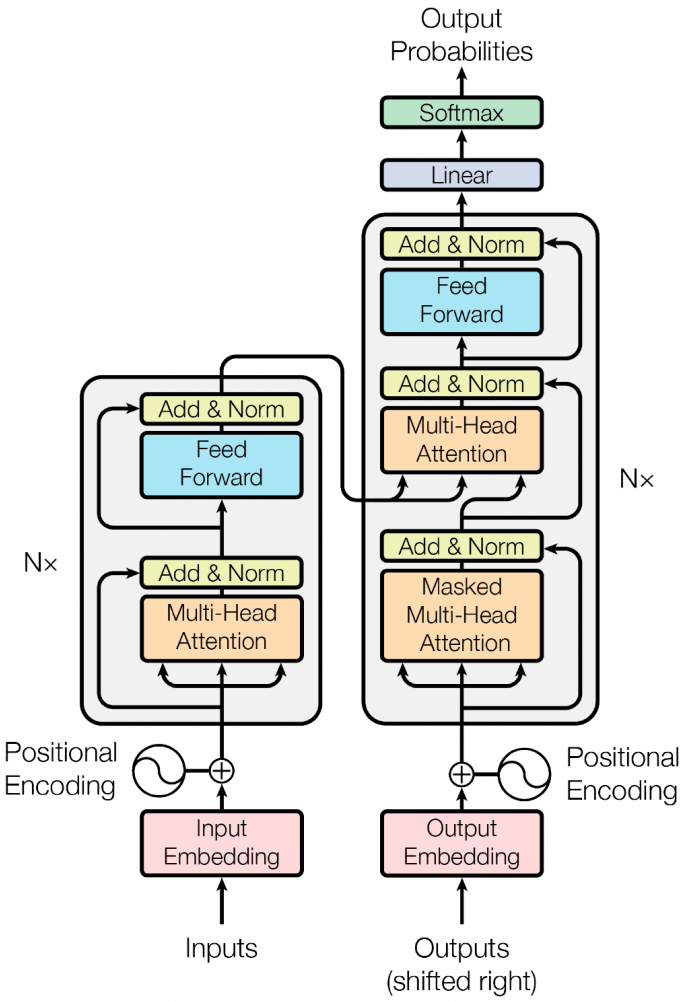

Abbildung 2.9: Architektur eines Transformer-Modells. Der Transformer folgt der Architektur aus einem Encoder (links) sowie einem Decoder (rechts). Die einzelnen Komponenten bieten in Kombination eine leistungsstarke Architektur für sequenzielle Daten [62].

Anschluss an die Berechnung hinzugefügt. Neben dem hier dargestellten Ansatz gibt es weitere Ansätze, die bspw. auch das Erlernen der Positionen (engl. learned PE) umfassen. Diese Methoden erlernen die relativen Positionen über einen Trainingsprozess in Form eines neuronalen Netzes [62].

Multi-Head Attention. Das Konzept der *Multi-Head Attention* beruht auf dem zu Beginn von Kapitel 2.4.1 vorgestellten Attention Mechanism. In [62] wird der Attention Mechanism als ein Mapping einer Query und einem Key-Value Paar zu einem Output beschrieben, bei dem alle Bestandteile Vektoren darstellen.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(QK^T\right)V, \tag{2.16}$$

mit Q als Query, K als Keys und V als Values. Diese Abstraktion des Mechanismus wird im Transformer weiterentwickelt und in das Multi-Head Attention Modul als Layer des Netzes integriert. Für die Integration werden Änderungen an dem klassischen Attention Mechanismus vorgenommen. Zum einen wird das Skalarprodukt der Keys K und Queries Q zur Berechnung der Attention Scores anstelle eines Feedforward Neural Networks (FFN) mit einem Hidden State verwendet. Zum anderen wird das Ergebnis aus $\left(QK^T\right)$ um $\frac{1}{\sqrt{d_{\text{encoding}}}}$ skaliert. Dieses Vorgehen beschreiben die Autoren in [62] als *Scaled Dot-Product Attention* und löst das Problem sehr kleiner Gradienten bei großen Dimensionen des Netzes. Innerhalb der Transformer Architektur werden anstelle der Berechnung eines einzelnen Attention Scores mehrere so genannter Heads verwendet. Diese Multi-Head Attention Module führen die Attention-Berechnung mehrfach mit unterschiedlichen und unabhängigen Gewichtsmatrizen durch. Dadurch wird es dem Modell ermöglicht, unterschiedliche Aspekte der Informationen zu erfassen. Die verschiedenen Attention Scores werden anschließend zusammengeführt und als Output der Multi-Head Attention Layer an die nächste Schicht weitergegeben [41].

Residual Connection und Layer Normalization. Ein weiterer Bestandteil der Transformer Architektur sind die *Residual Connections* und die *Layer Normalization*. Diese sind in Abb. 2.9 unter der Komponente *Add & Norm* zusammengefasst. Die Residual Connection (Add) ermöglicht es, die Ausgabe einer vorherigen Layer direkt mit der darauffolgenden Layer zu kombinieren. Diese Methode wurde in [26] eingeführt und als Konzept für das Problem von Vanishing Gradients entwickelt. Zudem verbessert diese Technik die Modellperformance im Allgemeinen [67]. Anschließend wird das Ergebnis normalisiert. Die Layer Normalization normalisiert die Aktivierungen über alle Neuronen in

einer Schicht für einen einzelnen Datensatz. Dies führt zu einem stabilen Trainingsprozess und optimiert die Leistungsfähigkeit der Transformer Architektur [62].

Position-wise Feedforward Networks. Am Ende jeder Sub-Schicht des Encoders sowie Decoders befindet sich jeweils ein FFN. Dieses besteht aus zwei linearen Transformationen und der ReLU Aktivierungsfunktion. Die Input- sowie Output-Schicht der jeweiligen FFN haben die Dimension d_{model}. Die innere Dimension des Netzes ist größer, je nach entsprechender Konfiguration. In der ursprünglichen Transformer Architektur besitzen die Netzwerke eine innere Dimension von $d_{ff} = 2048$.

Transformer Modelle zeigen aufgrund ihrer Architektur überlegene Fähigkeiten gegenüber bisherigen neuronalen Netzarchitekturen und bieten vor allem in der Verarbeitung von langen Sequenzen, im Besonderen für den NLP-Bereich, neue Möglichkeiten der Modellierung. Dies führte zu unterschiedlichen Adaptionen der Transformer-Architektur, die Grundlage für viele LLMs sind. LLMs sind primär zur Sprachverarbeitung entwickelt worden und können Sequenzen (Sätze) unterschiedlicher Länge verarbeiten. Dies legt eine Übertragung auf Zeitreihendaten nahe und wird in aktuellen Veröffentlichungen angewendet. Daher sollen im folgenden Kapitel die Grundlagen der LLMs gelegt werden. Dazu gehören auch die Architekturen führender Sprachmodelle sowie das Pre-Training und die Adaption vortrainierter Netze mittels effizienter Methoden.

2.4.2 Large Language Models und Pretrained Transformers

Große Sprachmodelle (engl. Large Language Models) repräsentieren eine fortschrittliche Kategorie der künstlichen Intelligenz. Sie zeichnen sich durch ein tiefgreifendes Verständnis menschlicher Sprache aus und sind in der Lage, diverse Aufgaben menschlicher Sprache zu bewältigen [13]. Die Basis für die Leistungsfähigkeit dieser Modelle bilden komplexe Modellstrukturen, eine erhebliche Modellgröße sowie umfangreiche Trainingsdatensätze [64]. Diese Faktoren befähigen LLMs insbesondere dazu, das nächste Wort in einem Satz präzise vorherzusagen. Im Kern der meisten LLMs wird die Transformer Architektur verwendet, die bei entsprechender Skalierung zu den emergenten Fähigkeiten der Modelle beiträgt [13].

Historisch entwickelten sich die ersten Sprachmodelle aus statistischen Modellen und wurden später durch neuronale Netzwerke, wie in den Kapiteln 2.3.2 und 2.3.3 beschrieben, erweitert. Mit dem Fortschritt in der technologischen Entwicklung und dem Zugang zu

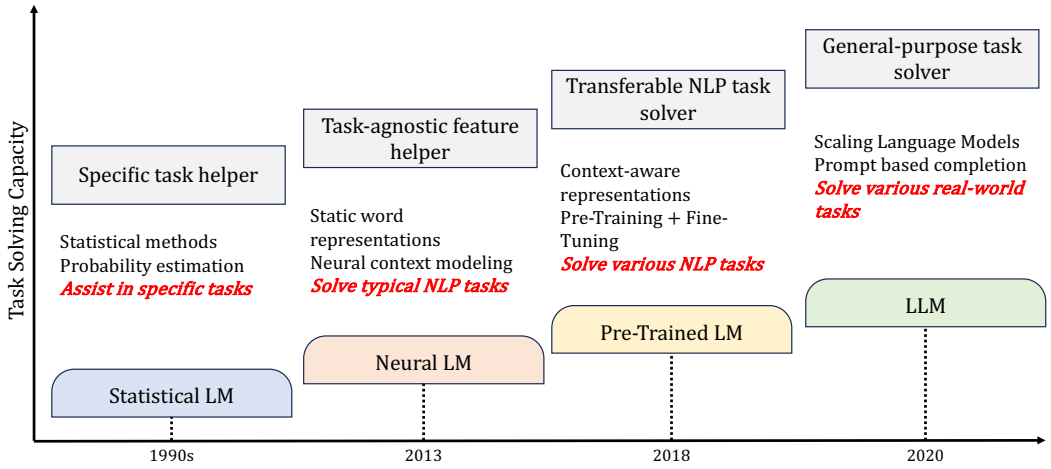

Abbildung 2.10: Evolutionsprozess von Sprachmodellen. Die Entwicklung von Sprachmodellen, betrachtet unter dem Aspekt ihrer Fähigkeit, Aufgaben zu bewältigen, reicht von statistischen Modellen, die bei spezifischen Aufgaben unterstützen, bis hin zu LLMs, die in der Lage sind, diverse Aufgaben zu lösen [72].

umfangreichen Rechenressourcen entstanden die sogenannten vortrainierten Sprachmodelle (engl. Pretrained Language Model (PLM)) [64]. Diese PLMs werden typischerweise auf einem großen Datenkorpus trainiert und anschließend für spezifische Aufgaben angepasst. Beispiele solcher Modelle sind ELMo, BERT und GPT-2. Nach dem Fine-Tuning sind diese Modelle in der Lage, bestimmte Aufgaben mit einem hohen Grad an Spezialisierung zu lösen [72].

Das grundlegende Prinzip des Pre-Trainings mit anschließendem Fine-Tuning der PLMs wird ebenfalls auf die LLMs angewendet. Diese unterscheiden sich von vorherigen Sprachmodellen allerdings in den Fähigkeiten. LLMs sind auch ohne ein spezifisches Fine-Tuning für eine bestimmte Aufgabe in der Lage, komplexe Problemstellungen zu lösen. Dies ist vor allem in der Modellgröße begründet. Spitzenmodelle, wie GPT-4 [47], Llama2 [60] oder Claude, können ohne spezifisches Training die meisten Aufgaben natürlicher Sprachverarbeitung und andere Aufgaben meistern [72]. Die Interaktion mit einem LLM findet an dieser Stelle über Prompting statt. Prompting ist die Beschreibung von Anweisungen in natürlicher Sprache, welche von einem LLM folglich der Anweisung prozessiert wird [13]. Abb. 2.10 fasst die Entwicklung der Sprachmodelle zusammen.

Mit zunehmender Größe beginnen Sprachmodelle, sogenannte emergente Fähigkeiten zu entwickeln, die besonders bei hochentwickelten Modellen erkennbar sind und sich auf

folgende Weise ausdrücken. Einerseits ermöglicht das *In-Context Learning* den LLMs, direkt aus dem Kontext eines vorgegebenen Textes zu lernen und entsprechend zu reagieren. Hierbei ist das Modell in der Lage, die Strukturen und den Kontext des Prompts zu analysieren und darauf basierend eine angemessene Antwort zu formulieren [13]. Andererseits beinhaltet das *Instruction Following* oder auch *Instruction Tuning*, das Fine-Tuning der Modelle mit einem Datensatz, der Instruktionen und Vorgehensmuster zum Lösen von Aufgaben umfasst. Nach diesem Training zeigen die Modelle abstrahierende Eigenschaften, die es ihnen erlauben, auch bisher unbekannte Instruktionen zu bearbeiten [72]. Zusätzlich ist ein LLM durch entsprechende Anpassungen zur Durchführung von schrittweisem logischen Denken (*Step-by-Step Reasoning*) befähigt. Diese Fähigkeit ist bei sehr großen Modelle inhärent vorhanden, während kleinere Modelle sie durch gezieltes Fine-Tuning oder mittels Prompting erlernen müssen. Wenn das Modell dann in einem Prompt ein schrittweises Vorgehen vollziehen muss, ist es in der Lage, komplexe *Chain-of-Thought* Strategien anzuwenden [72].

Nach der Darstellung der historischen Entwicklung von Sprachmodellen und der Beschreibung der zentralen Eigenschaften folgen nun die klassischen Architekturen der Modelle. Anschließend werden Techniken zum Training und Fine-Tuning bzw. Adaption der Sprachmodelle dargestellt, um Grundlagen zur Anpassung auf die Datenimputation zu schaffen.

LLM Architekturen

Die Transformer-Architektur ist die Standardarchitektur aller LLMs, da die Architektur Parallelisierbarkeit erlaubt und damit das Training sehr großer Modelle ermöglicht. Zudem ist die Architektur skalierbar, sodass kein allgemeines Limit der Modellgröße vorhanden ist [72]. Es sind drei generelle Architektur-Muster in Adaption an die Transformer Architektur entstanden [65].

Causal Decoder. Die *Causal Decoder* Architektur, eine *Decoder-Only* Variante, verzichtet auf einen separaten Encoder für die Verarbeitung der Eingabesequenz. Dies hat zur Folge, dass alle Tokens in der gleichen Weise durch den Decoder verarbeitet werden (siehe Abb. 2.11 links). Darüber hinaus hat der Decoder durch die Integration einer unidirektionalen Attention Mask nur Zugriff auf das letzte sowie aktuelle Token und kann somit als klassisches Modell zur Vorhersage des nächsten Tokens bezeichnet werden [65].

Die GPT-Modelle von OpenAI basieren bspw. auf dieser Architektur und belegen die Effektivität dieses Ansatzes. Der *Causal Decoder* wird zudem noch von zahlreichen weiteren LLMs adaptiert und ist aufgrund seiner einfachen Implementierung der aktuell führende Ansatz [72].

Non-Causal Decoder. Der *Non-Causal Decoder* verändert im Vergleich zu dem *Causal Decoder* nur die Maskierung des Attention Mechanismus. Diese wird je nach individueller Anpassung erweitert, sodass bidirektionale Aufmerksamkeit möglich ist. In Abb. 2.11 zeigt die mittlere Abbildung, dass der Decoder nicht nur auf vergangene, sondern je nach Anpassung auch auf nachfolgende Tokens Zugriff hat. Dies erlaubt dem Modell, auf eine umfassendere Informationsbasis innerhalb der Sequenz zuzugreifen, was zu reichhaltigeren und kontextuell vollständigeren Repräsentationen der Eingabedaten führt [72]. Trotz der Potenziale dieser Architektur wird sie, ähnlich wie die *Encoder-Decoder* Architektur, nur von wenigen Modelle umgesetzt. Bekannte Beispiele sind GLM-130B [69] und U-PaLM [59].

Encoder-Decoder. Diese Architektur basiert auf der klassischen Transformer Architektur und verwendet eine Encoder-Decoder Struktur bestehend aus zwei Transformer Blöcken. Der Encoder verarbeitet die Input-Sequenz durch Multi-Head Attention, wodurch eine interne Repräsentation der Sequenz entsteht. Der Decoder greift auf diese Repräsentation zurück und erzeugt auto-regressiv die Ausgabesequenz [72]. Innerhalb des Decoders kommen Cross-Attention-Schichten zum Einsatz, die es ermöglichen, die vom Encoder generierte Ausgabe in die Erzeugung des nächsten Tokens mit einzubeziehen. Auf zukünftige Token kann der Decoder nicht zugreifen, da diese maskiert werden (siehe Abb. 2.11 rechts) [65]. Aktuell setzen nur wenige relevante LLMs auf diese Architektur.

Attention Ansätze

Neben der allgemeinen Architektur unterscheiden sich LLMs darüber hinaus in der Wahl des Aufmerksamkeitsmechanismus. Die Implementierung nach der klassischen Transformer Architektur (*Full Attention, Multi-Head Attention*) [62] wird für unterschiedliche Modelle weiterentwickelt und entsprechend der Anforderungen angepasst [72].

Die *Sparse Attention* bildet eine Weiterentwicklung der *Full Attention* und zielt darauf ab, den Berechnungsaufwand der Attention zu reduzieren, welche vor allem bei langen Sequenzen sehr rechenintensiv ist. Anstatt die ganze Sequenz zu berücksichtigen, fokussiert sich dieser erweiterte Mechanismus auf eine Teilmenge der ursprünglichen Sequenz

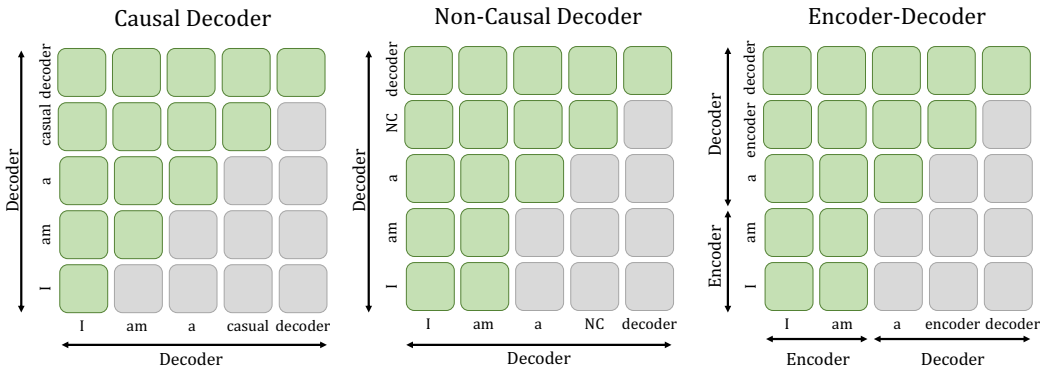

Abbildung 2.11: Attention Patterns in den drei typischen LLM Architekturen. Auf graue Tokens kann keine Aufmerksamkeit gelegt werden, grüne Tokens dagegen werden berücksichtigt. In einem *Causal Decoder* richtet jedes Token seine Aufmerksamkeit nur auf die vorherigen Tokens. Sowohl im *Non-Causal Decoder* als auch im *Encoder-Decoder* ist die Aufmerksamkeit in beide Richtungen unter gewissen Bedingung erlaubt. Diese Bedingung ist beim *Encoder-Decoder* bereits im Encoder-Teil integriert [65] und wird bei den anderen Modellen je nach Zielsetzung angepasst.

und verringert somit die Länge der Sequenz [68]. Dies ist darin begründet, dass der Einfluss von weit entfernten Tokens im Vergleich zu direkt folgenden Tokens minimal wird [72].

Multi-Query Attention bezeichnet eine Variante der Aufmerksamkeit, bei der verschiedene Heads dieselben Matrizen für Keys und Values teilen. Dies ermöglicht eine schnellere Inferenz bei nur geringem Qualitätsverlust der Modelle [55]. Eine Abwandlung dieses Vorgehens ist *Grouped-Query Attention (GQA)*. GQA stellt einen Kompromiss zwischen Multi-Query Attention und der klassischen Multi-Head Attention dar, in dem die Heads in Gruppen eingeteilt werden und innerhalb dieser Gruppen dieselben Matrizen genutzt werden [4]. Diese Methode wird bspw. in Llama2 verwendet [72]. Es gibt darüber hinaus weitere Anpassungen des klassischen Aufmerksamkeitsmechanismus, die an dieser Stelle nicht dargelegt werden sollen.

Pre-Training Tasks

Das Pre-Training oder auch Vor-Trainieren von LLMs ist ein aufwändiger Prozess. Neben der Wahl der richtigen Architektur ist das Vorhandensein einer ausreichend großen Datenmenge unabdingbar, damit die komplexen Muster der natürlichen Sprache erlernt

werden können. Die Daten müssen gesammelt und entsprechend der Anforderungen aufbereitet werden. Anschließend ist es möglich ein Modell zu trainieren, dass über den Trainingsprozess die Eigenschaften natürlicher Sprache erlernt [72]. Für den Prozess des Pre-Trainings gibt es unterschiedliche Tasks.

Language Modeling. Eine Art des Pre-Trainings ist das *Language Modeling*. Diese Methode wird primär bei Decoder-Only Architekturen verwendet. Das Trainingsziel ist die Vorhersage des nächsten Tokens einer Sequenz x_i basierend auf den vorangegangenen Tokens $x_{<i}$. Gegeben ist dabei eine Sequenz von Token $\mathbf{x} = x_1, ..., x_n$. Diese Art des Trainings hilft dem Modell, Texte zu generieren, die syntaktisch und thematisch schlüssig sind. Verallgemeinert ist das Ziel die Maximierung folgender Log-Likelihood-Funktion:

$$\mathcal{L}_{LM}(\mathbf{x}) = \sum_{i=1}^{n} \log P(x_i|\mathbf{x}_{<i}) \tag{2.17}$$

Denoising Autoencoding. Neben der Task des *Language Modeling* wird noch das *Denoising Autoencoding (DAE)* angewendet. Bei diesem Verfahren wird der Input $\mathbf{x}_{\setminus\tilde{\mathbf{x}}}$ mit zufällig ersetzten Abschnitten verfälscht. DAE zielt darauf ab, das Modell robuster zu machen und ein besseres Verständnis der Struktur der Sprache zu erlernen. Das LLM wird darauf trainiert die verfälschten Token wie folgt zu ersetzen:

$$\mathcal{L}_{DAE}(\mathbf{x}) = \log P(\tilde{\mathbf{x}}|\mathbf{x}_{\setminus\tilde{\mathbf{x}}}) \tag{2.18}$$

Anwendung findet dieses Verfahren im Vergleich zum *Language Modeling* allerdings seltener [72]. Nach dem Pre-Training der Modelle werden diese einem Fine-Tuning unterzogen um spezifischere Aufgaben bewältigen zu können und andere Fähigkeit zu erlernen. Auch hier gibt es mehrere Verfahren, deren Ziel die Verbesserung der Modelle sind.

Fine-Tuning und Adaptation

Die Methoden des Fine-Tunings für LLMs sind je nach Zielsetzung unterschiedlich und bieten verschiedene Möglichkeiten. Zu den Adaptionsmethoden zählen *Instruction Tuning*, *Alignment Tuning*, *Parameter-Efficient Fine-Tuning* und *Memory-Efficient Model Adaptation*. Die einzelnen Methoden werden im Folgenden kurz vorgestellt. Im Mittelpunkt der Darstellung stehen die Methoden der Parameter-Efficient Model Adaptation,

da diese Methoden bzgl. der Adaption auf andere Domänen und Anwendungsbereiche relevant sind.

Instruction Tuning. Die Fähigkeiten von LLMs können mit Hilfe von Instruktionen verbessert werden [44]. Die LLMs lernen dabei Instruktionen zu abstrahieren und dem allgemeinen Muster dieser Instruktionen zu folgen. Zur Durchführung des *Instruction Tunings* wird das Modell mit formatierten Instruktionen in natürlicher Sprache trainiert. Diese Art des Fine-Tunings verlangt die Überführung der Daten in ein Format der Instruktionen [44]. Das *Instruction Training* führt dazu, dass LLMs Lösungsmuster erkennen und so ungesehene Aufgaben selbstständig lösen können [72].

Alignment Tuning. LLMs zeigen nach dem Pre-Training häufig unerwartetes Verhalten. Diese Verhaltensweisen drücken sich bspw. in der Produktion von Falschinformationen (Halluzination), dem Verfolgen ungenauer Ziele und der Ausgabe von schädlichen Informationen aus. Das Ziel des *Alignment Tunings* ist die Anpassung des Modells an menschliche Verhaltensweisen und allgemeine Wertevorstellungen [6]. Dabei soll das LLM hilfreiche, ehrliche und wahre Ausgaben generieren und zudem keine offensiven oder diskriminierenden Inhalte produzieren [6]. Es konnte gezeigt werden, dass diese Anpassung mit einer Verringerung der Leistungsfähigkeit einhergeht, die sogenannte *Alignment Tax* [72].

Die beiden dargestellten Fine-Tuning Ansätze beschreiben Methoden zur Anpassung eines LLMs in Bezug auf ein spezifisches Ziel. Die folgenden beiden Methoden zielen darauf ab, das Training der sehr großen Modelle überhaupt möglich zu machen, da neben dem Pre-Training auch der Fine-Tuning Prozess sehr rechenintensiv ist und hohe Ressourcenanforderungen mit sich bringt [72].

Parameter-Efficient Fine-Tuning. Die Methoden des *Parameter-Efficient Fine-Tuning (PEFT)* reduzieren die trainierbaren Modellparameter bei maximal möglicher Erhaltung der Modellperformance [72]. Eine Methode des PEFT ist das *Adapter Tuning*. Beim Adapter Tuning werden Adapter in das Modell integriert. Diese Adapter sind kleine neuronale Netze, welche in die Transformer Layer des LLMs eingefügt werden und während des Fine-Tunings optimiert werden. Da die ursprünglichen Parameter bei diesem Verfahren eingefroren werden, reduziert diese Methode die Anzahl von anpassbaren Parametern signifikant [28]. Eine weiterer Ansatz ist das *Prefix Tuning* [39]. Dabei werden jeder Transformer Layer trainierbare Vektoren (Prefix-Vektoren) vorangestellt. Diese Vektoren sind aufgabenspezifisch und werden während des Fine-Tunings mittels einer Multilayer-Perceptron Methode optimiert [39].

Darüber hinaus findet beim Fine-Tuning der Ansatz der *Low-Rank Adaptation (LoRA)* Anwendung. Auch hier ist es das Ziel, nur einen Teil des Netzes anzupassen, sodass eine geringe Anzahl an Parametern optimiert werden muss [29]. Bei einem LoRA wird das Fine-Tuning wie folgt optimiert. Der generelle Prozess der Optimierung einer Parameter Matrix $\mathbf{W}$ kann als $\mathbf{W} \leftarrow W + \Delta W$ beschrieben werden. D.h., dass die gesamte Matrix $\mathbf{W}$ durch $W + \Delta W$ ersetzt wird. LoRA macht die Matrix $W \in \mathbb{R}^{m \times n}$ allerdings unveränderlich und nähert die Update-Matrix ΔW durch eine Dekomposition $\Delta W = A \cdot B^T$, bei der $A \in \mathbb{R}^{m \times k}$, $B \in \mathbb{R}^{n \times k}$ und $k \ll \min(m, n)$ sind, an [29]. Diese Dekomposition der Matrix erzeugt eine vereinfachte Version von ΔW, die weniger Informationen enthält, aber immer noch effektiv für die Anpassung an spezifische Aufgaben ist. Dieses Verfahren ermöglicht es zudem, mehrere LoRAs für unterschiedliche Aufgaben zu trainieren und abzuspeichern. Aufgrund dieser Flexibilität ist das Vorgehen zum de facto Standard beim Fine-Tuning von LLMs geworden [72].

Eine Erweiterung von LoRA ist *Adaptive Low-Rank Adaptation (AdaLora)*. Diese Methode erweitert LoRA um einen adaptiven Prozess, der eine gewichtete Verteilung der trainierbaren Parameter zu wichtigen Gewichtsmatrizen vornimmt. Eine Begründung für die Einführung dieses Vorgehen findet sich in der Tatsache, dass das klassische LoRA Verfahren zu jeder Modellkomponente Matrizen gleicher Rangordnung hinzufügt [70]. Es lässt sich allerdings belegen, dass einige Modellkomponenten für den Optimierungserfolg wichtiger als andere Komponenten sind. Daher wird innerhalb von AdaLora die Wichtigkeit der Komponenten bestimmt. Anhand dieser Relevanz wird der Rang der hinzugefügten Matrix festgelegt. Teile mit niedriger Wichtigkeit werden herabgestuft, während bedeutende Teile für das Fine-Tuning mit einer ranghöheren Matrix versehen werden [70]. Am Ende des Prozesses haben die unterschiedlichen LoRA Adapter somit nicht alle denselben Rang, sondern abweichende Ränge.

Neben den genannten LoRA-Methoden gibt es weitere PEFT-Methoden, die zum Fine-Tuning von LLMs geeignet sind. Eine dieser Methoden wird als $(IA)^3$ bezeichnet und entstammt einem Paper, in dem In-Kontext-Lernen mit dem parameter-effizienten Fine-Tuning von LLMs verglichen wird [43]. $(IA)^3$ fügt den Key und Value Komponenten der Aufmerksamkeitsschicht von Transformer Blöcken sowie den Aktivierungsfunktionen der FFN einen Skalierungsvektor hinzu, der aufgabenspezifisches Fine-Tuning ermöglicht. Dadurch verändert sich die Gleichung 2.16 der Multi-Head Attention aus Kapitel 2.4.1 wie folgt:

$$\text{softmax}\left(\frac{Q(l_k \odot K^T)}{\sqrt{d_k}}\right)(l_v \odot V) \tag{2.19}$$

$l_k \in \mathbb{R}^{d_k},\ l_v \in \mathbb{R}^{d_v}$ sind die zusätzlichen Vektoren während des Optimierungsprozesses. Mit Hilfe dieser Methode erreichen die Autoren vergleichbare Modellperformance bei wenigen Trainingsschritten und sehr parameter-effizienter Anpassung des Modells [43].

Memory-Efficient Model Adaptation. Der letzte zu nennende Teilbereich der Modell-Adaption beschäftigt sich mit der speicher-effizienten Anpassung von LLMs. Diese Art der Adaption ist notwendig, da LLMs wie bereits beschrieben sehr hohe Ressourcenanforderungen bzgl. der Inferenz sowie des Trainings haben. Das gleiche Ziel verfolgend wie die PEFT Methoden, zielt die *Memory-Efficient Model Adaptation* nicht auf die Modellgröße ab, sondern auf die Art und Weise der Speicherung der Modellgewichte und Aktivierungen. Dies wird durch einen Prozess der Quantisierung (Quantization) ermöglicht. Bei der Quantisierung werden Floating Point Numbers auf Integers gemappt und dadurch die Anforderungen an die Speichermenge reduziert [72]. Eine konkrete Methode der speicher-effizienten Anpassung ist QLoRA. Dieser Ansatz macht sich die Funktionsweise von LoRA zunutze und integriert eine Quantisierung des Modells [16]. Durch die Quantisierung des LLMs in 4-bit Normal Float werden die Anforderungen an die Rechenressourcen stark minimiert. Dem entgegen werden die LoRA-Komponenten in voller Präzision optimiert, sodass es nur eine minimale Reduktion der Leistungsfähigkeit hingenommen werden muss. Die Ergebnisse der Autoren zeigen, dass die Resultate vergleichbar mit klassischem Fine-Tuning sind [16].

2.4.3 Imputation mit Large Language Models

Im Anschluss an die Darlegung der grundlegenden Prinzipien der Imputation sowie der Vorstellung von Deep Learning Methoden zur Imputation in den vorangegangenen Kapiteln, wurden ausführliche Erläuterungen zu den Transformer-Modellen und LLMs präsentiert. Dieses Unterkapitel zielt darauf ab, einen Überblick über die allgemeinen Strategien der Imputation mittels LLMs zu geben. Eine detaillierte Diskussion spezifischer Studien und Arbeiten zu diesem Thema ist für das folgende Kapitel vorgesehen.

Grundsätzlich basiert die Anwendung von LLMs zur Imputation von Zeitreihen darauf, dass sowohl Zeitreihendaten als auch Textdokumente Sammlungen von Sequenzen darstellen. Im Gegensatz zu den numerischen Datenpunkten in Zeitreihen sind Textsequenzen als Zeichenketten ausgedrückt [23]. Da Sprachmodelle darauf ausgelegt sind,

komplexe Wahrscheinlichkeitsverteilungen über Sequenzen abzubilden, lässt sich in der Theorie eine Anwendung dieser Modelle auf Zeitreihendaten übertragen [23].

Es gibt grundsätzlich zwei Hauptansätze für die Anwendung von LLMs auf die Zeitreihendomäne, welche nach [35] in die Kategorien multimodale Neuausrichtung (engl. multimodal repurposing) und Schnittstellen basiertes Prompting (engl. api-based prompting) eingeteilt werden können. Die multimodale Neuausrichtung beinhaltete das Aktivieren aufgabenspezifischer Fähigkeiten von LLMs, indem eine Anpassung zwischen den Ziel und den Pre-Trainingsaufgaben erfolgt. Dies kann auch das erneute Fine-Tuning eines LLMs umfassen, verlangt allerdings nicht zwangsläufig ein Fine-Tuning. Im Rahmen der Neuausrichtung sind auch Umstrukturierungen des Sprachmodells durch das Hinzufügen neuer Komponenten möglich, abhängig davon, ob das LLM optimierbar sein soll oder nicht [35]. Der zweite Ansatz kombiniert die Zeitreihendaten mit natürlicher Sprache und lässt den Prompt von einem LLM beantworten. In diesem Rahmen werden dem Modell Informationen zur Domäne, der Zeitreihe an sich und weiteren Details übergeben, sodass dieses in der Lage ist, eine fundierte Annäherung zurückzugeben. Dieses Verfahren ist im Gegensatz zur Neuausrichtung sehr leichtgewichtig und erfordert lediglich die Schnittstelle zu einem Sprachmodell [35].

Die dargestellten Inhalte zeigen, dass LLMs unterschiedliche Architekturen und Konzepte annehmen können. Generell steht die Transformer Architektur im Mittelpunkt und wird entsprechend der Zielsetzung der Modelle angepasst und erweitert. Ist ein Modell konzipiert wird es einem Pre-Training unterzogen. Die zwei unterschiedlichen Kategorien des Pre-Trainings umfassen Language Modeling und Denoising Autoencoding. Nach dem Pre-Training werden die Modelle optimiert. Aufgrund der Modellkomplexität ist die Anwendung von ressourcensparenden Methoden, wie z.B. PEFT-Methoden notwendig. Im folgenden Kapitel werden relevante Arbeiten dargestellt, welche die vorgestellten theoretischen Inhalte auf die Imputation von Zeitreihen anwenden.

3 Verwandte Arbeiten

In diesem Kapitel werden Arbeiten dargestellt, welche die zuvor beschriebenen theoretischen Grundlagen konkret umsetzen. Diese Arbeiten zeigen Ansätze zur Imputation von Zeitreihendaten mittels Transformer-Architekturen oder LLMs und sind für die vorliegende Arbeit von Relevanz. Das Kapitel ist wie folgt aufgebaut. Im Folgenden werden zwei Methoden dargestellt, die auf der Grundlage klassischer Transformer Architekturen konzipiert sind (siehe Kapitel 3.1). Darauf folgend wird eine Arbeit vorgestellt, welche Ausgangspunkt für diese Arbeit ist (siehe Kapitel 3.2.1). Das Paper verwendet ein GPT-2-Modell zur Lösung unterschiedlicher Zeitreihenanalyse-Aufgaben, bspw. auch die Imputation von Zeitreihen. Zuletzt wird eine Arbeit präsentiert, die LLMs zur Vorhersage von Zeitreihendaten verwendet, indem das Modell mit einem LoRA an die entsprechende Aufgabe angepasst wird (siehe Kapitel 3.2.2). Nach der Vorstellung der Transformer basierten Ansätze folgen somit zwei LLM basierte Konzepte, welche die multimodale Neuausrichtung des Modells durch Komponenten-Optimierung verfolgen. Arbeiten, die ausschließlich auf Prompting basieren, finden in dieser Betrachtung keine Berücksichtigung.

3.1 Konzepte basierend auf Transformer-Modellen

3.1.1 SAITS: Self-attention-based Imputation for Time Series

SAITS ist ein Modell zur Imputation von unvollständigen Zeitreihen, welches auf der Transformer Architektur aufbaut. Den Ansatz der Autoren [19] zeichnen zwei Besonderheiten aus, zu denen die Konzeption des Optimierungsprozesses sowie die Anpassungen an der klassischen Transformer-Architektur gehören. Der Optimierungsprozess des SAITS-Modells besteht aus zwei Bestandteilen. Ein *Joint-Optimization Training Approach* der Autoren verbindet die Aufgaben der Imputation sowie die Rekonstruktion von korrumpierten Zeitreihen. Die zwei Methoden der Optimierung heißen zum einen

Masked Imputation Task (MIT) und zum anderen *Observed Reconstruction Task (ORT)*. MIT bezeichnet die Optimierung des Modells anhand der Berechnung eines Loss zwischen künstlich maskierten Werten und deren wahren Beobachtungen. Hierdurch soll das Modell die Aufgabe der Imputation erlernen. ORT dagegen ermöglicht dem Modell die allgemeine Verteilung der Daten zu erlernen. Diese Optimierungsmethode berechnet einen Verlust zwischen den vom Modell selber rekonstruierten und den wahren Zeitreihen. Durch diesen nachgelagerten Ansatz lernt das Modell die Imputation aber auch die Verteilung der Daten. Diese Kombination der Trainingsansätze führt bei SAITS zu einer optimierten Modellperformance [19].

Die zweite Besonderheit von SAITS ist die Anpassung der klassischen Transformer-Architektur zu einer für die Imputation verbesserten Architektur. Die Multi-Head Attention-Blöcke des ursprünglichen Transformers werden durch zwei hintereinander geschaltete *Diagonally-Masked Self-Attention (DMSA)* Komponenten ersetzt. Diese DMSA-Blöcke ermöglichen dem Modell die zeitlichen Abhängigkeiten und die Korrelation der Feature mit nur einer Aufmerksamkeitsberechnung zu abstrahieren. Der zweite DMSA Block nimmt den Output des ersten Blocks entgegen und führt das Training fort [19]. Am Ende der Datenverarbeitung werden die Outputs der einzelnen Blöcke unterschiedlich gewichtet (siehe Abb. 3.1, Weighted Combination Block).

SAITS basiert auf der Transformer Architektur und verwendet einen kombinierten Ansatz (MIT und ORT) zur Optimierung des Modells. Dabei erreicht das Modell eine State-of-the-Art Performance bei der Imputation von Daten.

3.1.2 Filling out the missing gaps: Time Series Imputation with Semi-Supervised Learning

Ein weiterer Ansatz aufbauend auf der Transformer-Architektur ist ST-Impute (Sparse Transformer Imputation) [1]. ST-Impute ist grundsätzlich sehr ähnlich wie SAITS konzipiert und verwendet nur das Encoder Netz der Transformer-Architektur. Der Optimierungsprozess des Modells beruht im Vergleich zu SAITS auf drei spezifischen Tasks. Zu den Aufgaben gehören *Masked Imputation Modeling (MIM)*, *Non-missing Reconstruction Loss (NRL)* sowie ein *Semi-Supervised Downstream Task Loss*. Die beiden ersten Trainingsziele entsprechen den genannten Konzepten aus SAITS. Der dritte darüber hinausgehende Ansatz bezieht den Loss einer nachgelagerten Zeitreihenaufgabe (bspw. Klas-

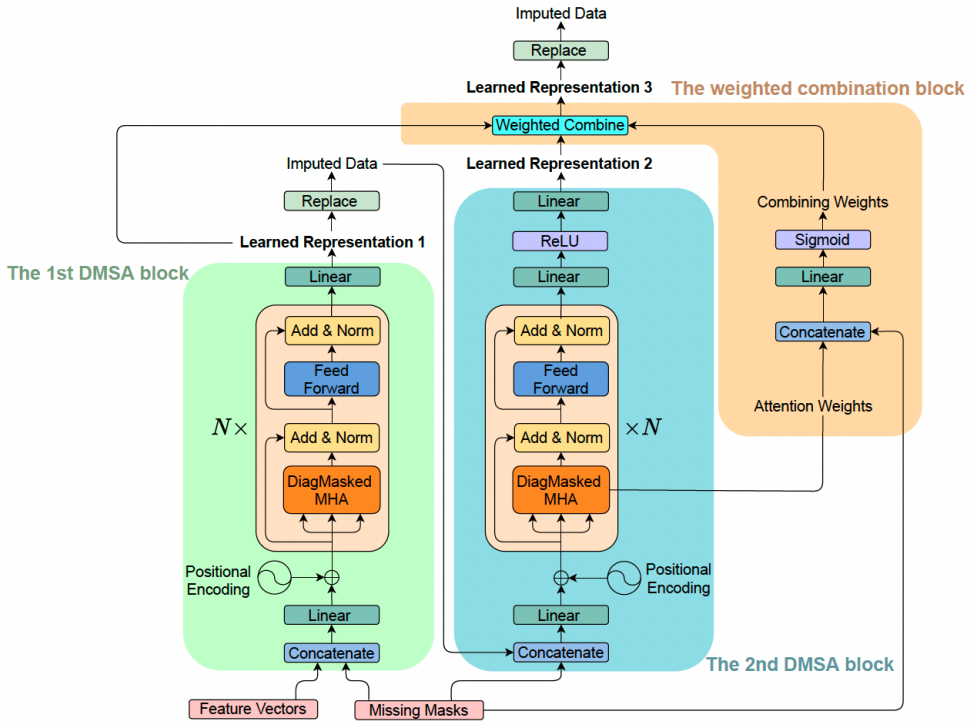

Abbildung 3.1: Architektur von SAITS, vorgestellt in [19]. Abweichend zur klassischen Transformer Architektur verwenden Du et al. DMSA Blöcke und eine gewichtete Kombination der Outputs.

sifikation, Regression) mit ein. Alle drei Werte werden am Ende des Prozesses als Loss für die Anpassung des Netzes verwendet [1].

ST-Impute nimmt wie auch SAITS Änderungen an der klassischen Transformer-Architektur vor und verwendet DMSA. Darüber hinaus nutzt ST-Impute nicht die klassische Self-Attention, sondern Sparse Self-Attention, eine Weiterentwicklung des Aufmerksamkeitsmechanismus. Weitere Änderungen an der Transformer-Architektur werden nicht vorgenommen. Zusammenfassend zeigt sich, dass ST-Impute der Architektur von SAITS ähnelt und auf dem gleichen Grundkonzept aufbaut [1]. Beide Ansätze erreichen eine überdurchschnittliche Modellperformance bei der Imputation von Zeitreihen, werden in den Veröffentlichungen allerdings nicht miteinander verglichen.

3.2 Konzepte basierend auf Sprachmodellen

3.2.1 One Fits All: Power General Time Series Analysis by Pretrained LM

Zhou et al. präsentieren einen Ansatz zur allgemeinen Lösung verschiedener Zeitreihenaufgaben durch vortrainierte Sprachmodelle [74]. Das präsentierte Modell heißt GPT4TS und erreicht vergleichbare Performance zu anderen State-of-the-Art Methoden. Im Kern des konzipierten Modells wird GPT-2 [51], also eines der Vorgängermodelle von GPT-4, verwendet. Das bereits trainierte GPT-2 Modell wird beim Optimierungsprozess nur zum Teil angepasst und die Aufmerksamkeitsschichten sowie FFNs eingefroren, da diese die erlernten Kenntnisse des Sprachmodells enthalten [74]. Die Positional Embeddings und Layer Normalization der GPT-2-Architektur sind Gegenstand des Anpassungsprozesses auf die Zeitreihenaufgaben. Damit ein für NLP-Aufgaben konzipiertes Modell die Input Zeitreihen verarbeiten kann, wird die Input Embedding Layer von GPT4TS darauf trainiert, die Zeitreihen auf die Modelldimension von GPT-2 zu mappen (siehe Abb. 3.2, Input Embedding) [74].

Die Ergebnisse von GPT4TS bei der Imputation auf sechs realen Datensätzen zeigen vergleichbare Ergebnisse mit anderen State-of-the-Art Methoden. Zur Erreichung dieser Ergebnisse reicht bereits das Training von einem Viertel der gesamten GPT-2 Modellgröße. Von den zwölf verfügbaren Schichten werden lediglich drei Schichten trainiert. Insgesamt zeigt der Ansatz, dass ein vortrainiertes Sprachmodell ein guter Ausgangspunkt für die Anpassung auf Zeitreihenaufgaben sein kann [74].

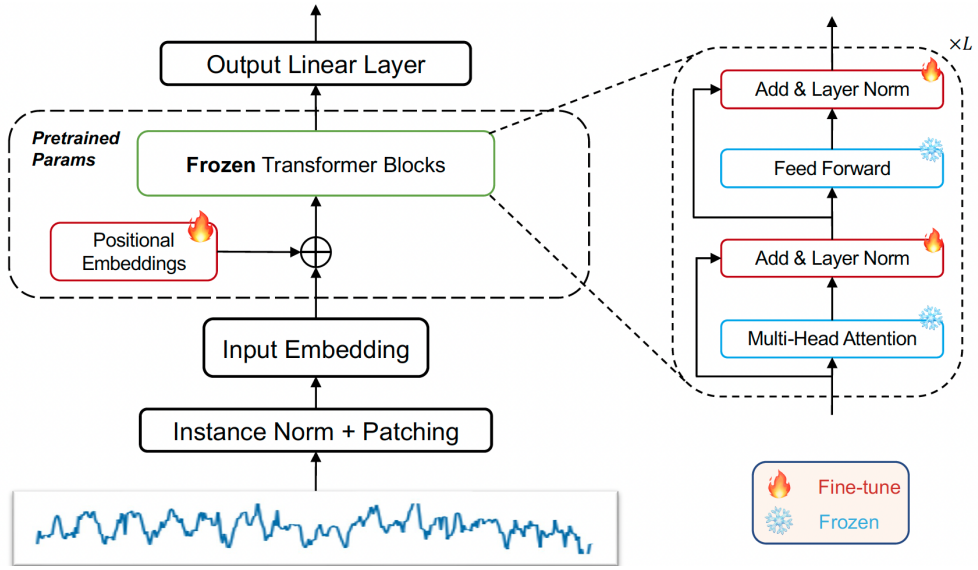

Abbildung 3.2: Architektur von GPT4TS. Der Großteil des vortrainierten Modells (GPT-2) bleibt während des Fine-Tunings unverändert. Nur die Embedding Layer, die Normalisierungsschicht und die Output Schicht werden zur Anpassung auf Zeitreihenaufgaben optimiert [74].

3.2.2 TEMPO: Prompt-based Generative Pre-trained Transformer for Time Series Forecasting

Ein weiteres Modell zur Verarbeitung von Zeitreihendaten ist TEMPO [10], das trotz seiner ausschließlichen Verwendung für Prognosezwecke darstellt, wie Sprachverarbeitungsmodelle für Zeitreihenaufgaben adaptiert werden können. Auch in TEMPO ist GPT-2 das Kernstück des Modells. Während des Fine-Tuning-Prozesses bleiben bestimmte Modellkomponenten unverändert, darunter die FFN-Schichten sowie die Aufmerksamkeitsgewichte. Eine signifikante Anpassung in TEMPO sind die in die Aufmerksamkeitsblöcke integrierten LoRA-Komponenten, die eine verbesserte Anpassung an die Zeitreihen Aufgaben ermöglichen sollen [10].

Über die Anpassung der Architektur hinaus wird eine besondere Vorverarbeitung der Daten vorgenommen. Dazu werden die Zeitreihen in die einzelnen Komponenten Trend, Saisonalität und Residuals aufgeteilt. Das Training von TEMPO findet auf den gleichen Datensätzen wie bei GPT4TS statt. Die Ergebnisse zeigen, dass das Modell die Performance von GPT4TS bei der Vorhersage von Zeitreihen übertreffen kann. Andere Zeitreihenaufgaben werden nicht berücksichtigt [10].

3.3 Diskussion

Die Imputation mit Transformer Modellen oder LLMs in den dargestellten Arbeiten zeigt die möglichen Erfolge der Transformer-Architektur bei der Übertragung auf die Zeitreihendomäne. Die Transformer basierten Modelle SAITS und ST-Impute sind vom Grundaufbau sehr ähnlich und unterscheiden sich lediglich in kleinen Details. ST-Impute ist näher an die ursprüngliche Transformer-Architektur angelehnt und verwendet nur den Encoder des Transformers. SAITS dagegen integriert beide Komponenten (Encoder und Decoder) und führt eine zusätzliche Gewichtung ein. Der Optimierungsprozess beider Modelle ist zielgerichtet auf die Imputation der Daten. Dies lässt sich in der Einführung der zwei präsentierten Loss-Values (z.B. MIT & ORT bei SAITS) begründen und führt dazu, dass das Modell neben der Aufgabe der Imputation auch die Wiederherstellung der eigentlichen Zeitreihe lernt. Die Kombination der beiden Ansätze führt am Ende zu einer verbesserten Modellperformance [19]. Durch ihre Fokussierung auf die Imputationsaufgabe verlangen diese beiden Modelle nicht, bei sämtlichen Aufgabenstellungen der Zeitreihenanalyse zu überzeugen. Im Vergleich zu den Architekturen basierend auf LLMs sind SAITS und ST-Impute somit spezialisierte Modelle.

Die Modelle TEMPO und GPT4TS verwenden im Zentrum ihrer Architektur das Sprachmodell GPT-2 und passen zum einen dieses Modell an, optimieren zum anderen aber auch die Komponenten um GPT-2 herum. Die Aufmerksamkeitsgewichte des LLMs werden bei GPT4TS nicht angepasst. Lediglich die Add & LN-Schicht des Netzes sind Gegenstand des Optimierungsprozesses. TEMPO dagegen passt auch die Blöcke mit den trainierten Gewichten für die Attention an. Dies erreichen die Autoren mit der Einführung von LoRAs. Das Konstrukt um das LLM herum unterscheidet sich bei beiden Ansätzen kaum und soll daher nicht weiter betrachtet werden.

Die untersuchten Veröffentlichungen verdeutlichen, dass sowohl Transformer-Modelle als auch LLMs erfolgreich für die Imputation eingesetzt werden können. Ein direkter Vergleich zwischen den Modellen, insbesondere zwischen SAITS und den LLM basierten Ansätzen, wurde allerdings nicht durchgeführt. Hier ist von Interesse, wie die spezialisierten Modelle im Vergleich zu den LLMs abschneiden. Ferner bleibt die Frage offen, ob größere LLMs, wie z.B. Llama2 [60] oder Phi-2 [24], unter gleichen Bedingungen vergleichbare oder sogar überlegene Ergebnisse erzielen würden. Darüber hinaus nehmen die Autoren in ihrem Setup an, dass das Fine-Tuning unterschiedlicher Komponenten auch verschiedene Resultate in der Imputation der Daten liefern würde [74]. Eine Untersuchung dieser Annahme erfolgt allerdings nicht.

Die hier diskutierten Punkte liefern Ansätze zur Überprüfung der vorherigen Arbeiten. Zudem unterstützen die offenen Diskussionspunkte die Wahl der aufgestellten Forschungsfragen dieser Arbeit. Im nächsten Kapitel folgt daher der methodische Aufbau und die damit einhergehenden Experimente zur Beantwortung der relevanten Punkte.

4 Methodik

Die hier präsentierte Architektur zur Imputation von Zeitreihendaten mittels LLMs stellt einen zentralen Bestandteil der vorliegenden Arbeit dar und ist das Resultat verschiedener Adaptionen vorheriger Forschungen. Die Architektur, welche Teile der LLM-Komponenten modifiziert und weitere Elemente um das Kernmodell herum trainiert, führt zu einem Ansatz, der speziell auf die Imputation von Zeitreihendaten zugeschnitten ist. Insgesamt ist die konzeptionierte Modellstruktur nahe an der von Zhou et al. [74] präsentierten GPT4TS-Architektur, da diese eine gute Ausgangslage für die modulare Aufbauweise bietet und eine Integration unterschiedlicher neuer Komponenten ermöglicht. Im Fokus des Entwurfs stehen im Allgemeinen unterschiedliche LLMs. Die unterschiedlichen LLMs werden im folgenden Kapitel vorgestellt. Es ist bereits zu erwähnen, dass ein Teil der betrachteten Sprachmodelle in der Parameteranzahl signifikant von dem GPT-2-Modell aus GPT4TS abweichen. Zur Abwandlung der Architektur werden Konzepte aus [10] und [13] herangezogen. Es ergibt sich folglich das Zielbild in Abb. 4.1. Während des Optimierungsprozesses sind bestimmte Komponenten der Architektur, speziell ein Teil der Komponenten des Pretrained LLM, fixiert und erfahren keine weiteren Anpassungen. Die Module außerhalb des gestrichelten Bereiches sind während des Fine-Tunings alle optimierbar, da sie entscheidend für das Mapping der Zeitreihendaten in ein vom LLM verarbeitbares Format sind. Nachfolgend werden die einzelnen Komponenten und ihre Funktionen innerhalb dieses Prozesses erläutert. Zudem werden abschließend die konkreten Veränderungen über die GPT4TS Architektur hinaus beschrieben. Diese Anpassungen werden des Weiteren im Bezug zu den Forschungsfragen kommentiert.

Pretrained LLM und PEFT-Methoden. Das zentrale Kernstück der diskutierten Architektur bildet, wie zuvor erläutert, ein vortrainiertes LLM. Die Konstruktion der Architektur ermöglicht einen flexiblen Austausch dieses zentralen Modells, wodurch jegliches Modell in den Versuchen des fünften Kapitels integriert werden kann. Im Unterschied zu GPT4TS, führt die vorliegende Arbeit eine Anpassung der Attention-Layer innerhalb des LLMs durch. Mit Hilfe der LoRA-Technik kann das Modell effizient und mit minimalem Leistungsverlust bei reduziertem Ressourcenbedarf modifiziert werden, wobei die

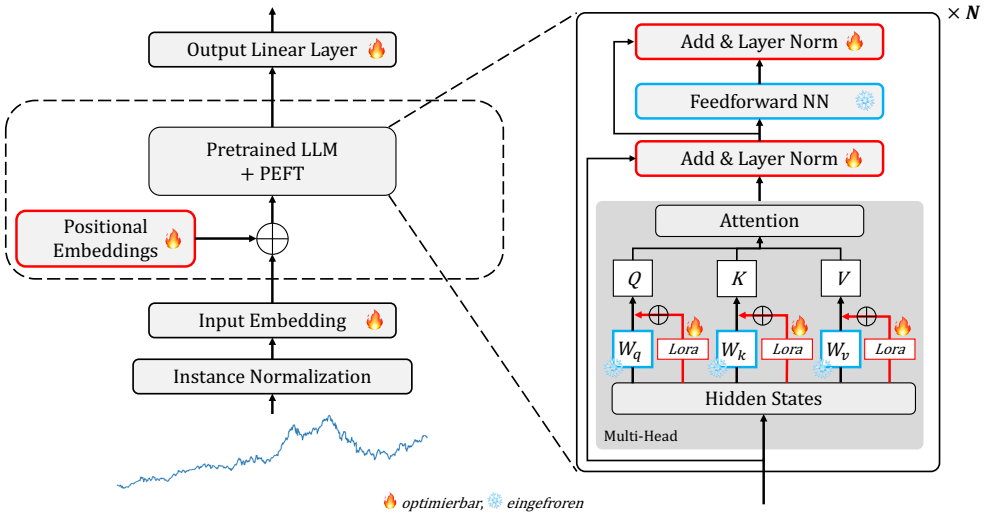

Abbildung 4.1: Adaptiertes Framework zum Fine-Tuning von LLMs zur Imputation. Diese Darstellung zeigt den in dieser Arbeit verwendeten Ablauf des Fine-Tunings eines LLMs zur Anpassung auf Zeitreihendaten. Die Komponenten im Bereich Pretrained LLM sind nur z. T. optimierbar. Die anderen Komponenten außerhalb dieses Bereichs werden während der Optimierung angepasst, um eine Anpassung auf die Imputation zu ermöglichen.

ursprünglichen Gewichte der Attention-Layer unverändert bleiben. Zusätzlich bleibt das FFN des LLMs während des Fine-Tunings fixiert. Die Komponente Add & Layer Norm des LLMs ist wie das Positional Embedding trainierbar und nicht eingefroren, da eine Anpassung dieser Komponenten zur Optimierung auf nachgelagerte Aufgaben Standard ist und somit Teil des Fine-Tuning-Prozesses sein muss [74].

Input Embedding. Das Input Embedding sorgt dafür, dass die Zeitreihe transformiert wird und von dem Positional Embedding des LLMs verarbeitet werden kann. Dies ermöglicht erst, dass die unvollständige Zeitreihe von dem Modell imputiert werden kann. Hier wird das Vorgehen aus [74] adaptiert.

Instance Normalization. Die Normalisierung ist eine zentrale Komponente der GPT4TS Architektur und wird auch hier übernommen. Die Zeitreihendaten werden pro Instanz über Mittelwert und Varianz normiert. Das Verfahren nach [37] ist im Bereich der Zeitreihenvorhersage ein gängiges Vorgehen und unterstützt den Lernprozess.

Output Linear Layer. Am Ende des Prozesses gibt das LLM die Zeitreihen in einem Format zurück, dass nicht dem Ursprungszustand der Zeitreihe entspricht. Diesen Vor-

gang übernimmt eine finale lineare Output Layer, welche die Ausgabe des Modells von der Modelldimension d_{model} wieder in die Form der Input-Daten transformiert.

Die in dieser Arbeit beschriebene Architektur stellt eine Erweiterung des Ansatzes von Zhou et al. [74] dar und integriert die Anwendung von PEFT-Methoden. Diese erweiterte Funktionalität findet sich in der Darstellung des Attention-Blocks in Abb. 4.1. Die modulare Konzeption der Architektur ermöglicht es zudem, die in Kapitel 1.2 formulierten Forschungsfragen zielgerichtet zu adressieren. Durch den möglichen Austausch der zentralen Komponente können verschiedene Sprachmodelle angepasst werden, während die übrige Struktur der Architektur unverändert bleibt. Dies unterstützt zudem die Vergleichbarkeit der entstehenden Ergebnisse, da alle Sprachmodelle unter einem einheitlichen Architektur-Setup optimiert werden.

Dieses Framework ist dabei ausschließlich für die LLM basierten Ansätze konzipiert. Andere State-of-the-Art Methoden, wie z.B. SAITS, verwenden folglich nicht diese Architektur. Darüber hinaus ist das adaptierte Framework dazu geeignet, einen Vergleich des Fine-Tunings unterschiedlicher Modellkomponenten zu ermöglichen. Innerhalb dieser Architektur können einzelne Komponenten in den Optimierungsprozess einbezogen und diverse Kombinationen dieser Komponenten effizient implementiert werden. Zur Untersuchung der letzten Forschungsfrage, dem Vergleich der PEFT-Methoden, kann der LoRA-Adapter in Abb. 4.1 durch die geeigneten Ansätze ausgetauscht werden. Das modulare Konzept der Architektur unterstützt somit die zielgerichtete Erforschung der aufgestellten Fragen und gewährleistet die Vergleichbarkeit der Ergebnisse, indem klar definiert ist, welche Komponenten innerhalb eines Experiments modifiziert werden.

Im Anschluss an die Vorstellung der Methodik folgt die Durchführung der notwendigen Experimente.

5 Experimente

Das Kapitel der Experimente umfasst die einzelnen Teile des Versuchsaufbaus, der Ergebnisdarstellung sowie einer Diskussion der Ergebnisse. Die Darstellung sowie Diskussion der Ergebnisse sind dabei nach den einzelnen Forschungsfragen strukturiert.

5.1 Versuchsaufbau

Der Versuchsaufbau zur Durchführung der Experimente stellt anfangs die ausgewählten Datensätze dar. Anschließend folgen die unterschiedlichen Baseline Modelle. Im Rahmen dieser Darstellung sind die ausgewählten Sprachmodelle von zentraler Bedeutung. Das Kapitel schließt mit den Details der Implementierung ab. Hier werden die relevanten Optimierungsparameter dargelegt.

5.1.1 Datensätze

Die Experimente und das Fine-Tuning der Modelle findet auf sechs Datensätzen statt, die in vorherigen Arbeiten zu anderen Zeitreihen-Modellen, wie z.B. TimesNet [66], Informer [73] oder GPT4TS [74], Anwendung finden. Im Folgenden werden diese Datensätze vorgestellt.

Electricity Transformer Temperature (ETT)[1]. Die ETT Daten entstammen der Arbeit von Zhou et al. [73] und dokumentieren Leistungsindikatoren von zwei Elektrizitätstransformatoren die in unterschiedlichen Gebieten in China stationiert sind. Über einen Zeitraum von zwei Jahren wurden die Indikatoren jeweils im 15 Minuten (ETTm1, ETTm2) sowie im stündlichen (ETTh1, ETTh2) Intervall gemessen. Jeder Datenpunkt

[1] Datensatz verfügbar unter: https://github.com/zhouhaoyi/ETDataset

besteht dabei aus sechs Features mit einer Zielvariablen, der Öltemperatur. Die Trainingsdaten für das Experiment umfassen zwölf Monate, die Validierungs- und Testdaten jeweils vier Monate der ETT-Daten [73].

Electricity Consuming Load (ECL)[2]. Der Datensatz ECL umfasst ca. 26.000 Datenreihen mit jeweils 321 Spalten. Die Spalten sind unterschiedliche Klienten, deren Verbrauch in kWh alle 15 Minuten für jeden Tag innerhalb des Erhebungszeitraumes erfasst wurde. Somit ergeben sich für einen Tag 96 unterschiedliche Datensätze [73].

Weather[3]. Dieser Datensatz enthält gemessene Klimafaktoren für einen spezifischen Ort. Die Messungen werden alle zehn Minuten durchgeführt und über einen Zeitraum von einem Jahr erhoben. Dies führt dazu, dass ca. 52.500 Messdaten vorliegen. Der Datensatz wird für den Versuch in 70% Trainings-, 20% Test- und 10% Validierungsdaten eingeteilt [73].

Infolgedessen ergeben sich sechs Datensätze mit unterschiedlichen Dimensionen aus der realen Welt, die alle auf eine Sequenzlänge von 96 Einträgen zugeschnitten werden können. Tab. 5.1 fasst die Datensätze in tabellarischer Form zusammen. Die Größe des Datensatzes ist eingeteilt in Trainings-, Validierungs- und Testdaten. Die konkrete Aufteilung unterscheidet sich zwischen den Datensätzen und ist Wu et al. [66] entnommen.

Tabelle 5.1: Zeitreihendatensätze für die Experimente

Datensatz	Dimension	Datensatz Größe	Information
ETTTm1, ETTm2	7	(34.465, 11.521, 11.521)	Elektrische Spannung
ETTTh1, ETTh2	7	(8.545, 2.881, 2.881)	Elektrische Spannung
Electricity	321	(18.317, 2.633, 5.261)	Stromverbrauch in kWh
Weather	21	(36.792, 5.271, 10.540)	Klimadaten

5.1.2 Baseline Modelle

Zur validen Evaluation der in dieser Arbeit adaptierten Modelle wird ein Vergleich mit führenden State-of-the-Art-Modellen durchgeführt, die teilweise als Fundament dieser

[2]Datensatz verfügbar unter: https://archive.ics.uci.edu/dataset/321/electricityloaddiagrams20112014
[3]Datensatz verfügbar unter: https://www.bgc-jena.mpg.de/wetter/

Arbeit dienen. Diese Gegenüberstellung soll die Effektivität der neu entworfenen Architektur anhand der erzielten Ergebnisse darstellen. Zu den Vergleichsmodellen gehören zum einen das auf der Transformer-Architektur basierende Modell SAITS [19] sowie ein Standard-Transformer ohne spezifische Modifikationen. Darüber hinaus wird das Modell TimesNet herangezogen, das als Foundation-Modell für diverse Zeitreihenaufgaben entwickelt wurde [66]. Weitere Modelle werden in diesem Kontext nicht betrachtet.

Das vorgestellte Framework in Kap. 4 zeichnet sich durch seine modulare Struktur aus. Dies ermöglicht den Austausch der zentralen Komponente, nämlich dem vortrainierten LM. In dieser Arbeit werden unterschiedliche Modelle ausgewählt, zu denen sowohl vier PLMs als auch zwei LLMs gehören. Die spezifischen Modelle werden nachstehend aufgeführt und unterscheiden sich primär in der Anzahl der Parameter. Es ist hervorzuheben, dass die Gewichte aller in dieser Arbeit verwendeten Modelle als Open-Source verfügbar sind. Die Modelle können daher auch für weiterführende Forschungsarbeiten verwendet werden.

GPT-2. Die von OpenAI entwickelte Generative Pretrained Transformer (GPT) Architektur basiert auf dem Transformer-Prinzip, verwendet jedoch ausschließlich den Decoder-Teil der ursprünglichen Architektur. Das Modell wurde in vier unterschiedlichen Größen veröffentlicht, wobei das Basemodell zwölf Layer und zwölf Attention Heads besitzt. Die Modelldimension d_{model} beträgt 768. Insgesamt umfasst GPT-2 ca. 117 Millionen Parameter und wurde für die Anpassung an NLP-Aufgaben mit etwa 40 GB an Daten bestehend aus ca. acht Millionen Dokumenten trainiert. Durch seine Architektur bedingt, kann das Modell ausschließlich auf vorangegangene Tokens zugreifen. Die Art des Trainings ist also unidirektional. Weiterhin wurde das GPT-2 Modell mit dem Language Modeling Ansatz vortrainiert und anschließend auf spezifische Tasks optimiert [51].

BERT. Im Gegensatz zu GPT-2 ist BERT ein *Encoder-only-Modell* und wurde von Google veröffentlicht. BERT steht dabei für **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Durch die Einbeziehung bidirektionaler Kontextinformationen ist es dem Modell möglich, einen Zugriff sowohl auf den links- als auch den rechtsseitigen Kontext innerhalb von Sprachrepräsentationen zu erhalten, wodurch ein ausgeprägtes Sprachverständnis erzielt wird. Mit zwölf Schichten, zwölf Attention-Köpfen und einer Modelldimension d_{model} von 768 ist die Modellarchitektur von BERT ähnlich zu GPT-2. Auch die Parameterzahl des Basismodells mit rund 110 Millionen ist vergleichbar. Das Prinzip des Vortrainings bei BERT weicht vom klassischen Language Modeling ab und ist zweifach ausgerichtet. Einerseits wird das *Masked Language Modeling* eingesetzt, bei dem zufällige

Tokens in einer Sequenz maskiert werden und das Modell darauf trainiert wird, diese zu rekonstruieren [17]. Trotz der namentlichen Ähnlichkeit unterscheidet sich dieser Ansatz vom Language Modeling, da der Ansatz den Prinzipien des Denoising Autoencoding (siehe Kap. 2.4.2) folgt und Ähnlichkeiten zur Imputationsaufgabe aufweist. Zusätzlich zu dem Masked Langue Modeling wurde BERT darauf trainiert, den nächsten Satz vorherzusagen (Next Sentence Prediction). Anschließend an die Aufgaben des Pre-Trainings fand ein Fine-Tuning auf den unterschiedlichen Tasks des NLPs statt [17].

T5. Das T5 Modell wurde zeitlich gesehen nach BERT veröffentlicht und ist ebenfalls ein Forschungsergebnis von Google. Dieses Modell baut auf der klassischen Transformer-Architektur auf und verwendet einen Encoder sowie Decoder mit wenigen Anpassungen der ursprünglichen Variante. Sowohl der Encoder als auch der Decoder verfügen jeweils über insgesamt zwölf Schichten. Das 220 Millionen Parameter große Modell ist ähnlich wie BERT auf einem Ansatz vortrainiert, der die Input Sequenzen manipuliert und Tokens maskiert. Bereits während des Trainings lernt das Modell auf unterschiedlichen Aufgaben. Hier sprechen die Autoren von *Multi-Task Pre-Training* [52]. Nach dem Pre-Training findet ein Fine-Tuning auf weiteren unterschiedlichen NLP Tasks statt.

BART. Zusätzlich zu BERT und T5 ist BART ein weiteres LM, dass auf dem Ansatz des Denoising Autoencodings trainiert wurde. Das Encoder-Decoder basierte Modell baut ebenfalls auf der Transformer-Architektur auf und besitzt jeweils sechs Layer mit einer Modelldimension d_{model} von 768 in beiden Komponenten. Im Mittelpunkt des Pre-Trainings steht die Veränderung der Inputsequenz, bspw. durch Token Deletion, Masking oder auch Token Rotation, und die anschließende Wiederherstellung der ursprünglichen Sequenz. Das Grundmodell wird anschließend auf die spezifischen NLP-Tasks wie Klassifikation, Textgenerierung, etc. angepasst. In seiner Basisvariante kommt BART auf ca. 140 Millionen Parameter und liegt damit in der Größenordnung der vorher genannten Modelle [38].

Im nächsten Abschnitt erfolgt die Darstellung von zwei LLMs, die sich insbesondere hinsichtlich der Anzahl an Parametern signifikant von den zuvor beschriebenen Modellen unterscheiden. Ein zusätzliches Differenzierungsmerkmal dieser Modelle liegt im Umfang der Trainingsdaten, die für das Pre-Training verwendet wurden. Im Gegensatz zu den früheren Datensätzen, die etwa maximal 130 Milliarden Tokens umfassten, wurden diese Modelle mit Trainingsdaten in der Größenordnung von Billionen von Tokens optimiert [52].

Llama2. Das von Meta veröffentlichte Modell Llama2 ist eines der ersten öffentlich zugänglichen LLMs, das in gewissen Benchmarks mit proprietären Modellen konkurrieren konnte [60]. Das Modell wurde in drei Größen (7, 13 und 70 Milliarden Parametern) herausgegeben und übertrifft somit die Parameteranzahl der vorherigen Modelle signifikant. Als reines Decoder-Modell mit 32 Attention Heads und 32 Schichten, zeichnet sich Llama2 zudem durch eine Modelldimension d_{model} von 4096 aus, die vorherige Modelldimensionen deutlich übersteigt. Das Training des Modells wurde auf einem speziell vorverarbeiteten Datensatz mit zwei Billionen Tokens durchgeführt. Im Vergleich zur klassischen Transformer Decoder-Architektur wurden Anpassungen vorgenommen, die eine Optimierung der Leistung ermöglichen sollen. Diese Anpassungen sind an dieser Stelle allerdings nicht Gegenstand der Betrachtung. Für weitere Details bietet das Paper zu Llama2 die Grundlage [60]. Das Trainingsziel des Pre-Trainings ist bei Llama2 das Language Modeling, also die Vorhersage des nächsten Wortes einer Sequenz. Das weiterführende Fine-Tuning wird mit unterschiedlichen Ansätzen durchgeführt, zu denen das Instruction Tuning und Reinforcement Learning with Human Feedback (RLHF) gehören. Aufgrund des hohen Ressourcenbedarfs der größeren Modelle wird für die nachfolgenden Experimente das kleinste Modell von Llama2 verwendet.

Phi-2. Das zweite LLM ist Phi-2. Dieses Modell besitzt zwei Milliarden Parameter und ist von der Architektur sehr ähnlich zu Llama2. Das Decoder-Only-Modell (24 Layer, 32 Attention Heads) baut wie auch die anderen Modelle auf dem Transformer auf und verwendet die Flash Attention im Vergleich zur klassischen Attention im Transformer. Die Pre-Training-Task ist das Language Modeling, sodass das Modell explizit darauf trainiert ist, das nächste Wort eines Satzes vorherzusagen. Besonders an dem Modell ist, dass es keinem Fine-Tuning unterzogen wurde. Die Herausgeber von Microsoft zeigen trotz des fehlenden Fine-Tunings, dass Phi-2 vorherige Modelle bei gewissen Benchmarks übertrifft [32]. Dies wird durch eine explizite Auswahl an hochqualitativen Trainingsdaten in Fachliteratur-Qualität erreicht. Es wird deutlich, dass die Beschränkung der Modellgröße durch sehr gute Trainingsdaten ausgeglichen werden kann [32]. Eine Übersicht der Modelle, mitsamt der erwähnten Details bezüglich Modelldimension, Anzahl von Parametern, etc., befindet sich in Anhang A.1. Obwohl in der vorherigen Darstellung eine Unterteilung in PLMs und LLMs vorgenommen wurde, sollen die Modelle aller Größenordnungen im Folgenden zur Vereinfachung unter dem Begriff der LLMs zusammengefasst werden.

5.1.3 Details der Implementierung

Das hier dargestellte Setup zeigt den generellen Aufbau für die folgenden Experimente. Alle der dargestellten Modelle werden bzgl. der Imputation angepasst und innerhalb des konstruierten Rahmens (siehe Kapitel 4) ausgetauscht. Es werden somit die Positional Embeddings und die Add & LN-Komponente der Transformer-Blöcke der LLMs angepasst. Darüber hinaus werden die Attention Blöcke mit Hilfe der parametereffizienten Methode LoRA angepasst. Beim Feinabstimmungsprozess der Modelle kommt der Mean Squarred Error (MSE) zum Einsatz, während der Mean Absolute Error (MAE) zusätzlich zur Validierung der Ergebnisse herangezogen wird. Die vorhandenen Zeitreihen aus den Datensätzen werden jeweils in Sequenzen mit einer Länge von 96 eingeteilt. Im Rahmen des Trainings werden die Zeitreihen dann zufällig maskiert, sodass fehlende Daten des Typs MCAR vorliegen und isolierte sowie sequenzielle Lücken entstehen können. Das Ziel des Trainings ist es, die maskierten Werte wiederherzustellen. Im Rahmen des zufälligen Maskierens der Werte werden unterschiedliche Häufigkeiten für das fehlen der Werte angenommen. So haben die unterschiedlichen Experimente Fehlraten von 12,5%, 25%, 37,5% und 50%. Die hier genannten Details der Implementierung adaptieren das Setup der Experimente aus [66] und [74]. Gemäß der Methode von Zhou et al. [74] erfolgt eine Unterteilung der Sprachmodelle in kleinere Modelleinheiten, indem ein Viertel der ursprünglichen Modellschichten für Optimierungszwecke eingesetzt wird. Im Detail bedeutet dies, dass nur ein Teil der vollständigen Modelle verwendet wird. Somit ergeben sich für GPT-2, BERT und T5 jeweils $N = 3$ Layer. Llama2 und Phi-2 haben entsprechend $N = 8$ Layer. An dieser Stelle ist zu erwähnen, dass Phi-2 laut Veröffentlichung 24 Layer besitzt. Trotz dessen stellt Microsoft das Modell auf der Huggingface Plattform als 32 Layer Modell bereit[4], welches für die Versuche verwendet wird. BART als Encoder-Decoder Modell besitzt insgesamt zwölf Layer, die sich zu gleichen Teilen in dem Encoder und dem Decoder wiederfinden. Da eine Aufteilung zu 25% nicht möglich ist, bleiben in den beiden Komponenten jeweils zwei Transformer Blöcke übrig.

Bezüglich der Implementierung der Fine-Tuning Methode LoRA wird auf die Umsetzung von Huggingface verwiesen [45] und folgende Konfiguration für alle Versuche angenommen, in denen LoRA zum Einsatz kommt.

- LoRA Attention Dimension: $r = 8$
- Alpha Parameter für die Skalierung: $lora\ alpha = 16$

[4]Phi-2 Huggingface Model: https://huggingface.co/microsoft/phi-2

- Dropout für LoRA-Layer: $lora\ dropout = 0.05$
- Bias Layer des LoRA Adapters: $bias = none$

Ein weiterer wichtiger Parameter der LoRA Konfiguration sind die Zielmodule (Target Modules). Dieser Parameter benennt die Schichten, welche als LoRA abgebildet werden sollen. Je nach dem angewendeten Modell unterscheiden sich die Namen der Aufmerksamkeitsschichten. Ziel des Trainings sind im Generellen allerdings die *Key, Query* und *Value* Komponenten der LLMs. Das komplette Setup für die Versuche ist als Github Repository öffentlich zugänglich[5]. An dieser Stelle ist zu erwähnen, dass die hier dargestellten Implementierungsdetails zentral für die erste Fragestellung dieser Arbeit gelten. Das übergreifende Konstrukt gilt allerdings für alle Versuche. Notwendige Änderungen für die Durchführung weiterführender Experimente zur Beantwortung der anderen Fragestellungen werden in den jeweiligen Unterkapiteln aufgeführt. In dem folgenden Kapitel der Ergebnisdarstellung werden die einzelnen Versuche zur Beantwortung der Forschungsfragen dargestellt.

[5]Github Repository Imputation with LLMs: https://github.com/mijacobsenHAW/time-series-imputation-with-llms

5.2 Ergebnisdarstellung

Das Kapitel der Ergebnisdarstellung folgt der Struktur der Forschungsfragen und stellt die Ergebnisse der Versuche dar. Dafür werden für jeden Versuch notwendige technische Anpassungen vorgestellt und eine Abgrenzung zu den anderen Versuchen angeführt. Die detaillierte Diskussion und Interpretation der Ergebnisse folgt im Kapitel 5.3.

5.2.1 Leistungsvergleich vortrainierter Sprachmodelle

In diesem Abschnitt erfolgt die Präsentation der Ergebnisse bezüglich der ersten Forschungsfrage, die einen Vergleich der Leistungsfähigkeit aktueller State-of-the-Art Methoden mit der Imputationstechnik unter Einsatz von LLMs vorsieht. Die Durchführung des Experiments folgt dem in Kapitel 5.1 dargelegten allgemeinen Experiment-Aufbau. Als Vergleichsgrundlage zu den LLMs dienen die bereits erwähnten Modelle SAITS, ein traditionelles Transformer-Modell sowie TimesNet. Die einheitliche Optimierungskonfiguration (LoRA) für die Sprachmodelle führt zu den Modellgrößen, welche in Tabelle 5.2 dargestellt sind. Entsprechend der Ausgangsgröße verringert sich die Gesamtanzahl der Parameter durch die zielgerichtete Auswahl einer Teilmenge der Modellschichten. Je nach Ausgangsgröße bleiben bis zu 50% der ursprünglichen Größe übrig (siehe z.B. GPT-2, ursprünglich 120 Mio. Parameter). Die adaptiven Parameter sind während des Optimierungsprozesses nicht eingefroren und werden für die Adaption auf die Imputationsaufgabe trainiert. Beim Llama2 Modell sind bspw. lediglich 0,12% der Parameter trainierbar. In Summe ergeben sich trotz dessen ca. zwei Milliarden adaptive Parameter. Der Anteil der Parameter ist abhängig von der Modellarchitektur und den inneren Modelldimensionen. Daher ist hier eine Schwankung der Anteile nicht zu verhindern und eine Vereinheitlichung nicht zielführend.

Tabelle 5.2: Anzahl adaptiver Parameter der Sprachmodelle

Modell	Gesamtanzahl	Adaptive Parameter	Anteil (in %)
GPT-2	60.759.552	906.240	1,49
BERT	45.838.848	551.424	1,20
Llama2	1.752.240.128	2.162.688	0,12
Phi-2	761.758.720	1.351.680	0,18
T5	74.676.416	456.128	0,61
BART	73.556.736	1.883.136	2,56

Tabelle 5.3: Ergebnisse aggregiert pro Datensatz (alle Modelle)

	GPT-2		BERT		Saits		TimesNet		Transformer		Llama2		Phi-2		T5		BART	
Datensatz	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	0,087	0,194	0,075	0,182	**0,044**	**0,139**	0,093	0,204	0,096	0,217	**0,064**	**0,167**	0,066	0,172	0,077	0,187	0,082	0,190
ETTh2	0,054	0,152	**0,049**	**0,146**	0,081	0,196	0,052	0,152	0,249	0,362	0,051	0,147	0,050	**0,145**	**0,049**	0,147	0,050	0,148
ETTm1	0,038	0,129	0,027	0,110	**0,020**	**0,092**	0,028	0,109	0,032	0,123	**0,027**	**0,105**	0,027	0,106	0,033	0,122	0,031	0,117
ETTm2	0,026	0,100	**0,022**	**0,090**	0,039	0,132	**0,022**	**0,091**	0,189	0,315	0,025	0,095	0,026	0,097	0,024	0,095	0,024	0,095
Electricity	0,084	0,201	0,084	0,200	0,165	0,284	0,097	0,213	0,162	0,286	**0,080**	**0,196**	**0,080**	**0,196**	0,090	0,209	0,088	0,205
Weather	0,031	0,057	0,042	0,070	0,037	0,083	**0,030**	**0,055**	0,036	0,087	0,045	0,076	0,070	0,111	**0,030**	**0,056**	0,035	0,067

*Legende: **Bester Wert**, **Zweitbester Wert**

Die Ergebnisse dieses Versuches sind in Anhang A.2 zusammengefasst. Die Übersicht der Ergebnisse im Anhang zeigt, dass es kein Modell gibt, welches auf allen Datensätzen und Fehlraten dominiert. Insgesamt sind die besten Leistungen über einen Großteil der Modelle verteilt. Es wird allerdings deutlich, dass die spezialisierten Modelle SAITS und TimesNet insgesamt gute Ergebnisse liefern. SAITS übertrifft bei zwei Datensätzen (ETTh1 & ETTm1) die anderen Modelle deutlich und hat insgesamt bei acht Datensatz-Fehlraten-Kombinationen beim MSE und MAE die geringsten Fehler. Andere gute Modelle sind BERT und TimesNet bezogen auf den MSE sowie Phi-2 beim MAE. Die schwankende Verteilung der niedrigsten Fehler zeigt sich auch darin, dass keines der LLMs immer den zweitbesten Wert im Test hervorbringt. Hier wechseln sich Modelle wie BERT, Llama2, Phi-2 und T5 jeweils ab. Dennoch wird deutlich, dass die Modelle GPT-2 und BART im Vergleich zu anderen Modellen selten die besten Ergebnisse erzielen. Es wird ebenfalls ersichtlich, dass das klassische Transformer-Modell nicht auf demselben Niveau wie die anderen Modelle agiert. Obwohl es in einigen Experimenten vergleichbare Leistungen erbringt, gelingt es in keinem Versuch, sich als führend zu positionieren. Das ganzheitliche Bild dieser Ergebnisse zeigt sich ebenfalls bei der Betrachtung der aggregierten Ergebnisse pro Datensatz. Die besten Leistungen erzielen SAITS und die LLMs Phi-2, BERT und T5 (siehe Tab. 5.3).

Angesichts der Uneindeutigkeit der vorliegenden Ergebnisse erfolgt eine spezifische Betrachtung der Versuchsergebnisse der LLMs. Diese Darstellung soll aufzeigen, inwieweit sich die Ergebnisse in der Teilmenge der Sprachmodelle vom Gesamtbild unterscheiden oder Auffälligkeiten deutlich werden. Tabelle 5.4 bietet eine ausschließliche Zusammenfassung der Ergebnisse der Sprachmodelle und bildet eine Teilmenge der Ergebnisse aus Anhang A.2 ab. Aufgrund der ausschließlichen Betrachtung der LLMs ist die Tabelle erneut abgebildet, damit eine übersichtliche Darstellung möglich ist. Die Analyse der Versuchsdaten zeigt, dass die minimalen Fehlerquoten sowohl bei den großen Sprachmodellen Llama2 und Phi-2 als auch bei den signifikant kleineren Modellen BERT und T5 verzeichnet wurden. Es zeichnet sich zudem eine Tendenz ab, nach der entweder die

Tabelle 5.4: Vollständige Ergebnisse (nur LMs)

Modelle		GPT-2		BERT		Llama2		Phi-2		T5		BART	
	Mask Ratio	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	12,5%	0,0524	0,1542	0,0466	0,1461	**0,0409**	**0,1377**	**0,0415**	**0,1393**	0,0513	0,1553	0,0527	0,1553
	25%	0,0738	0,1821	0,0638	0,1696	**0,0516**	**0,1528**	**0,0551**	**0,1595**	0,0688	0,1779	0,0673	0,1752
	37,5%	0,0979	0,2071	0,0837	0,1942	**0,0689**	**0,1750**	**0,0705**	**0,1796**	0,0840	0,1959	0,0920	0,2045
	50%	0,1228	0,2317	0,1068	0,2163	**0,0927**	**0,2036**	**0,0955**	**0,2080**	0,1049	0,2184	0,1145	0,2250
ETTh2	12,5%	0,0414	0,1310	**0,0395**	**0,1276**	0,0407	0,1307	0,0410	**0,1299**	**0,0397**	0,1307	0,0406	0,1324
	25%	0,0496	0,1465	**0,0458**	**0,1406**	0,0478	0,1428	**0,0458**	**0,1383**	0,0468	0,1436	0,0471	0,1435
	37,5%	0,0567	0,1578	**0,0519**	0,1516	0,0532	0,1517	0,0525	**0,1495**	**0,0517**	**0,1511**	0,0534	0,1534
	50%	0,0666	0,1717	0,0605	0,1649	0,0615	0,1629	0,0606	**0,1615**	**0,0579**	**0,1608**	**0,0603**	0,1642
ETTm1	12,5%	0,0244	0,1045	0,0196	0,0931	**0,0193**	**0,0911**	**0,0188**	**0,0904**	0,0241	0,1049	0,0223	0,1002
	25%	0,0329	0,1212	0,0238	0,1029	**0,0233**	**0,0985**	**0,0226**	**0,0979**	0,0304	0,1168	0,0270	0,1106
	37,5%	0,0413	0,1356	0,0297	0,1151	**0,0288**	**0,1090**	**0,0286**	**0,1094**	0,0356	0,1264	0,0331	0,1218
	50%	0,0533	0,1529	**0,0364**	0,1277	**0,0361**	**0,1214**	0,0381	**0,1260**	0,0421	0,1379	0,0411	0,1351
ETTm2	12,5%	0,0209	0,0872	**0,0185**	**0,0802**	0,0210	0,0872	0,0226	0,0897	0,0205	0,0868	**0,0200**	**0,0855**
	25%	0,0250	0,0970	**0,0206**	**0,0855**	0,0236	0,0917	0,0231	**0,0914**	**0,0226**	0,0917	0,0229	0,0927
	37,5%	0,0282	0,1044	**0,0233**	**0,0930**	**0,0249**	**0,0953**	0,0269	0,0996	0,0250	0,0980	0,0251	0,0974
	50%	0,0313	0,1112	**0,0264**	**0,0997**	0,0292	0,1048	0,0301	0,1067	**0,0282**	0,1052	0,0284	**0,1048**
Electricity	12,5%	0,0716	0,1860	0,0713	0,1857	**0,0650**	**0,1766**	**0,0654**	**0,1769**	0,0804	0,1977	0,0778	0,1931
	25%	0,0800	0,1963	0,0795	0,1956	**0,0770**	**0,1923**	**0,0755**	**0,1901**	0,0867	0,2047	0,0841	0,2004
	37,5%	0,0886	0,2068	0,0873	0,2049	**0,0852**	**0,2030**	**0,0846**	**0,2018**	0,0933	0,2127	0,0908	0,2086
	50%	0,0961	0,2162	0,0963	0,2158	**0,0942**	**0,2139**	**0,0943**	**0,2135**	0,1005	0,2215	0,0988	0,2183
Weather	12,5%	**0,0260**	0,0476	0,0269	**0,0496**	0,0265	**0,0480**	0,0803	0,1242	**0,0261**	0,0497	0,0296	0,0580
	25%	0,0297	**0,0545**	**0,0295**	**0,0547**	0,0310	0,0566	0,0829	0,1298	**0,0291**	0,0540	0,0342	**0,0654**
	37,5%	0,0330	**0,0602**	**0,0324**	**0,0591**	0,0352	0,0645	0,0866	0,1369	**0,0316**	0,0588	0,0347	0,0647
	50%	**0,0373**	0,0669	**0,0374**	0,0662	0,0393	**0,0700**	0,0837	0,1271	**0,0349**	0,0632	0,0433	**0,0802**
Count 1st		1	1	**6**	6	**7**	**9**	5	**7**	5	1	0	1
Count 2nd		1	1	5	3	**6**	**6**	**7**	**10**	4	1	2	3

*Legende: **Bester Wert**, **Zweitbester Wert**

größeren oder die kleineren Modelle dominieren. Konkret bedeutet dies, dass eine hohe Leistungsfähigkeit eines großen Sprachmodells wie Llama2 typischerweise mit einer nahezu ebenso hohen Performance von Phi-2 korrelieren (siehe Tab. 5.4 Datensatz ETTh1, ETTm1, Electricity), während bei den Datensätzen ETTh2 und Weather primär BERT und T5 die niedrigsten Fehlerwerte erzielen. Hier ist jedoch zu erwähnen, dass Llama2 sowie auch Phi-2 bei einigen Fehlraten eine führende Rolle einnehmen.

Zur Veranschaulichung der Ergebnisse werden im folgenden Abschnitt ausgewählte Abbildungen herangezogen, die das Imputationsverhalten der untersuchten Modelle miteinander vergleichen. Abb. 5.1 zeigt eine Zeitreihe, die von unterschiedlichen Modellen nach vorheriger Maskierung vervollständigt wird, wobei eine Fehlrate von 50% vorliegt und der zugrundeliegende Datensatz ETTm2 ist. Im Fokus des Vergleichs stehen die genannten Modelle, die im Vergleich der LLMs eine führende Rolle einnehmen. Die einzelnen Diagramme visualisieren eine Zeitreihe mit 96 Zeitschritten sowie den entsprechenden normierten Werten für eine Variable. Überlappungen der blauen (Predictions) und roten

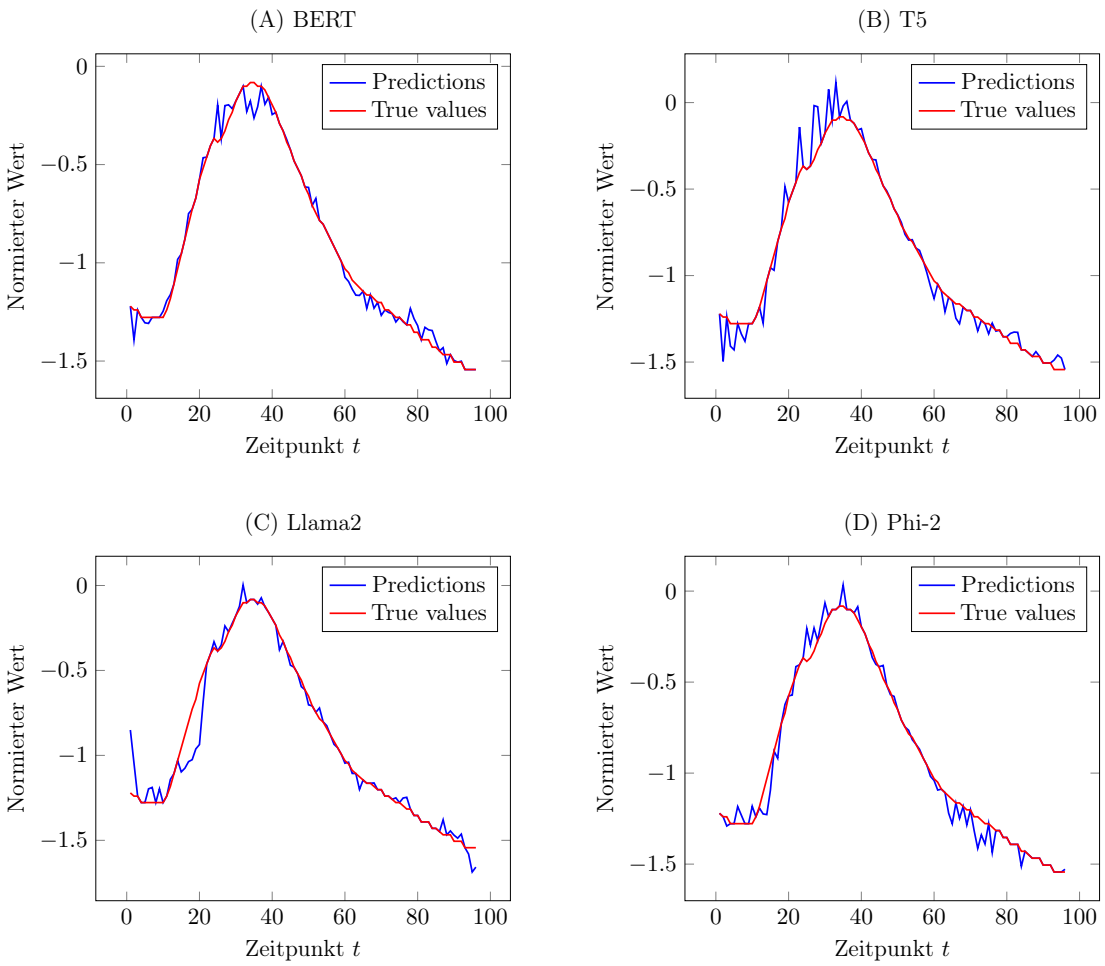

Abbildung 5.1: Imputierte Zeitreihen verschiedener Modelle, ETTm2, Fehlrate 50%. Alle Modelle imputieren die wahre Zeitreihe (True Values) gut. Die Imputation durch T5 schwankt bis zum Zeitpunkt $t = 40$ stärker als die anderen imputierten Werte. Bert und Llama bieten die besten Ergebnisse bezogen auf die abgebildete Zeitreihe.

Linien (True Values) zeigen Bereiche ohne Datenmaskierung. In den anderen Bereichen wurden die fehlenden Daten durch das jeweilige Modell ersetzt. Die Abbildung unterstreicht, dass alle Modelle in der Lage sind, angenäherte Werte zu generieren und allgemein eine hohe Imputationsleistung zu beobachten ist. Die Modelle T5 (B) und Phi-2 (D) zeigen im Durchschnitt bei isolierter Betrachtung dieser Zeitreihe eine stärkere Abweichung vom Ist-Zustand als BERT (A) und Llama2 (C). BERT zeichnet sich dabei durch die Generierung von imputierten Werten aus, die den ursprünglichen Daten sehr nahekommen und nur geringe Schwankungen aufweisen. T5 dagegen zeigt vor allem bis Zeitpunkt $t = 40$ eine Tendenz zu einer stärkeren Schwankung (siehe Abb. 5.1, (B)). Die Abbildung zeigt insgesamt die hohe Imputationsleistung aller vier Modelle. Besonders BERT sticht hervor, was auch in den umfassenden Daten der Tabelle 5.4 abgebildet ist. Das beschriebene Gesamtbild der Ergebnisse ergibt sich ebenfalls bei Betrachtung auf aggregierter Sicht pro Datensatz. Llama2 und Phi-2 erzielen insgesamt die geringsten Fehlerquoten (ETTh1, ETTm1, Electricity), wobei die kleineren Modelle BERT und T5 bei ausgewählten Datensätzen (ETTh2, ETTm2) ebenfalls führende Leistungen zeigen (siehe Tab. 5.5).

Tabelle 5.5: Ergebnisse aggregiert pro Datensatz (nur LMs)

	GPT-2		BERT		Llama2		Phi-2		T5		BART	
Datensatz	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	0,0867	0,1938	0,0752	0,1815	**0,0635**	**0,1673**	**0,0657**	**0,1716**	0,0773	0,1869	0,0816	0,1900
ETTh2	0,0536	0,1518	**0,0494**	**0,1462**	0,0508	0,1470	0,0500	**0,1448**	**0,0490**	0,1465	0,0504	0,1484
ETTm1	0,0380	0,1285	0,0274	0,1097	**0,0268**	**0,1050**	**0,0270**	**0,1059**	0,0331	0,1215	0,0309	0,1169
ETTm2	0,0264	0,1000	**0,0222**	**0,0896**	0,0247	**0,0947**	0,0257	0,0969	**0,0240**	0,0954	0,0241	0,0951
Electricity	0,0841	0,2013	0,0836	0,2005	**0,0804**	**0,1965**	**0,0799**	**0,1956**	0,0902	0,2092	0,0879	0,2051
Weather	**0,0315**	**0,0573**	0,0316	0,0574	0,0330	0,0598	0,0833	0,1295	**0,0304**	**0,0564**	0,0355	0,0670

*Legende: **Bester Wert**, **Zweitbester Wert**

5.2.2 Optimierung spezifischer Modellkomponenten

Das Fine-Tuning der Modelle aus dem vorherigen Abschnitt folgt einem einheitlichen Vorgehen, nach dem alle Modelle konsistent trainiert wurden. Zur Untersuchung der zweiten Forschungsfrage wird im Folgenden nun ein abweichender experimenteller Ansatz vorgestellt, der speziell das Fine-Tuning verschiedener Komponenten der Transformer-Architektur betrachtet. Im Fokus steht die gezielte Optimierung unterschiedlicher Schichten des Modells. Basierend auf der Transformer-Architektur (siehe Kapitel 2.4.1) bietet die Struktur der LLMs die Möglichkeit, einzelne oder mehrere Schichten gezielt zu optimieren. Insbesondere die Attention Layer, das FFN sowie die Add & LN-Komponente

Tabelle 5.6: Komponenten-Tuning - Anzahl der trainierbaren Parameter

Modellkomponente	Gesamtanzahl	adaptive Parameter	Anteil (in %)
Attention	60.648.960	46.470.912	76.62
Add & LN	60.648.960	39.394.560	64.96
FFN	60.648.960	53.551.104	88.30
Attention + FFN	60.648.960	60.638.208	99.98
FFN + Add & LN	60.648.960	53.561.856	88.31
Attention + Add & LN	60.648.960	46.481.664	76.64
Attention + Add & LN + FFN	60.648.960	60.648.960	100.00

des Transformer-Blocks werden sowohl einzeln als auch in Kombination untersucht. Als Grundlage für die Untersuchung dient das GPT-2 Modell [51], welches bereits im ersten Versuch verwendet wurde und den primären Bestandteil der dargestellten verwandten Arbeiten im Kapitel 3 bildet. Darüber hinaus entspricht die Architektur von GPT-2 dem grundsätzlichen Aufbau der großen führenden Sprachmodelle (hier Llama2, Phi-2), sodass die Ergebnisse auch auf diese Modelle übertragbar sind und die Wahl von GPT-2 eine Relevanz mit sich bringt.

Das GPT-2 Modell hat ca. 120 Millionen Parameter. Durch die Anpassungen und das Fine-Tuning von drei der insgesamt zwölf möglichen Schichten, reduziert sich die Parameteranzahl auf ungefähr 60 Millionen. Da die verschiedenen Modellkomponenten einen unterschiedlichen Anteil an den gesamten Parametern haben, fasst Tabelle 5.6 die sich ergebenden prozentualen Anteile zusammen. Im Rahmen der Versuche werden die Input Embeddings des Modells (Word Token Embedding, Word Positional Embedding) immer optimiert. Es ergibt sich somit eine Grundanzahl von Parametern, die anpassbar sind. Zusätzlich wird in jedem Lauf eine bestimmte Komponente oder eine Kombination der Komponenten angepasst. Die Ergebnisse des Komponenten-Trainings sind in Tabelle 5.7 in einer aggregierten Form zusammengefasst. Die vollständigen Ergebnisse des zweiten Versuches befinden sich in Anhang A.3.

Die Zusammenfassung zeigt, dass das Fine-Tuning des Modells und die Anpassung auf die Imputation nicht alleine durch eine Optimierung einer einzigen Komponente signifikant besser wird. Die Optimierung der Attention (1) oder FFN (3) Schicht führt insgesamt zu einer verbesserten Imputationsleistung im Vergleich zur Anpassung mittels Add & LN-Fine-Tuning. Dieses Verhalten ist für fünf der sechs Datensätze zu beobachten. Im Durchschnitt imputiert die Kombination aus der Attention- und FFN- Komponente am besten (siehe Tab. 5.7, Spalte 1 & 3). Diese Leistung zeigt sich besonders auf den ETT-Daten, so-

Tabelle 5.7: Komponenten-Tuning - Aggregierte Ergebnisse

	Attention (1)		Add & LN (2)		FFN (3)		1 & 3		2 & 3		1 & 2		1 & 2 & 3	
Datensatz	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	0,0767	0,1842	0,1005	0,2064	0,0734	0,1795	**0,0688**	**0,1724**	0,0698	0,1743	0,0753	0,1815	**0,0691**	**0,1730**
ETTh2	0,0513	0,1481	0,0563	0,1564	0,0510	0,1453	**0,0489**	**0,1415**	0,0499	0,1435	0,0506	0,1466	**0,0490**	**0,1418**
ETTm1	0,0316	0,1163	0,0454	0,1396	0,0293	0,1095	**0,0274**	**0,1058**	0,0287	0,1083	0,0311	0,1156	**0,0281**	**0,1072**
ETTm2	0,0248	0,0955	0,0279	0,1036	0,0228	0,0884	**0,0214**	**0,0854**	0,0224	0,0872	0,0241	0,0940	**0,0214**	**0,0854**
Electricity	**0,0844**	**0,2009**	0,0854	0,2033	0,0896	0,2056	0,0907	0,2073	0,0901	0,2064	**0,0843**	**0,2006**	0,0911	0,2075
Weather	0,0308	0,0555	0,0321	0,0581	0,0305	0,0545	0,0304	**0,0538**	**0,0302**	**0,0542**	0,0306	0,0554	**0,0303**	0,0540

*Legende: **Bester Wert**, **Zweitbester Wert**

dass hier das Fine-Tuning der angesprochenen Komponenten führende Ergebnisse erzielt. Im Bezug auf die anderen Datensätze (Electricity, Weather) ist ein nicht einheitliches Ergebnis-Bild zu beobachten. Bei der Imputation des Electricity-Datensatzes erzielt die Kombination aus Attention und Add & LN das beste Ergebnis. Bezüglich des Wetter-Datensatzes sind die Unterschiede in der Wiederherstellung der maskierten Zeitreihen sehr gering und alle Variationen bewegen sich in einem vergleichbaren Bereich. Auch wenn die Kombination aus Add & LN und FFN eine geringeren Fehlerwert aufweisen, ist das Ergebnis homogen. Neben der reinen Betrachtung der Imputationsleistung sollen die Ergebnisse unter Bezugnahme der adaptiven Modellparameter entsprechend Tab. 5.6 betrachtet werden. Im Allgemeinen zeigt die Tabelle, dass sich die Modellparameter primär erhöhen, wenn die Komponenten Attention oder FFN in den Optimierungsprozess mit einbezogen werden. Dies ist darin begründet, dass diese Komponenten den Hauptbestandteil der Transformer Blöcke ausmachen. Zur Darstellung der Modellleistung in Abhängigkeit der optimierbaren Parameter wird Abb. 5.2 herangezogen.

Die Abbildung zeigt die unterschiedlichen Kombinationen der Modellkomponenten in geordneter Reihenfolge nach ihrer Platzierung im Mittel über alle Datensätze hinweg. Die Höhe der Balken stellt die Anzahl der Parameter dar. Die einzelnen Balken sind nach Rangordnung sortiert. D.h., dass der Balken links (Add & LN (2)) die höchsten Fehlerraten und somit die geringste Imputationsleistung hat. Darauf folgt die Kombination mit der nächst besten Leistung. Insgesamt zeigt die Abbildung, dass die Kombination aus Attention- und FFN-Block (1 & 3) die besten Ergebnisse erzielt. Dieses Ergebnis wurde bereits durch Tab. 5.7 beschrieben. Darüber hinaus wird deutlich, dass die Leistungsfähigkeit mit steigender Anzahl an adaptiven Parametern zunimmt. Trotz dieser Tendenz ist auffällig, dass das volle Fine-Tuning aller Komponenten nicht die beste Imputationsleistung erbringt. Die Kombination aus allen Komponenten hat mehr adaptive Parameter (ca. 10.000) als die Kombination aus Attention und FFN, erzielt allerdings keine besseren Ergebnisse.

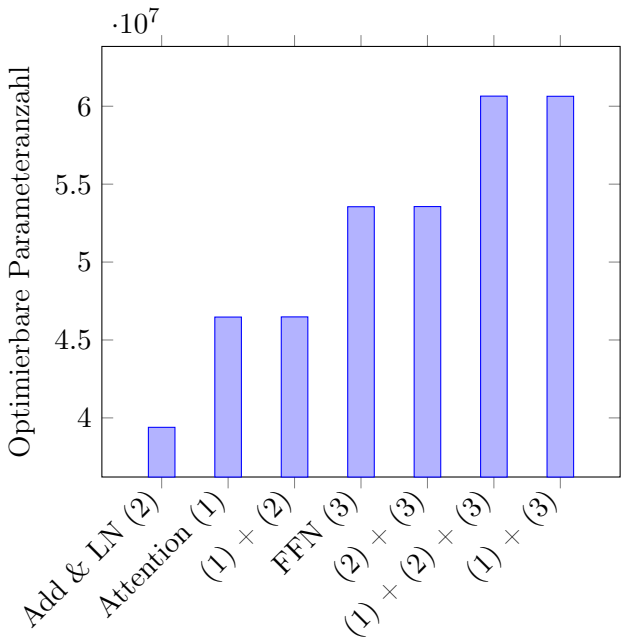

Abbildung 5.2: Imputationsleistung in Abhängigkeit von der Parameteranzahl. Die einzelnen Kombinationen der Transformer-Komponenten sind auf der X-Achse abgebildet und zur vereinheitlichten Darstellung nummeriert. Aufgrund der Skalierung scheinen einige der Balken gleich hoch zu sein. Die Kombinationen mit Add & LN-Komponente haben allerdings mehr Parameter, was in der Abbildung nicht direkt ersichtlich wird.

5.2.3 Vergleich ausgewählter Fine-Tuning-Methoden

In den bislang dargestellten Versuchen im Rahmen der Untersuchung der generellen Leistungsfähigkeit von LLMs (siehe Kapitel 5.2.1) wurde LoRA als Fine-Tuning Methode verwendet. Wie in den theoretischen Grundlagen bereits dargestellt, gibt es eine Reihe verschiedener Optimierungsmethoden, welche eine parameter-effiziente Optimierung der Modelle ermöglichen (Kapitel 2.4.2). Im Folgenden soll untersucht werden, welchen Einfluss die PEFT-Methode auf die Imputationsleistung des Modells hat. Zur Untersuchung dieses Aspektes wird der Versuchsaufbau für die erste Forschungsfrage verfolgt und innerhalb des modularen Frameworks die PEFT-Methode ausgetauscht. Zur Vereinheitlichung wird erneut das GPT-2-Modell als Ausgangsmodell für die notwendigen Anpassungen verwendet.

Die konkreten Methoden zur Optimierung sind LoRA, AdaLora, (IA^3) sowie die quantisierte Methode von LoRA, also Qlora. Obwohl Qlora keine klassische PEFT-Methode ist, sondern eine Memory-Efficient Optimierungsmethode, wird diese Methode aufgrund ihrer weiten Verbreitung beim Fine-Tuning von LLMs angewendet. Die Konfiguration der einzelnen Verfahren entspricht der LoRA-Parametrisierung aus dem Versuchsaufbau, im Speziellen den Details der Implementierung (Kapitel 5.1.3). Diese Konfiguration ist lediglich relevant für die Verfahren, welche LoRA ähneln und in der Implementierung die gleichen Parameter verwenden. An dieser Stelle wird auf die PEFT Bibliothek von Huggingface verwiesen[6]. Die Implementierung des (IA^3)-Verfahrens verlangt lediglich die Angabe von Zielmodulen. Dies überschneidet sich mit der LoRA-Konfiguration und betrifft die Attention Blöcke.

Im Anschluss an die Beschreibung des allgemeinen Versuchsaufbaus soll die Anzahl der adaptiven Parameter der einzelnen Methoden betrachtet werden. Ähnlich zu dem Verhalten beim Fine-Tuning unterschiedlicher Modellkomponenten, variiert bei der Wahl der Optimierungsmethode die Anzahl an trainierbaren Parametern. Tab. 5.8 zeigt für jede Methode die Anzahl der insgesamt optimierbaren Parameter sowie die zusätzlichen Parameter, die durch die jeweilige Methode hinzugefügt werden. Aufgrund des identischen Versuchsaufbaus dieses Experiments im Vergleich zu den Experimenten der ersten Forschungsfrage, sind einige Parameter in jedem Szenario optimierbar (Add & LN + Embedding). Diese Parameter in Kombination mit den zusätzlichen Parametern der gewählten Methode ergeben die insgesamt betrachteten Parameter in den Experimenten. Bei Anwendung von LoRA und Qlora werden etwa 110.000 Parameter ergänzt. Die Me-

[6]PEFT Library Github: https://github.com/huggingface/peft

Tabelle 5.8: PEFT-Tuning - Adaptive Parameter der Methoden

PEFT-Methode	zusätzliche Parameter	insgesamt
LoRA	110.592	906.240
AdaLora	165.960	961.608
(IA^3)	9.216	795.648
Qlora	110.592	906.240

thode AdaLora, die einen flexibleren Ansatz verfolgt, resultiert in einer größeren Zahl hinzugefügter Parameter (ungefähr 166.000). Das Verfahren (IA^3) weicht signifikant von den anderen Methoden ab und fügt dem Modell nur 9.216 Parameter hinzu. Die Ergebnisse des Versuchs sind in Anhang A.4 in vollständiger Darstellung abgebildet. Auf aggregierter Ebene je Datensatz fasst Tab. 5.9 die Ergebnisse zusammen. Im Detail zeigen die Resultate, dass die Methoden Qlora und LoRA zu sehr ähnlichen Fehlerwerten führen und vor allem auf den ETT-Datensätzen und dem Electricity Datensatz führende Ergebnisse aufweisen. Lediglich beim Wetter Datensatz ist eine Abweichung zu erkennen. Hier führt die Wahl der AdaLora-Methode zu besseren Ergebnissen, wobei auch die LoRA-Methode sehr gute Ergebnisse mit sich bringt. Bei einer Betrachtung der vollständigen Ergebnisse aus Anhang A.4 werden weitere Einblicke deutlich. Wie bereits beschrieben, dominieren die Methoden LoRA und Qlora die ETT-Datensätze. Auf aggregierter Sicht ist dieses ebenfalls für den Electricity Datensatz zu beobachten. Allerdings ist die Methode AdaLora bei diesem Datensatz ebenfalls sehr performant. Dieses Verhalten verstärkt sich dann beim Wetter-Datensatz, bei dem AdaLora eine führende Rolle einnimmt. Zusammenfassend bedeutet dies, dass die Methoden LoRA und Qlora sehr einheitliche Ergebnisse zeigen. Bei einem Wechsel des Datensatzes und somit der Domäne werden die Ergebnisse heterogener und die Methode AdaLora zeigt ebenfalls vergleichbare Ergebnisse. (IA^3) erzielt auf keinem der Datensätze führende Fehlerraten. Die Methode übertrifft einzig Qlora beim Wetter-Datensatz (siehe Tab. 5.9, Wetter-Datensatz). Ansonsten sinkt die Leistungsfähigkeit der Imputation bei dem Einsatz der (IA^3)-Methode. Im Anschluss an die Darstellung der Ergebnisse, folgt nun die Diskussion der Inhalte sowie die Beantwortung der Forschungsfragen.

Tabelle 5.9: PEFT-Tuning - Aggregierte Ergebnisse

	LoRA		AdaLora		(IA^3)		Qlora	
Datensatz	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	**0,0867**	**0,1938**	0,0915	0,1973	0,1005	0,2067	**0,0885**	**0,1942**
ETTh2	**0,0538**	**0,1519**	0,0542	0,1528	0,0564	0,1569	**0,0538**	**0,1522**
ETTm1	**0,0380**	**0,1285**	0,0414	0,1339	0,0452	0,1394	**0,0376**	**0,1281**
ETTm2	**0,0264**	**0,1000**	0,0270	0,1014	0,0280	0,1039	**0,0263**	**0,0996**
Electricity	**0,0841**	**0,2013**	0,0842	0,2017	0,0856	0,2036	**0,0842**	**0,2015**
Weather	**0,0315**	**0,0573**	**0,0316**	**0,0571**	0,0318	0,0577	0,0320	0,0579

*Legende: **Bester Wert**, **Zweitbester Wert**

5.3 Diskussion

Die dargestellten Ergebnisse verfolgen das Ziel, die aufgestellten Forschungsfragen möglichst detailliert beantworten zu können. Die Diskussion und Interpretation der Ergebnisse im Hinblick auf die Forschungsfragen sind Gegenstand dieses Kapitels. Zuerst folgt eine kurze Zusammenfassung der Ergebnisse. Anschließend werden die einzelnen Forschungsfragen diskutiert und relevante Erkenntnisse anderer Arbeiten miteinbezogen. Daraufhin folgen die Limitationen dieser Arbeit und mögliche Aspekte für die weitere Forschung im Bereich der Imputation mit LLMs.

Für die Durchführung der Experimente und die Beantwortung der Forschungsfragen wurde das Framework GPT4TS [74] adaptiert und PEFT-Methoden implementiert. Zentraler Gegenstand war der Einsatz verschiedener LLMs zum Vergleich der unterschiedlichen Eignung zur Imputation. In der Gesamtbetrachtung untermauern die Ergebnisse die Eignung von Sprachmodellen als Grundlage für die Imputation von Zeitreihendaten. Im direkten Vergleich mit aktuellen State-of-the-Art-Methoden zeigen Modelle wie Llama2, Phi-2 sowie BERT und T5 eine gleichwertige oder überlegene Leistungsfähigkeit. Darüber hinaus wird ersichtlich, das keines der Modelle über die Gesamtheit der durchgeführten Experimente immer überlegene Ergebnisse erzielt. Die genannten Modelle erbringen bei unterschiedlichen Datensätzen jeweils andere Ergebnisse. Bei der Betrachtung des Fine-Tunings unterschiedlicher Komponenten der Transformer-Architektur wird deutlich, dass eine Kombination aus Attention- und FFN-Schicht die höchste Leistungsfähigkeit erreicht. Die Add & LN-Komponente nimmt aufgrund der geringen Parameteranzahl im Vergleich zu den anderen Komponenten keine entscheidende Rolle ein. Obwohl generell das Fine-Tuning mit mehr Parametern eine bessere Leistung zu zeigen scheint, wird

ebenfalls deutlich, dass dies je nach Datensatz variieren kann. Neben der Analyse der unterschiedlichen Komponenten hat ein Vergleich der PEFT-Methoden aufgewiesen, dass der LoRA-Ansatz zu optimalen Ergebnissen führt. Die anderen angewendeten Verfahren (AdaLora, (IA^3)) sind zwar vergleichbar, aber erzielen keine führenden Ergebnisse.

RQ1: Inwieweit übertreffen vortrainierte Sprachmodelle die aktuellen State-of-the-Art Verfahren zur Imputation von Zeitreihendaten und spielt in diesem Kontext die Größe der ausgewählten Modelle eine entscheidende Rolle?

Vortrainierte Sprachmodelle erbringen vergleichbare Leistungen zu aktuellen State-of-the-Art Verfahren bei der Imputation von Zeitreihendaten und übertreffen diese auf einigen Datensätzen sogar. Es wird deutlich, dass die in dieser Arbeit ausgewählten LLMs insgesamt eine gute Grundlage für die Imputation darstellen. Die unterschiedliche Größe der Modelle, gemessen an der Parameteranzahl (siehe Anhang A.1), hat dabei einen Einfluss auf die Wiederherstellung der Zeitreihen. Diese Beobachtung begründet sich in den dargestellten Ergebnissen des vorangegangenen Kapitels. Die großen Sprachmodelle Llama2 und Phi-2 erzielen auf der Hälfte der Datensätze führende Ergebnisse. Im selben Zuge weisen die kleineren Modelle zum großen Teil eine limitierte Leistungsfähigkeit auf. D.h., dass auf diesen Datensätzen die Größe der Modelle ausschlaggebend für die Leistung ist. Hier ist zu erwähnen, dass die Betrachtung ausschließlich auf den Sprachmodellen stattfindet. SAITS hat auf den Datensätzen, auf denen Llama2 und Phi-2 gute Ergebnisse erzielen, eine führende Leistungsfähigkeit. Trotz dessen steht hier die Größe der Sprachmodelle im Mittelpunkt, sodass ausschließlich die LLM-Methoden betrachtet werden.

Neben der Größe der Sprachmodelle ist die Art des Pre-Trainings eines Modells für die Leistungsfähigkeit zur Imputation ausschlaggebend. BERT, T5 und BART sind wie beschrieben auf einem Ansatz trainiert, der die Input-Sequenzen verfälscht. Bei BERT ist es bspw. das Maskieren der Eingabesequenz. T5 wird auf einem ähnlichen Ansatz trainiert. Die Ergebnisse legen daher Nahe, dass die generelle Art des Pre-Trainings (siehe Abschnitt 2.4.2) ausschlaggebend für die Wiederherstellung der Zeitreihen ist. Die erwähnten Modelle sind deutlich kleiner und zeigen in denselben Experimenten auf einem Drittel der Datensätze bessere Ergebnisse als die großen Sprachmodelle. Schlussfolgernd bestätigen die Experimente somit die hohe Ähnlichkeit der Imputation von Zeitreihen mit der Methode des Denoising Autoencodings für LLMs. Daher ergibt sich, dass deutlich kleinere Sprachmodelle, wenn sie auf dem Denoising Autoencoding Ansatz trainiert wurden, eine bessere Fähigkeit zur Imputation besitzen. Die davon abweichende Trai-

ningsaufgabe des Language Modelings ist zwar auch zielführend, verlangt aber ein deutlich umfangreicheres Modell mit vielen Parametern. Diese allgemeine Beobachtung lässt sich bei BART allerdings nicht beobachten. BART wurde nach unterschiedlichen Ziel-Vorgaben trainiert. Dazu gehörten nicht nur das Maskieren der Input-Sequenz, sondern auch das Löschen oder die Rotation von Tokens (siehe Kap. 5.1.2). Es ist anzunehmen, dass diese Art des Vortrainings, obgleich es zum Teil dem Denoising Autoencoding Ansatz entspricht, nicht für die Übertragung auf die Imputation geeignet ist.

Insgesamt zeigt sich, dass mit dem angewendeten Framework eine Übertragung der Zeitreihenimputation auf die LLMs möglich ist und diese Ansätze eine führende Rolle einnehmen. Darüber hinaus weisen die Ergebnisse darauf hin, dass neben der Modellgröße auch die Art des Pre-Trainings sehr entscheidend sein kann. Im Umkehrschluss bedeutet dies, dass spezialisierte Modelle, die auf dieser Art des Pre-Trainings beruhen, für die Imputation von Zeitreihen besser geeignet sind und weniger Rechenressourcen benötigen. Darüber hinaus wird diese Annahme auch in der Betrachtung der Ergebnisse von SAITS und TimesNet unterstützt, da diese Modelle speziell für die Zeitreihenimputation konzipiert sind und eine vergleichbare Performance aufweisen.

RQ2: Welche Modellkomponenten der Sprachmodelle eignen sich für ein Fine-Tuning im Kontext der Imputation von Zeitreihendaten?

Im Rahmen verwandter Arbeiten wird nicht im Detail geprüft, welche der Modellkomponenten eines LLMs sich besonders für ein Fine-Tuning auf die Imputation von Zeitreihendaten eignen. Das Framework GPT4TS [74] geht davon aus, dass die meisten Kenntnisse eines Sprachmodells in den Attention-Schichten gespeichert sind und passt diese Schicht im Fine-Tuning-Prozess nicht an. Im Rahmen dieser Arbeit wurde bereits gezeigt, dass die Integration von PEFT-Methoden in der Attention-Schicht zu guten Ergebnissen führen kann. Darüber hinaus zeigen die Ergebnisse aus Kapitel 5.2.2, dass eine Anpassung der Attention-Schicht in Kombination mit dem FFN die höchste Leistungsfähigkeit erbringt und das Experiment-Setup aus Zhou et al. [74] erweitert.

Die Ergebnisse auf den ETT-Datensätzen zeigen zudem, dass das Fine-Tuning der Add & LN-Komponente nur einen geringfügigen Einfluss auf die Imputationsleistung hat. Dieses Verhalten ist allerdings auf den zwei weiteren Datensätzen (Electricity, Weather) nicht zu beobachten. Wie bereits beschrieben, sind die Ergebnisse bzgl. des Einflusses der Komponente hier nicht eindeutig. Andere Paper, welche das Fine-Tuning von LLMs auf weiterführende NLP-Aufgaben untersuchen, zeigen, dass die ausschließliche Anpassung

der Add & LN- Komponente effektive Ergebnisse ermöglichen kann [71]. Dieses Verhalten kann in den durchgeführten Versuchen nicht beobachtet werden.

Insgesamt zeigen die Ergebnisse im Bezug auf die zweite Forschungsfrage, dass im Allgemeinen die Attention-Schicht und das FFN angepasst werden sollten. Dies hat zur Folge, dass ein Großteil des gesamten Modells optimiert werden muss und im Grunde ein komplettes Fine-Tuning des Modells angestrebt wird. Entgegen der Zielsetzung, bei einer Adaption eines LLMs auf eine neue Domäne möglichst wenige Parameter anzupassen, legen die beobachteten Ergebnisse nahe, dass ein umfangreiches Fine-Tuning des gesamten Modells die besten Ergebnisse liefert. Damit steigen die Ressourcenanforderungen für diesen Prozess allerdings auch deutlich. Die Beobachtungen der Ergebnisse entsprechen generell den Erwartungen aus anderen Domänen und Anwendungsfällen und können auch für die Optimierung von LLMs auf die Imputation bestätigt werden.

RQ3: Welchen Einfluss hat die Wahl der Optimierungsmethode, die für die Anpassung auf nachgelagerte NLP-Aufgaben entwickelt wurden, auf die Anwendbarkeit von LLMs bei der Imputation?

Zur Untersuchung dieser Forschungsfrage wurden unterschiedliche Optimierungsmethoden angewendet und die Ergebnisse lassen folgende Schlussfolgerungen zu. Die führende Leistungsfähigkeit der Modelle bei Wahl der LoRA-Methode bestätigt die aktuelle Anwendung als weit verbreitetes Verfahren zum Fine-Tuning von LLMs. Diese Dominanz ist bei fast allen Datensätzen zu beobachten, sodass hier ein sehr einheitliches Ergebnisbild zu betrachten ist. Die geringere Leistungsfähigkeit der (IA^3)-Methode lässt sich mit der geringen Anzahl an hinzugefügten Parametern begründen. Aufgrund der Funktionsweise von (IA^3) werden lediglich einzelne Skalierungsvektoren in die Komponenten des LLMs integriert. In diesem Kontext zeigt sich, dass die anderen Methoden, welche mehr optimierbare Parameter hinzufügen, bessere Leistungen erbringen. Die beobachteten Ergebnisse waren daher zu erwarten. Andere Untersuchung [30, 50] bestätigen ebenfalls die führende Rolle von LoRA. Daher wird deutlich, dass die PEFT-Methoden für die Anpassung auf die Imputationsaufgabe geeignet sind. Das verwendete Framework verarbeitet die Zeitreihendaten bereits vor der Übergabe an die Sprachmodelle, sodass hier kein Unterschied zum Fine-Tuning zu nachgelagerten NLP-Tasks zu beobachten ist.

Zusammenfassend offenbart die Analyse des Leistungsvergleichs verschiedener PEFT-Methoden, dass LoRA im betrachteten Imputationssetup die beste Performance zeigt. Diese Beobachtung steht im Einklang mit den Erwartungen, angesichts der prominenten

Anwendung von LoRA bei der Optimierung von LLMs. Abhängig vom Bedarf an ressourceneffizienter Optimierung stellen jedoch auch andere Ansätze, wie bspw. Qlora oder (IA^3), geeignete Alternativen dar.

Nach der Diskussion der einzelnen Untersuchungsergebnisse werden im folgenden Abschnitt die Limitationen und Grenzen dieser Arbeit beschrieben. Zu einer dieser Limitationen gehört die Tatsache, dass im Rahmen der gegebenen Ressourcen keine Betrachtung von den größten frei verfügbaren LLMs möglich ist. Modelle mit mehreren Milliarden von Parametern können zum aktuellen Zeitpunkt nur auf überdurchschnittlich leistungsfähigen Grafikkarten optimiert werden. Das ressourcenintensive Setting dieser Arbeit zeigt sich zudem in der notwendigen Anwendung von den Fine-Tuning-Methoden. Einige der ausgewählten Modelle verlangen die Verwendung einer PEFT-Methode, da ein klassisches Fine-Tuning der Attention-Blöcke bereits zu Engpässen führen konnte. Aufgrund dieser Beschränkung war bspw. eine Betrachtung des Llama2-Modells im Rahmen der zweiten Forschungsfrage nicht möglich. Diese Limitationen wurden dahingehend umgangen, dass die größtmöglichen Modelle in die Vergleiche aufgenommen wurden. Zudem sind Sprachmodelle mit verschiedenen Grundkonzepten des Pre-Trainings verglichen worden, damit die Ergebnisse auf größere Modelle mit gleichen Grundkonzepten übertragen werden können. Insgesamt heißt dies, dass die Beschränkung der Rechenressourcen zu einem Setup leitet, dass insgesamt die bestmögliche Auswahl an Sprachmodellen und eine Übertragbarkeit auf andere Modelle ermöglicht.

Im Mittelpunkt dieser Arbeit steht explizit die Imputation von Zeitreihendaten. Andere Zeitreihenaufgaben sind daher nicht Gegenstand der Betrachtung und sollten in weiterführenden Arbeiten ebenso mit dem notwendigen Detailgrad betrachtet werden. Daher unterscheiden sich die Beobachtungen auch mit denen aus vorangegangenen Arbeiten, wie z.B. GPT4TS [74], da in diesen Arbeiten die Konzeption eines Foundation-Modells im Mittelpunkt stand, welches übergreifende Zeitreihenaufgaben behandeln kann.

Die Ergebnisse dieser Studie legen nahe, dass weiterführende Untersuchungen anderer Aspekte von Interesse sind. Ein zentraler Aspekt betrifft die Reproduktion der hier dargelegten Erkenntnisse anhand größerer Sprachmodelle, wie z. B. Llama2 mit 70 Milliarden Parametern, um die Beobachtungen zu bestätigen. Dabei sollte insbesondere ein vollständiges Fine-Tuning des Modells im Fokus stehen, wie es die Analyse der Optimierung einzelner Modellkomponenten nahelegt. Darüber hinaus erscheint es angesichts der kontinuierlichen Entwicklung neuer LLMs, die State-of-the-Art-Ergebnisse erzielen, sinnvoll, diese in zukünftigen Forschungsprojekten zu berücksichtigen (bspw. Gemma von

Google). Weiterhin sind andere Datensätze zu untersuchen. Obwohl die in dieser Arbeit verwendeten Datensätze auf dem Vorgehen früherer Studien basieren, ist die Bandbreite an Zeitreihendatensätzen damit nicht vollständig erfasst. Insbesondere Datensätze aus unterschiedlichen Domänen könnten neue Einblicke gewähren und sollten daher in zukünftigen Analysen berücksichtigt werden.

6 Fazit

Die Imputation von unvollständigen Zeitreihen ist eine herausfordernde aber zugleich sehr relevante Aufgabe, um nachgelagerte Analysetätigkeiten mit validen Daten zu ermöglichen. Aufgrund der hohen Relevanz entwickelten sich eine Reihe fortgeschrittener Methoden, die auf Grundlage der Transformer-Architektur State-of-the-Art Ergebnisse liefern. Die Erfolge von großen Sprachmodellen im Bereich des NLPs motivierte die Anwendung dieser Modelle in der Zeitreihendomäne. In dieser Arbeit wurden die LLMs zur Imputation von Zeitreihendaten erfolgreich angewendet. Die Ergebnisse zeigen, dass die Sprachmodelle als zentrale Komponente innerhalb des Prozesses der Imputation dienen können und dabei eine vergleichbare Leistungsfähigkeit gegenüber State-of-the-Art Modellen zeigen. Der Vergleich unterschiedlicher Sprachmodelle verdeutlicht zudem, dass die Imputationsfähigkeit eines Modells zum einen von der Anzahl der Parameter und zum anderen von der Art des Pre-Trainings der Modelle abhängig ist. Die großen Sprachmodelle, in dieser Arbeit Llama2 und Phi-2, haben generell eine hohe Leistungsfähigkeit in einem Großteil der Experimente und übertreffen die kleineren Modelle. Es zeigt sich allerdings auch, dass kleinere Modelle ebenfalls sehr gute Ergebnisse erbringen, wenn diese auf dem Ansatz des Denoising Autoencodings trainiert wurden. Bei einer deutlich geringeren Parameteranzahl haben BERT und T5 bei einigen Experimenten eine führende Leistungsfähigkeit gezeigt. Das Denoising Autoencoding kommt der Aufgabe der Imputation von Zeitreihen sehr nahe, sodass die genannten Modelle auf eine solche Art von Aufgabe spezialisiert sind. Schlussfolgernd lässt sich sagen, dass spezialisierte Modelle mit geringer Anzahl von Parametern ebenso gute Ergebnisse erbringen können, wie die LLMs mit einer Parameteranzahl im Bereich der Billionen. Diese Beobachtung im Rahmen der Experimente stellt einen wichtigen Beitrag für folgende Forschungen dar. Im Rahmen des Fine-Tunings der unterschiedlichen Modellkomponenten und verschiedener Methoden konnte gezeigt werden, dass die PEFT-Methoden zur Anpassung der LLMs im Bezug zur Imputation anwendbar sind und vor allem der LoRA-Ansatz gute Ergebnisse liefert. Dies deckt sich mit der aktuellen Dominanz dieser Methode im Fine-Tuning von Sprachmodellen für weiterführende NLP-Aufgaben. Zudem wird deutlich, dass die

Attention-Schicht sowie das FFN Gegenstand von Fine-Tuning Prozessen sein sollten. Diese Kombination der Modellkomponenten führte in den Experimenten zu den besten Ergebnissen.

Neben den hier dargestellten Beobachtungen bleiben allerdings auch Fragen offen, die in einer weiterführenden Arbeit mit angepasster Zielsetzung behandelt werden sollten. Dazu gehört die intensive Betrachtung der hier gesammelten Erkenntnisse im Kontext anderer Zeitreihenaufgaben. Es gilt zu untersuchen, inwieweit die unterschiedlichen Pre-Training Ansätze der verschiedenen LLMs eine Auswirkung auf andere Aufgaben, wie z.B. die Vorhersage von Zeitreihen haben. Darüber hinaus sind die sehr hohen Anforderungen der größten Sprachmodelle, wie z.B. Llama2 mit 70 Billionen Parametern, ein Hindernis in der weiteren Untersuchung noch komplexerer Modelle als in dieser Arbeit. Bei Zugriff auf umfangreiche Ressourcen ist die Eignung von noch größeren Modellen zur Zeitreihenimputation zu untersuchen. Zum jetzigen Zeitpunkt der Veröffentlichung gibt es bereits Nachfolger-Modelle von Llama2 und Phi-2, die aufgrund ihrer Neuartigkeit nicht in die Arbeit aufgenommen werden konnten. Daher ist es wichtig, die hier gesammelten Erkenntnisse mit den neuesten Modellen zu bestätigen.

Insgesamt lässt sich zusammenfassen, dass diese Arbeit einen wichtigen Beitrag zur Zeitreihenimputation mit Hilfe von LLMs erbringt. Die gezielte Untersuchung verschiedener Sprachmodelle verdeutlicht, dass die Modelle im Allgemeinen zur Imputation von Zeitreihen geeignet sind und eine gute Grundlage bieten. Es bleibt abzuwarten, ob Foundation Modelle entwickelt werden, die spezifisch auf alle Zeitreihenaufgaben trainiert werden und gleiche Erfolge wie die Sprachmodelle auf NLP-Aufgaben mit sich bringen.

Literaturverzeichnis

[1] Aggarwal, Karan ; Srivastava, Jaideep: *Filling out the missing gaps: Time Series Imputation with Semi-Supervised Learning.* 2023

[2] Ahmed, Sabeen ; Nielsen, Ian E. ; Tripathi, Aakash ; Siddiqui, Shamoon ; Ramachandran, Ravi P. ; Rasool, Ghulam: Transformers in Time-Series Analysis: A Tutorial. In: *Circuits, Systems, and Signal Processing* 42 (2023), Juli, Nr. 12, S. 7433–7466. – ISSN 1531-5878

[3] Ahn, Hyun ; Sun, Kyunghee ; Pio Kim, Kwanghoon: Comparison of Missing Data Imputation Methods in Time Series Forecasting. In: *Computers, Materials and Continua* 70 (2022), Nr. 1, S. 767–779. – ISSN 1546-2226

[4] Ainslie, Joshua ; Lee-Thorp, James ; Jong, Michiel de ; Zemlyanskiy, Yury ; Lebron, Federico ; Sanghai, Sumit: GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2023, S. 4895–4901

[5] Amidi, Afshine ; Amidi, Shervine: *Recurrent Neural Networks Cheatsheet.* 2019. – URL https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks. – [18.01.2024]

[6] Askell, Amanda ; Bai, Yuntao ; Chen, Anna ; Drain, Dawn ; Ganguli, Deep ; Henighan, Tom ; Jones, Andy ; Joseph, Nicholas ; Mann, Ben ; DasSarma, Nova ; Elhage, Nelson ; Hatfield-Dodds, Zac ; Hernandez, Danny ; Kernion, Jackson ; Ndousse, Kamal ; Olsson, Catherine ; Amodei, Dario ; Brown, Tom ; Clark, Jack ; McCandlish, Sam ; Olah, Chris ; Kaplan, Jared: *A General Language Assistant as a Laboratory for Alignment.* 2021

[7] Backhaus, Klaus ; Erichson, Bernd ; Plinke, Wulff ; Weiber, Rolf: *Multivariate Analysemethoden: Eine anwendungsorientierte Einführung.* Springer Berlin Heidelberg, 2018. – ISBN 9783662566558

[8] Bahdanau, Dzmitry ; Cho, Kyunghyun ; Bengio, Yoshua: Neural Machine Translation by Jointly Learning to Align and Translate. In: *CoRR* (2014)

[9] Börse, Frankfurt: *Historische Kurse und Umsätze DAX*. 2024. – URL https://www.boerse-frankfurt.de/index/dax/kurshistorie/historische-kurse-und-umsaetze. – [05.01.2024]

[10] Cao, Defu ; Jia, Furong ; Arik, Sercan O. ; Pfister, Tomas ; Zheng, Yixiang ; Ye, Wen ; Liu, Yan: TEMPO: Prompt-based Generative Pre-trained Transformer for Time Series Forecasting. In: *The Twelfth International Conference on Learning Representations*, 2024

[11] Cao, Wei ; Wang, Dong ; Li, Jian ; Zhou, Hao ; Li, Lei ; Li, Yitan: BRITS: Bidirectional Recurrent Imputation for Time Series. In: *Advances in Neural Information Processing Systems* Bd. 31, Curran Associates, Inc., 2018, S. 6776–6786

[12] Chang, Ching ; Peng, Wen-Chih ; Chen, Tien-Fu: *LLM4TS: Aligning Pre-Trained LLMs as Data-Efficient Time-Series Forecasters*. 2023

[13] Chang, Yupeng ; Wang, Xu ; Wang, Jindong ; Wu, Yuan ; Yang, Linyi ; Zhu, Kaijie ; Chen, Hao ; Yi, Xiaoyuan ; Wang, Cunxiang ; Wang, Yidong ; Ye, Wei ; Zhang, Yue ; Chang, Yi ; Yu, Philip S. ; Yang, Qiang ; Xie, Xing: A Survey on Evaluation of Large Language Models. In: *ACM Transactions on Intelligent Systems and Technology* (2024), Januar. – ISSN 2157-6912

[14] Che, Zhengping ; Purushotham, Sanjay ; Cho, Kyunghyun ; Sontag, David ; Liu, Yan: Recurrent Neural Networks for Multivariate Time Series with Missing Values. In: *Scientific Reports* 8 (2018), April, Nr. 1. – ISSN 2045-2322

[15] Chung, Junyoung ; Gulcehre, Caglar ; Cho, KyungHyun ; Bengio, Yoshua: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. In: *NIPS 2014 Workshop on Deep Learning*, Dezember 2014

[16] Dettmers, Tim ; Pagnoni, Artidoro ; Holtzman, Ari ; Zettlemoyer, Luke: *QLoRA: Efficient Finetuning of Quantized LLMs*. 2023

[17] Devlin, Jacob ; Chang, Ming-Wei ; Lee, Kenton ; Toutanova, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *North American Chapter of the Association for Computational Linguistics*, Association for Computational Linguistics, 2019, S. 4171–4186

[18] Donders, A. Rogier T. ; Heijden, Geert J. van der ; Stijnen, Theo ; Moons, Karel G.: Review: A gentle introduction to imputation of missing values. In: *Journal of Clinical Epidemiology* 59 (2006), oct, Nr. 10, S. 1087–1091

[19] Du, Wenjie ; Côté, David ; Liu, Yan: SAITS: Self-attention-based imputation for time series. In: *Expert Systems with Applications* 219 (2023), jun, S. 119619

[20] Fang, Chenguang ; Wang, Chen: *Time Series Data Imputation: A Survey on Deep Learning Approaches*. 2020

[21] Gers, Felix A. ; Schmidhuber, Jürgen ; Cummins, Fred: Learning to Forget: Continual Prediction with LSTM. In: *Neural Computation* 12 (2000), Oktober, Nr. 10, S. 2451–2471. – ISSN 1530-888X

[22] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016. – URL http://www.deeplearningbook.org. – [16.03.2024]

[23] Gruver, Nate ; Finzi, Marc ; Qiu, Shikai ; Wilson, Andrew G.: Large Language Models Are Zero-Shot Time Series Forecasters. In: *Thirty-seventh Conference on Neural Information Processing Systems*, 2023

[24] Gunasekar, Suriya ; Zhang, Yi ; Aneja, Jyoti ; Mendes, Caio César T. ; Del Giorno, Allie ; Gopi, Sivakanth ; Javaheripi, Mojan ; Kauffmann, Piero ; Rosa, Gustavo de ; Saarikivi, Olli ; Salim, Adil ; Shah, Shital ; Behl, Harkirat S. ; Wang, Xin ; Bubeck, Sébastien ; Eldan, Ronen ; Kalai, Adam T. ; Lee, Yin T. ; Li, Yuanzhi: *Textbooks Are All You Need*. 2023

[25] Guresen, Erkam ; Kayakutlu, Gulgun: Definition of artificial neural networks with comparison to other networks. In: *Procedia Computer Science* 3 (2011), S. 426–433. – ISSN 1877-0509

[26] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Juni 2016, S. 770–778

[27] Hewamalage, Hansika ; Bergmeir, Christoph ; Bandara, Kasun: Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. In: *International Journal of Forecasting* 37 (2021), Januar, Nr. 1, S. 388–427. – ISSN 0169-2070

[28] Houlsby, Neil ; Giurgiu, Andrei ; Jastrzebski, Stanislaw ; Morrone, Bruna ; Laroussilhe, Quentin de ; Gesmundo, Andrea ; Attariyan, Mona ; Gelly, Sylvain: Parameter-Efficient Transfer Learning for NLP. In: *International conference on machine learning*, 2019, S. 2790–2799

[29] Hu, Edward J. ; Shen, Yelong ; Wallis, Phillip ; Allen-Zhu, Zeyuan ; Li, Yuanzhi ; Wang, Shean ; Wang, Lu ; Chen, Weizhu: LoRA: Low-Rank Adaptation of Large Language Models. In: *International Conference on Learning Representations*, 2022

[30] Hu, Zhiqiang ; Wang, Lei ; Lan, Yihuai ; Xu, Wanyu ; Lim, Ee-Peng ; Bing, Lidong ; Xu, Xing ; Poria, Soujanya ; Lee, Roy: LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Dezember 2023, S. 5254–5276

[31] Jain, Aditya ; Kulkarni, Gandhar ; Shah, Vraj: Natural Language Processing. In: *International Journal of Computer Sciences and Engineering* 6 (2018), Januar, Nr. 1, S. 161–167. – ISSN 2347-2693

[32] Javaheripi, Mojan ; Bubeck, Sébastien: *Phi-2: The surprising power of small language models.* Dezember 2023. – URL https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/. – [19.02.2024]

[33] Jin, Ming ; Koh, Huan Y. ; Wen, Qingsong ; Zambon, Daniele ; Alippi, Cesare ; Webb, Geoffrey I. ; King, Irwin ; Pan, Shirui: *A Survey on Graph Neural Networks for Time Series: Forecasting, Classification, Imputation, and Anomaly Detection.* 2023

[34] Jin, Ming ; Wang, Shiyu ; Ma, Lintao ; Chu, Zhixuan ; Zhang, James Y. ; Shi, Xiaoming ; Chen, Pin-Yu ; Liang, Yuxuan ; Li, Yuan-Fang ; Pan, Shirui ; Wen, Qingsong: Time-LLM: Time Series Forecasting by Reprogramming Large Language Models. In: *International Conference on Learning Representations (ICLR)*, 2024

[35] Jin, Ming ; Wen, Qingsong ; Liang, Yuxuan ; Zhang, Chaoli ; Xue, Siqiao ; Wang, Xue ; Zhang, James ; Wang, Yi ; Chen, Haifeng ; Li, Xiaoli ; Pan, Shirui ; Tseng, Vincent S. ; Zheng, Yu ; Chen, Lei ; Xiong, Hui: Large Models for Time Series and Spatio-Temporal Data: A Survey and Outlook. In: *CoRR* (2023)

[36] Kazijevs, Maksims ; Samad, Manar D.: Deep imputation of missing values in time series health data: A review with benchmarking. In: *Journal of Biomedical Informatics* 144 (2023), aug, S. 104440

[37] Kim, Jinhee ; Kim, Taesung ; Choi, Jang-Ho ; Choo, Jaegul: End-to-end Multi-task Learning of Missing Value Imputation and Forecasting in Time-Series Data. In: *2020 25th International Conference on Pattern Recognition (ICPR)*, IEEE, jan 2021

[38] Lewis, Mike ; Liu, Yinhan ; Goyal, Naman ; Ghazvininejad, Marjan ; Mohamed, Abdel rahman ; Levy, Omer ; Stoyanov, Veselin ; Zettlemoyer, Luke: BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, Juli 2020, S. 7871–7880

[39] Li, Xiang L. ; Liang, Percy: Prefix-Tuning: Optimizing Continuous Prompts for Generation. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, Association for Computational Linguistics, August 2021, S. 4582–4597

[40] Lim, Bryan ; Zohren, Stefan: Time-series forecasting with deep learning: a survey. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 379 (2021), Februar, Nr. 2194, S. 20200209. – ISSN 1471-2962

[41] Lin, Tianyang ; Wang, Yuxin ; Liu, Xiangyang ; Qiu, Xipeng: A survey of transformers. In: *AI Open* 3 (2022), S. 111–132. – ISSN 2666-6510

[42] Little, Roderick ; Rubin, Donald: *Statistical Analysis with Missing Data, Third Edition*. Wiley, April 2019. – ISBN 9781119482260

[43] Liu, Haokun ; Tam, Derek ; Mohammed, Muqeeth ; Mohta, Jay ; Huang, Tenghao ; Bansal, Mohit ; Raffel, Colin: Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning. In: *Advances in Neural Information Processing Systems* Bd. 35, 2022, S. 1950–1965

[44] Lou, Renze ; Zhang, Kai ; Yin, Wenpeng: *A Comprehensive Survey on Instruction Following*. 2023

[45] Mangrulkar, Sourab ; Gugger, Sylvain ; Debut, Lysandre ; Belkada, Younes ; Paul, Sayak ; Bossan, Benjamin: *PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods*. https://github.com/huggingface/peft. 2022

[46] Niu, Zhaoyang ; Zhong, Guoqiang ; Yu, Hui: A review on the attention mechanism of deep learning. In: *Neurocomputing* 452 (2021), September, S. 48–62. – ISSN 0925-2312

[47] OpenAI: *GPT-4 Technical Report*. 2023

[48] Osman, Muhammad S. ; Abu-Mahfouz, Adnan M. ; Page, Philip R.: A Survey on Data Imputation Techniques: Water Distribution System as a Use Case. In: *IEEE Access* 6 (2018), S. 63279–63291

[49] Pratama, Irfan ; Permanasari, Adhistya E. ; Ardiyanto, Igi ; Indrayani, Rini: A review of missing values handling methods on time-series data. In: *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, IEEE, Oktober 2016, S. 1–6

[50] Pu, George ; Jain, Anirudh ; Yin, Jihan ; Kaplan, Russell: Empirical Analysis of the Strengths and Weaknesses of PEFT Techniques for LLMs. In: *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2023

[51] Radford, Alec ; Wu, Jeff ; Child, Rewon ; Luan, David ; Amodei, Dario ; Sutskever, Ilya: Language Models are Unsupervised Multitask Learners. (2019)

[52] Raffel, Colin ; Shazeer, Noam ; Roberts, Adam ; Lee, Katherine ; Narang, Sharan ; Matena, Michael ; Zhou, Yanqi ; Li, Wei ; Liu, Peter J.: Exploring the limits of transfer learning with a unified text-to-text transformer. In: *J. Mach. Learn. Res.* 21 (2020), jan, Nr. 1. – ISSN 1532-4435

[53] Saad, Muhammad ; Chaudhary, Mohita ; Karray, Fakhri ; Gaudet, Vincent: Machine Learning Based Approaches for Imputation in Time Series Data and their Impact on Forecasting. In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, oct 2020, S. 2621–2627

[54] Sarker, Iqbal H.: Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. In: *SN Computer Science* 2 (2021), August, Nr. 6. – ISSN 2661-8907

[55] Shazeer, Noam: *Fast Transformer Decoding: One Write-Head is All You Need.* 2019

[56] Singh, Sushant ; Mahmood, Ausif: The NLP Cookbook: Modern Recipes for Transformer Based Deep Learning Architectures. In: *IEEE Access* 9 (2021), S. 68675–68702. – ISSN 2169-3536

[57] Sutskever, Ilya ; Vinyals, Oriol ; Le, Quoc V.: Sequence to Sequence Learning with Neural Networks. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2.* Cambridge, MA, USA : MIT Press, 2014 (NIPS'14), S. 3104–3112

[58] Tawakuli, Amal ; Kaiser, Daniel ; Engel, Thomas: Experience: Differentiating Between Isolated and Sequence Missing Data. In: *Journal of Data and Information Quality* 15 (2023), jun, Nr. 2, S. 1–15

[59] Tay, Yi ; Wei, Jason ; Chung, Hyung W. ; Tran, Vinh Q. ; So, David R. ; Shakeri, Siamak ; Garcia, Xavier ; Zheng, Huaixiu S. ; Rao, Jinfeng ; Chowdhery, Aakanksha ; Zhou, Denny ; Metzler, Donald ; Petrov, Slav ; Houlsby, Neil ; Le, Quoc V. ; Dehghani, Mostafa: *Transcending Scaling Laws with 0.1 Extra Compute.* 2022

[60] Touvron, Hugo ; Martin, Louis ; Stone, Kevin ; Albert, Peter ; Almahairi, Amjad ; Babaei, Yasmine ; Bashlykov, Nikolay ; Batra, Soumya ; Bhargava, Prajjwal ; Bhosale, Shruti ; Bikel, Dan ; Blecher, Lukas ; Ferrer, Cristian C. ; Chen, Moya ; Cucurull, Guillem ; Esiobu, David ; Fernandes, Jude ; Fu, Jeremy ; Fu, Wenyin ; Fuller, Brian ; Gao, Cynthia ; Goswami, Vedanuj ; Goyal, Naman ; Hartshorn, Anthony ; Hosseini, Saghar ; Hou, Rui ; Inan, Hakan ; Kardas, Marcin ; Kerkez, Viktor ; Khabsa, Madian ; Kloumann, Isabel ; Korenev, Artem ; Koura, Punit S. ; Lachaux, Marie-Anne ; Lavril, Thibaut ; Lee, Jenya ; Liskovich, Diana ; Lu, Yinghai ; Mao, Yuning ; Martinet, Xavier ; Mihaylov, Todor ; Mishra, Pushkar ; Molybog, Igor ; Nie, Yixin ; Poulton, Andrew ; Reizenstein, Jeremy ; Rungta, Rashi ; Saladi, Kalyan ; Schelten, Alan ; Silva, Ruan ; Smith, Eric M. ; Subramanian, Ranjan ; Tan, Xiaoqing E. ; Tang, Binh ; Taylor, Ross ; Williams, Adina ; Kuan, Jian X. ; Xu, Puxin ; Yan, Zheng ; Zarov, Iliyan ; Zhang, Yuchen ; Fan, Angela ; Kambadur, Melanie ; Narang, Sharan ; Rodriguez, Aurelien ; Stojnic, Robert ; Edunov, Sergey ; Scialom, Thomas: *Llama 2: Open Foundation and Fine-Tuned Chat Models.* 2023

[61] Van Houdt, Greg ; Mosquera, Carlos ; Nápoles, Gonzalo: A review on the long short-term memory model. In: *Artificial Intelligence Review* 53 (2020), Mai, Nr. 8, S. 5929–5955. – ISSN 1573-7462

[62] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Lukasz ; Polosukhin, Illia: Attention Is All You Need. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems* Bd. 30, Curran Associates Inc., 2017, S. 6000–6010

[63] Verbesselt, Jan ; Hyndman, Rob ; Newnham, Glenn ; Culvenor, Darius: Detecting trend and seasonal changes in satellite image time series. In: *Remote Sensing of Environment* 114 (2010), Januar, Nr. 1, S. 106–115. – ISSN 0034-4257

[64] Wan, Zhongwei ; Wang, Xin ; Liu, Che ; Alam, Samiul ; Zheng, Yu ; Liu, Jiachen ; Qu, Zhongnan ; Yan, Shen ; Zhu, Yi ; Zhang, Quanlu ; Chowdhury, Mosharaf ; Zhang, Mi: Efficient Large Language Models: A Survey. (2023)

[65] Wang, Thomas ; Roberts, Adam ; Hesslow, Daniel ; Scao, Teven L. ; Chung, Hyung W. ; Beltagy, Iz ; Launay, Julien ; Raffel, Colin: What Language Model Architecture and Pretraining Objective Work Best for Zero-Shot Generalization? In: *Proceedings of the 39th International Conference on Machine Learning* Bd. 162, PMLR, Juli 2022, S. 22964–22984

[66] Wu, Haixu ; Hu, Tengge ; Liu, Yong ; Zhou, Hang ; Wang, Jianmin ; Long, Mingsheng: TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis. In: *International Conference on Learning Representations*, 2023

[67] Xiong, Ruibin ; Yang, Yunchang ; He, Di ; Zheng, Kai ; Zheng, Shuxin ; Xing, Chen ; Zhang, Huishuai ; Lan, Yanyan ; Wang, Liwei ; Liu, Tie-Yan: On layer normalization in the transformer architecture. In: *Proceedings of the 37th International Conference on Machine Learning*, JMLR.org, 2020 (ICML'20 975)

[68] Zaheer, Manzil ; Guruganesh, Guru ; Dubey, Kumar A. ; Ainslie, Joshua ; Alberti, Chris ; Ontanon, Santiago ; Pham, Philip ; Ravula, Anirudh ; Wang, Qifan ; Yang, Li u. a.: Big bird: Transformers for longer sequences. In: *Advances in Neural Information Processing Systems* Bd. 33, 2020, S. 17283–17297

[69] Zeng, Aohan ; Liu, Xiao ; Du, Zhengxiao ; Wang, Zihan ; Lai, Hanyu ; Ding, Ming ; Yang, Zhuoyi ; Xu, Yifan ; Zheng, Wendi ; Xia, Xiao ; Tam, Weng L. ; Ma, Zixuan ; Xue, Yufei ; Zhai, Jidong ; Chen, Wenguang ; Zhang, Peng ; Dong,

Yuxiao ; Tang, Jie: GLM-130B: An Open Bilingual Pre-trained Model. In: *The Eleventh International Conference on Learning Representations*, 2022

[70] Zhang, Qingru ; Chen, Minshuo ; Bukharin, Alexander ; He, Pengcheng ; Cheng, Yu ; Chen, Weizhu ; Zhao, Tuo: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning. In: *The Eleventh International Conference on Learning Representations*, 2023

[71] Zhao, Bingchen ; Tu, Haoqin ; Wei, Chen ; Mei, Jieru ; Xie, Cihang: Tuning LayerNorm in Attention: Towards Efficient Multi-Modal LLM Finetuning. In: *The Twelfth International Conference on Learning Representations*, arXiv, 2023

[72] Zhao, Wayne X. ; Zhou, Kun ; Li, Junyi ; Tang, Tianyi ; Wang, Xiaolei ; Hou, Yupeng ; Min, Yingqian ; Zhang, Beichen ; Zhang, Junjie ; Dong, Zican ; Du, Yifan ; Yang, Chen ; Chen, Yushuo ; Chen, Zhipeng ; Jiang, Jinhao ; Ren, Ruiyang ; Li, Yifan ; Tang, Xinyu ; Liu, Zikang ; Liu, Peiyu ; Nie, Jian-Yun ; Wen, Ji-Rong: *A Survey of Large Language Models*. 2023

[73] Zhou, Haoyi ; Zhang, Shanghang ; Peng, Jieqi ; Zhang, Shuai ; Li, Jianxin ; Xiong, Hui ; Zhang, Wancai: Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In: *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference* Bd. 35, AAAI Press, 2021, S. 11106–11115

[74] Zhou, Tian ; Niu, PeiSong ; Wang, Xue ; Sun, Liang ; Jin, Rong: One Fits All: Power General Time Series Analysis by Pretrained LM. In: *Thirty-seventh Conference on Neural Information Processing Systems*, 2023

A Anhang

A.1 Zusammenfassung Baseline Modelle

Modell	**Architektur**	**Pre-Training Task**	**Parameter (in Millionen)**	**Layer**	**Attention Heads**	**Modell-Dimension**
BART	Encoder, Decoder	Denoising Autoencoding	140	Jeweils 6		768
BERT	Encoder Only	Masked Language Modeling	110	12	12	768
GPT-2	Decoder Only	Language Modeling	117	12	12	768
T5	Encoder, Decoder	Denoising Autoencoding	220	12	12	768
Llama2	Decoder Only	Language Modeling	7.000	32	32	4096
Phi-2	Decoder Only	Language Modeling	2.000	24 (32)	32	2048 (2560)

A.2 Vollständige Ergebnisse (alle Modelle)

Modelle		GPT-2		BERT		Saits		TimesNet		Transformer		Llama2		Phi-2		T5		BART	
	Mask Ratio	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	12,5%	0,052	0,154	0,047	0,146	**0,028**	**0,112**	0,068	0,176	0,059	0,172	**0,041**	**0,138**	0,042	0,139	0,051	0,155	0,053	0,155
	25%	0,074	0,182	0,064	0,170	**0,039**	**0,132**	0,085	0,195	0,082	0,204	**0,052**	**0,153**	0,055	0,159	0,069	0,178	0,067	0,175
	37,5%	0,098	0,207	0,084	0,194	**0,047**	**0,143**	0,100	0,212	0,108	0,232	**0,069**	**0,175**	0,070	0,180	0,084	0,196	0,092	0,205
	50%	0,123	0,232	0,107	0,216	**0,064**	**0,168**	0,121	0,231	0,135	0,261	**0,093**	**0,204**	0,096	0,208	0,105	0,218	0,115	0,225
ETTh2	12,5%	0,041	0,131	**0,040**	**0,128**	0,063	0,171	0,041	0,136	0,168	0,299	0,041	0,131	0,041	**0,130**	**0,040**	0,131	0,041	0,132
	25%	0,050	0,146	**0,046**	**0,141**	0,061	0,172	0,048	0,147	0,228	0,348	0,048	0,143	**0,046**	**0,138**	0,047	0,144	0,047	0,143
	37,5%	0,057	0,158	**0,052**	0,152	0,091	0,211	0,054	0,156	0,287	0,391	0,053	0,152	0,052	**0,149**	**0,052**	**0,151**	0,053	0,153
	50%	0,067	0,172	0,061	0,165	0,108	0,232	0,062	0,167	0,311	0,410	0,061	0,163	0,061	**0,162**	**0,058**	**0,161**	**0,060**	0,164
ETTm1	12,5%	0,024	0,105	0,020	0,093	**0,015**	**0,082**	0,020	0,093	0,021	0,101	**0,019**	0,091	0,019	**0,090**	0,024	0,105	0,022	0,100
	25%	0,033	0,121	0,024	0,103	**0,016**	**0,084**	0,024	0,102	0,028	0,116	**0,023**	0,099	0,023	**0,098**	0,030	0,117	0,027	0,111
	37,5%	0,041	0,136	0,030	0,115	**0,018**	**0,088**	0,030	0,114	0,035	0,130	**0,029**	**0,109**	0,029	0,109	0,036	0,126	0,033	0,122
	50%	0,053	0,153	0,036	0,128	**0,029**	**0,114**	0,037	0,126	0,044	0,146	**0,036**	**0,121**	0,038	0,126	0,042	0,138	0,041	0,135
ETTm2	12,5%	0,021	0,087	**0,019**	**0,080**	0,023	0,097	**0,019**	**0,083**	0,150	0,284	0,021	0,087	0,023	0,090	0,020	0,087	0,020	0,086
	25%	0,025	0,097	**0,021**	**0,085**	0,030	0,117	**0,021**	**0,087**	0,157	0,282	0,024	0,092	0,023	0,091	0,023	0,092	0,023	0,093
	37,5%	0,028	0,104	**0,023**	**0,093**	0,046	0,147	**0,023**	**0,093**	0,226	0,345	0,025	0,095	0,027	0,100	0,025	0,098	0,025	0,097
	50%	0,031	0,111	**0,026**	**0,100**	0,056	0,166	**0,026**	**0,099**	0,222	0,347	0,029	0,105	0,030	0,107	0,028	0,105	0,028	0,105
Electricity	12,5%	0,072	0,186	0,071	0,186	0,146	0,272	0,089	0,204	0,147	0,276	**0,065**	**0,177**	**0,065**	**0,177**	0,080	0,198	0,078	0,193
	25%	0,080	0,196	0,079	0,196	0,161	0,281	0,097	0,211	0,159	0,283	**0,077**	**0,192**	**0,075**	**0,190**	0,087	0,205	0,084	0,200
	37,5%	0,089	0,207	0,087	0,205	0,172	0,288	0,098	0,215	0,165	0,287	**0,085**	**0,203**	**0,085**	**0,202**	0,093	0,213	0,091	0,209
	50%	0,096	0,216	0,096	0,216	0,181	0,295	0,104	0,222	0,179	0,297	**0,094**	**0,214**	**0,094**	**0,214**	0,100	0,222	0,099	0,218
Weather	12,5%	**0,026**	**0,048**	0,027	0,050	0,028	0,064	**0,026**	**0,048**	0,032	0,087	0,027	0,048	0,080	0,124	0,026	0,050	0,030	0,058
	25%	0,030	0,055	0,030	0,055	0,031	0,069	**0,029**	**0,053**	0,034	0,084	0,077	0,120	0,031	0,056	**0,029**	**0,054**	0,034	0,065
	37,5%	0,033	0,060	0,032	0,059	0,050	0,113	**0,032**	**0,058**	0,037	0,087	0,035	0,064	0,087	0,137	**0,032**	**0,059**	0,035	0,065
	50%	0,037	**0,067**	0,077	0,118	0,039	0,086	**0,036**	0,064	0,040	0,091	0,039	0,070	0,084	0,127	**0,035**	**0,063**	0,043	0,080
Count 1st		1	0	**4**	4	**8**	**8**	**4**	4	0	0	2	1	2	**5**	3	2	0	0
Count 2nd		0	2	3	2	0	0	**4**	3	0	0	**10**	**9**	3	**5**	3	3	1	0

*Legende: **Bester Wert**, **Zweitbester Wert**

A.3 Vollständige Ergebnisse Komponenten-Tuning

		Attention (1)		Add & LN (2)		FFN (3)		1 & 3		2 & 3		1 & 2		1 & 2 & 3	
	Mask Ratio	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	12,5%	0,0465	0,1459	0,0618	0,1635	0,0456	0,1465	**0,0416**	**0,1388**	0,0419	0,1397	0,0438	0,1409	**0,0410**	**0,1371**
	25%	0,0661	0,1728	0,0869	0,1956	0,0593	0,1643	**0,0540**	**0,1560**	**0,0540**	**0,1561**	0,0642	0,1706	0,0559	0,1577
	37,5%	0,0830	0,1946	0,1118	0,2205	0,0772	0,1867	**0,0722**	**0,1786**	**0,0748**	0,1837	0,0804	0,1910	0,0752	**0,1829**
	50%	0,1112	0,2237	0,1415	0,2460	0,1113	0,2207	**0,1074**	**0,2164**	0,1084	0,2176	0,1127	0,2234	**0,1043**	**0,2144**
ETTh2	12,5%	0,0410	0,1304	0,0440	0,1361	0,0417	0,1303	**0,0394**	**0,1263**	0,0405	0,1286	0,0406	0,1289	**0,0384**	**0,1249**
	25%	0,0465	0,1411	0,0519	0,1503	0,0469	0,1390	**0,0443**	**0,1341**	0,0460	0,1377	0,0462	0,1405	**0,0449**	**0,1346**
	37,5%	0,0551	0,1548	0,0593	0,1625	0,0542	0,1510	**0,0522**	**0,1468**	**0,0526**	**0,1483**	0,0536	0,1521	0,0526	0,1486
	50%	0,0627	0,1662	0,0700	0,1767	0,0610	0,1607	**0,0597**	**0,1588**	0,0605	0,1593	0,0621	0,1648	**0,0601**	**0,1590**
ETTm1	12,5%	0,0203	0,0941	0,0288	0,1133	0,0191	0,0916	**0,0172**	**0,0862**	0,0184	0,0893	0,0202	0,0944	**0,0176**	**0,0874**
	25%	0,0267	0,1084	0,0397	0,1322	0,0235	0,1003	**0,0223**	**0,0973**	0,0233	0,1000	0,0257	0,1066	**0,0227**	**0,0981**
	37,5%	0,0340	0,1221	0,0491	0,1471	0,0312	0,1135	**0,0295**	**0,1106**	0,0305	**0,1125**	0,0332	0,1209	**0,0304**	0,1128
	50%	0,0454	0,1405	0,0639	0,1660	0,0435	0,1328	**0,0407**	**0,1292**	0,0425	0,1315	0,0454	0,1406	**0,0416**	**0,1304**
ETTm2	12,5%	0,0200	0,0839	0,0220	0,0901	0,0193	0,0796	**0,0182**	**0,0773**	0,0187	0,0779	0,0193	0,0825	**0,0176**	**0,0760**
	25%	0,0233	0,0922	0,0259	0,0998	0,0216	0,0852	**0,0198**	**0,0815**	0,0209	0,0832	0,0226	0,0904	**0,0201**	**0,0820**
	37,5%	0,0258	0,0982	0,0297	0,1083	0,0238	0,0909	**0,0223**	**0,0877**	0,0235	0,0900	0,0255	0,0976	**0,0223**	**0,0875**
	50%	0,0300	0,1075	0,0338	0,1161	0,0266	0,0980	**0,0255**	**0,0952**	0,0267	0,0978	0,0291	0,1054	**0,0258**	**0,0959**
Electricity	12,5%	0,0729	**0,1869**	**0,0725**	0,1873	0,0778	0,1914	0,0815	0,1957	0,0796	0,1934	**0,0724**	**0,1863**	0,0801	0,1947
	25%	**0,0805**	**0,1963**	0,0815	0,1984	0,0860	0,2021	0,0872	0,2030	0,0871	0,2029	**0,0808**	**0,1964**	0,0876	0,2029
	37,5%	**0,0877**	**0,2050**	0,0895	0,2085	0,0941	0,2103	0,0933	0,2105	0,0937	0,2103	**0,0882**	**0,2050**	0,0948	0,2120
	50%	**0,0963**	**0,2154**	0,0982	0,2191	0,1003	0,2186	0,1008	0,2201	0,1000	0,2191	**0,0958**	**0,2146**	0,1018	0,2203
Weather	12,5%	0,0260	0,0467	0,0263	0,0476	0,0255	0,0457	**0,0253**	**0,0451**	0,0256	0,0469	0,0257	0,0468	**0,0252**	**0,0454**
	25%	0,0288	0,0526	0,0312	0,0577	0,0291	0,0537	0,0289	**0,0513**	**0,0285**	0,0517	0,0290	0,0529	**0,0280**	**0,0503**
	37,5%	0,0321	0,0582	0,0332	0,0605	0,0317	0,0565	**0,0316**	**0,0559**	**0,0314**	**0,0562**	0,0320	0,0580	0,0320	0,0575
	50%	0,0364	0,0645	0,0378	0,0668	**0,0357**	**0,0620**	0,0359	0,0630	**0,0355**	**0,0619**	0,0358	0,0637	0,0360	0,0629
	Count 1st	2	2	0	0	0	0	**11**	**13**	2	1	2	2	**7**	**6**
	Count 2nd	1	2	1	0	1	1	**7**	**6**	4	4	2	2	**8**	**9**

*Legende: **Bester Wert**, **Zweitbester Wert**

A.4 PEFT-Tuning Methodenvergleich

Methoden		LoRA		AdaLora		(IA^3)		Qlora	
	Mask Ratio	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	12,5%	**0,0524**	0,1542	0,0531	**0,1538**	0,0625	0,1649	**0,0522**	**0,1534**
	25%	**0,0738**	**0,1821**	0,0793	0,1863	0,0858	0,1940	**0,0729**	**0,1798**
	37,5%	**0,0979**	**0,2071**	0,1040	0,2123	0,1118	0,2207	**0,0996**	**0,2080**
	50%	**0,1228**	**0,2317**	0,1297	0,2368	0,1421	0,2472	**0,1291**	**0,2356**
ETTh2	12,5%	**0,0421**	**0,1315**	0,0428	0,1332	0,0444	0,1365	**0,0420**	**0,1320**
	25%	**0,0496**	**0,1465**	0,0498	0,1468	0,0525	0,1517	**0,0492**	**0,1454**
	37,5%	**0,0567**	**0,1578**	0,0579	0,1597	0,0599	0,1636	**0,0575**	**0,1596**
	50%	0,0666	**0,1717**	**0,0663**	**0,1716**	0,0688	0,1756	**0,0666**	0,1719
ETTm1	12,5%	**0,0244**	**0,1045**	0,0267	0,1096	0,0291	0,1138	**0,0238**	**0,1036**
	25%	**0,0329**	**0,1212**	0,0358	0,1263	0,0384	0,1306	**0,0328**	**0,1213**
	37,5%	**0,0413**	**0,1356**	0,0452	0,1411	0,0502	0,1481	**0,0413**	**0,1354**
	50%	**0,0533**	**0,1529**	0,0577	0,1585	0,0632	0,1650	**0,0526**	**0,1520**
ETTm2	12,5%	**0,0209**	**0,0872**	0,0222	0,0902	0,0226	0,0914	**0,0210**	**0,0870**
	25%	0,0250	**0,0970**	**0,0249**	0,0975	0,0261	0,1002	**0,0246**	**0,0965**
	37,5%	**0,0282**	**0,1044**	0,0285	0,1051	0,0296	0,1079	**0,0280**	**0,1039**
	50%	**0,0313**	**0,1112**	0,0322	0,1128	0,0339	0,1164	**0,0316**	**0,1111**
Electricity	12,5%	0,0716	0,1860	**0,0714**	**0,1858**	0,0723	0,1871	**0,0714**	**0,1859**
	25%	**0,0800**	**0,1963**	0,0807	0,1975	0,0817	0,1989	**0,0805**	**0,1970**
	37,5%	0,0886	0,2068	**0,0882**	**0,2065**	0,0898	0,2089	**0,0882**	**0,2065**
	50%	**0,0961**	**0,2162**	**0,0965**	0,2168	0,0984	0,2194	0,0966	**0,2167**
Weather	12,5%	**0,0260**	0,0476	0,0264	**0,0473**	**0,0261**	**0,0475**	0,0263	0,0478
	25%	**0,0297**	**0,0545**	**0,0294**	**0,0541**	0,0299	0,0551	0,0302	0,0553
	37,5%	**0,0330**	**0,0602**	**0,0332**	**0,0605**	0,0336	0,0614	0,0340	0,0618
	50%	**0,0373**	0,0669	**0,0373**	**0,0665**	0,0377	0,0668	0,0375	**0,0668**
Count 1st		**10**	**8**	**4**	6	0	0	**10**	**10**
Count 2nd		**10**	**11**	4	2	1	1	**9**	**10**

*Legende: **Bester Wert**, **Zweitbester Wert**

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Hamburg	25.04.2024	[redacted]
Ort	Datum	Unterschrift im Original