Sebastian Raschka

# Large Language Models selbst programmieren

Mit Python und PyTorch
ein eigenes LLM entwickeln

dpunkt.verlag

# Über dieses Buch

*Large Language Models selbst programmieren* wurde geschrieben, um Ihnen zu helfen, Ihre eigenen GPT-ähnlichen großen Sprachmodelle (*Large Language Models*, LLMs) von Grund auf zu verstehen und zu erstellen. Es beginnt bei den grundlegenden Arbeiten mit Textdaten und der Codierung von Attention-Mechanismen (Aufmerksamkeitsmechanismen) und führt Sie dann durch die Implementierung eines vollständigen GPT-Modells von Grund auf. Anschließend geht es um den Vortrainingsmechanismus sowie das Feintuning bei spezifischen Aufgaben wie Textklassifizierung und dem Befolgen von Anweisungen. Wenn Sie sich bis zum Ende des Buchs durchgearbeitet haben, werden Sie über profunde Kenntnisse zur Funktionsweise von LLMs verfügen und in der Lage sein, Ihre eigenen Modelle zu erstellen. Obwohl derartige Modelle im Vergleich zu den großen Grundlagenmodellen deutlich kleiner sind, verwenden sie dieselben Konzepte und dienen als leistungsstarke Lehrmittel, um die Kernmechanismen und -techniken zu verstehen, die in modernen LLMs zum Einsatz kommen.

## Wer das Buch lesen sollte

*Large Language Models selbst programmieren* richtet sich an Enthusiasten des Machine Learning, also Ingenieure, Forscherinnen, Studenten und Praktikerinnen, die ein tiefes Verständnis von der Funktionsweise von LLMs erlangen möchten und lernen wollen, ihre eigenen Modelle von Grund auf zu erstellen. Sowohl Einsteiger als auch erfahrene Entwicklerinnen werden in der Lage sein, ihre vorhandenen Fähigkeiten und Kenntnisse zu nutzen, um die Konzepte und Techniken zu verstehen, die für die Erstellung von LLMs relevant sind.

Von anderen Büchern hebt sich dieses Buch dadurch ab, dass es umfassend den gesamten Prozess der LLM-Erstellung abdeckt, von der Arbeit mit Datensätzen bis zur Implementierung der Modellarchitektur, dem Vortraining mit ungelabelten Daten und der Feinabstimmung oder Optimierung für spezifische Aufgaben. Zur Entstehungszeit dieses Buchs gab es keine andere Quelle, die einen so vollständigen und praxisnahen Ansatz zur Erstellung von LLMs von Grund auf bietet.

Um die Codebeispiele in diesem Buch zu verstehen, sollten Sie über solide Kenntnisse in der Python-Programmierung verfügen. Vorteilhaft kann es sein, wenn Sie mit Machine Learning, Deep Learning und künstlicher Intelligenz schon etwas vertraut sind, wobei aber ein umfassendes Hintergrundwissen in diesen Bereichen nicht erforderlich ist. LLMs sind ein einzigartiger Teilbereich der KI, sodass Sie auch dann, wenn Sie auf diesem Gebiet relativ neu sind, problemlos dem Buch folgen können.

Falls Sie bereits Erfahrung mit tiefen neuronalen Netzen (*Deep Neural Networks*) haben, sind Ihnen bestimmte Konzepte vielleicht schon vertraut, da LLMs auf diesen Architekturen aufbauen. Die Beherrschung von PyTorch ist jedoch keine Voraussetzung. Anhang A bietet eine kurze Einführung in PyTorch, die Sie mit den notwendigen Fähigkeiten ausstattet, um die Codebeispiele im Buch zu verstehen.

Als hilfreich können sich auch Kenntnisse in höherer Mathematik erweisen, insbesondere im Umgang mit Vektoren und Matrizen, wenn wir die Funktionsweise von LLMs erkunden. Darüber hinausgehendes mathematisches Wissen ist jedoch nicht erforderlich, um die im Buch vorgestellten Schlüsselkonzepte und Ideen zu verstehen.

Die wichtigste Voraussetzung ist eine solide Grundlage in der Python-Programmierung. Mit diesen Kenntnissen sind Sie gut gerüstet, um die faszinierende Welt der LLMs zu erkunden und die Konzepte sowie die Codebeispiele im Buch in eigenen Projekten umsetzen zu können.

## Wie das Buch aufgebaut ist: ein Wegweiser

Dieses Buch ist so konzipiert, dass Sie es sequenziell lesen sollten, da jedes Kapitel auf den Konzepten und Techniken aufbaut, die in den vorangegangenen Kapiteln eingeführt wurden. Gegliedert ist das Buch in sieben Kapitel, die die wesentlichen Aspekte von LLMs und deren Implementierung behandeln.

Kapitel 1 bietet eine umfassende Einführung in die grundlegenden Konzepte von LLMs. Es erläutert die Transformer-Architektur, die die Basis für LLMs bildet, wie sie beispielsweise auf der ChatGPT-Plattform realisiert sind.

Kapitel 2 legt einen Plan für den Aufbau eines LLM von Grund auf fest. Es beschreibt den Ablauf davon, wie der Text für das LLM-Training vorbereitet wird. Dazu gehört die Aufteilung des Texts in Wort- und Teilworttokens, die Verwendung der Bytepaar-Codierung für eine fortgeschrittene Tokenisierung, die Auswahl der Stichproben von Trainingsbeispielen mit einem Schiebefensteransatz und das Konvertieren von Tokens in Vektoren, die in das LLM eingespeist werden.

Kapitel 3 konzentriert sich auf die Attention-Mechanismen (die Aufmerksamkeitsmechanismen), die in LLMs verwendet werden. Es stellt ein grundlegen-

des Framework für Self-Attention vor und geht dann zu einem erweiterten Self-Attention-Mechanismus über. Außerdem behandelt das Kapitel die Implementierung eines kausalen Attention-Moduls, das LLMs in die Lage versetzt, einzelne Tokens nacheinander zu erzeugen, zufällig ausgewählte Attention-Gewichte mit Dropout zu maskieren, um Überanpassung zu verringern, und mehrere kausale Attention-Module in einem Multi-Head-Attention-Modul übereinanderzustapeln.

Der Schwerpunkt von Kapitel 4 ist die Codierung eines GPT-artigen LLM, das sich trainieren lässt, um Klartext zu erzeugen. Es beschreibt Techniken wie die Normalisierung von Schichtaktivierungen, um das Training neuronaler Netze zu stabilisieren, das Hinzufügen von Shortcut-Verbindungen in Deep Neural Networks (tiefen neuronalen Netzen), um Modelle effektiver zu trainieren, das Implementieren von Transformer-Blöcken, um GPT-Modelle verschiedener Größen zu erzeugen, und die Berechnung der Parameteranzahl und des Speicherbedarfs von GPT-Modellen.

Kapitel 5 implementiert den Vortrainingsprozess von LLMs. Hier erfahren Sie, wie Sie die Verluste von Trainings- und Validierungsmengen berechnen, um die Qualität des LLM-generierten Texts zu bewerten, wie Sie eine Trainingsfunktion implementieren und das LLM vortrainieren und wie Sie die Modellgewichte speichern und wieder laden, um das Training eines LLM fortzusetzen sowie vortrainierte Gewichte von OpenAI zu laden.

Kapitel 6 stellt verschiedene Ansätze für das Feintuning von LLMs vor. Es beschreibt, wie Sie einen Datensatz für die Textklassifizierung vorbereiten, ein vortrainiertes LLM zum Feintuning modifizieren, ein LLM feintunen, um Spam-Nachrichten zu identifizieren, und die Genauigkeit eines feingetunten LLM-Klassifizierers bewerten.

Kapitel 7 untersucht den Prozess des Feintunings von LLMs per Anweisung, die Organisation von Anweisungsdaten in Trainingsstapeln, das Laden eines vortrainierten LLM und dessen Feinabstimmung, um menschliche Anweisungen zu befolgen, das Extrahieren von LLM-generierten Antworten auf Anweisungen zur Bewertung und die Bewertung eines per Anweisung feingetunten LLM.

## Über den Code

Damit Sie die Codebeispiele in diesem Buch möglichst einfach nachvollziehen können, finden Sie sie auf der Manning-Website unter *https://www.manning.com/books/build-a-large-language-model-from-scratch* und im Jupyter-Notebook-Format auf GitHub unter *https://github.com/rasbt/LLMs-from-scratch*. Und machen Sie sich keine Sorgen, wenn Sie nicht weiterkommen – die Lösungen zu allen Codeübungen finden Sie in Anhang C.

Dieses Buch enthält viele Beispiele für Quellcode sowohl in nummerierten Listings als auch im laufenden Text. In beiden Fällen ist der Quellcode in `Schreibmaschinenschrift` formatiert, um ihn von normalem Text zu unterscheiden.

In vielen Fällen ist der ursprüngliche Quellcode neu formatiert worden. Es sind Zeilenumbrüche hinzugekommen und geänderte Einrückungen, um die Codezeilen an den Platz auf einer Druckseite anzupassen. Außerdem wurden oftmals die Kommentare im Quellcode aus den Listings entfernt, wenn der Text ohnehin den Code beschreibt. Codeanmerkungen sind in vielen Listings zu finden, um wichtige Konzepte hervorzuheben.

Eines der Hauptziele dieses Buchs ist die Zugänglichkeit, sodass Codebeispiele sorgfältig so gestaltet wurden, dass sie sich auf einem normalen Laptop effizient ausführen lassen, ohne dass eine spezielle Hardware erforderlich ist. Wenn Sie aber auf eine GPU zugreifen können, geben Ihnen bestimmte Abschnitte hilfreiche Tipps dazu, wie Sie die Datensätze und Modelle skalieren, um diese zusätzliche Leistung zu nutzen.

Das gesamte Buch hindurch verwenden wir PyTorch als Bibliothek für Tensor-Operationen und Deep-Learning-Routinen, um LLMs von Grund auf zu implementieren. Sollte PyTorch für Sie neu sein, empfehle ich, mit Anhang A zu beginnen, der eine ausführliche Einführung bietet und Empfehlungen für die Einrichtung gibt.

## Andere Onlineressourcen

Interessieren Sie sich für die neuesten Trends in der KI- und LLM-Forschung?

- Besuchen Sie mein Blog unter *https://magazine.sebastianraschka.com*, in dem ich regelmäßig über die neueste KI-Forschung mit Schwerpunkt auf LLMs diskutiere.

Benötigen Sie Hilfe, um sich schneller mit Deep Learning und PyTorch vertraut zu machen?

- Ich biete mehrere kostenlose Kurse auf meiner Website unter *https://sebastianraschka.com/teaching* an. Nutzen Sie diese Ressourcen, um Ihren Einstieg in diese Gebiete anzukurbeln.

Suchen Sie nach Bonusmaterialien zum Buch?

- Im GitHub-Repository des Buchs unter *https://github.com/rasbt/LLMs-from-scratch* finden Sie zusätzliche Ressourcen und Beispiele, die Ihr Lernen ergänzen.

## Danksagungen

Ein Buch zu schreiben ist ein beträchtliches Unterfangen, und ich möchte meiner Frau Liza meinen aufrichtigen Dank für ihre Geduld und Unterstützung während dieses Prozesses aussprechen. Ihre bedingungslose Liebe und ständige Ermutigung waren unverzichtbar.

Unglaublich dankbar bin ich Daniel Kleine, dessen unschätzbares Feedback zu den entstehenden Kapiteln und zum Code meine Erwartungen übertroffen hat. Mit seinem scharfen Blick für Details und seinen aufschlussreichen Vorschlägen haben Daniels Beiträge zweifellos dazu beigetragen, dass dieses Buch zu einem entspannten und unterhaltsamen Leseerlebnis wird.

Ich möchte auch den wunderbaren Mitarbeitern von Manning Publications danken, darunter Michael Stephens für die vielen produktiven Diskussionen, die dazu beigetragen haben, die Ausrichtung dieses Buchs zu bestimmen, und Dustin Archibald, dessen konstruktives Feedback und dessen Anleitung zur Einhaltung der Manning-Richtlinien entscheidend waren. Ich weiß auch eure Flexibilität zu schätzen, mit der ihr den einzigartigen Anforderungen dieses unkonventionellen Ansatzes Rechnung getragen habt. Ein besonderer Dank gilt Aleksandar Dragosavljević, Kari Lucke und Mike Beady für ihre Arbeit an den professionellen Layouts und Susan Honeywell und ihrem Team für die Präzisierung und den Feinschliff der Grafiken.

Robin Campbell und ihrem hervorragenden Marketingteam möchte ich für ihre unschätzbare Unterstützung während des gesamten Schreibprozesses von ganzem Herzen danken.

Schließlich möchte ich mich bei den Gutachtern bedanken: Anandaganesh Balakrishnan, Anto Aravinth, Ayush Bihani, Bassam Ismail, Benjamin Muskalla, Bruno Sonnino, Christian Prokopp, Daniel Kleine, David Curran, Dibyendu Roy Chowdhury, Gary Pass, Georg Sommer, Giovanni Alzetta, Guillermo Alcántara, Jonathan Reeves, Kunal Ghosh, Nicolas Modrzyk, Paul Silisteanu, Raul Ciotescu, Scott Ling, Sriram Macharla, Sumit Pal, Vahid Mirjalili, Vaijanath Rao und Walter Reade für ihr gründliches Feedback zu den Entwürfen. Ihre scharfen Augen und die aufschlussreichen Kommentare haben wesentlich dazu beigetragen, die Qualität dieses Buchs zu verbessern.

Allen, die an dieser Reise mitgewirkt haben, bin ich aufrichtig dankbar. Ihre Unterstützung, ihr Fachwissen und ihr Engagement haben einen entscheidenden Beitrag dazu geleistet, dass dieses Buch zustande gekommen ist. Ich danke euch!

# 1 LLMs verstehen

**In diesem Kapitel:**

- Erläuterungen der grundlegenden Konzepte hinter Large Language Models (LLMs) im Überblick
- Einblicke in die Transformer-Architektur, von der LLMs abgeleitet werden
- Ein Plan für den Aufbau eines LLM von Grund auf

*Large Language Models* (LLMs, große Sprachmodelle), wie sie in ChatGPT von OpenAI angeboten werden, sind Modelle tiefer neuronaler Netze (*Deep Neural Networks*), die in den letzten Jahren entwickelt wurden. Sie haben eine neue Ära in der Verarbeitung natürlicher Sprache (*Natural Language Processing*, NLP) eingeläutet. Bevor LLMs aufgekommen sind, genügten herkömmliche Methoden vollauf bei Kategorisierungsaufgaben wie zum Beispiel E-Mail-Spam-Klassifizierung und einfacher Mustererkennung, die sich mit handgestrickten Regeln oder einfacheren Modellen erfassen ließen. Allerdings waren sie bei Sprachaufgaben, die komplexe Verständnis- und Generierungsfähigkeiten erfordern, wie zum Beispiel detaillierte Anweisungen parsen, Kontextanalysen durchführen sowie kohärent und kontextuell angemessene Originaltexte erzeugen, in der Regel unterlegen. Zum Beispiel konnten frühere Generationen von Sprachmodellen keine E-Mail aus einer Liste von Schlüsselwörtern schreiben – eine Aufgabe, die für moderne LLMs trivial ist.

LLMs besitzen bemerkenswerte Fähigkeiten, um menschliche Sprache zu verstehen, zu erzeugen und zu interpretieren. Allerdings müssen wir Folgendes klarstellen: Wenn wir sagen, dass Sprachmodelle etwas »verstehen«, meinen wir, dass sie Text in einer Weise verarbeiten und erzeugen können, der kohärent und kontextuell relevant erscheint, und nicht, dass sie menschenähnliches Bewusstsein oder Verständnis besitzen.

Dank der Fortschritte beim *Deep Learning*, einem Teilbereich des *Machine Learning* (des maschinellen Lernens) und der *künstlichen Intelligenz* (KI), der sich auf neuronale Netze konzentriert, werden LLMs mit riesigen Mengen von

Textdaten trainiert. Dieses groß angelegte Training versetzt LLMs in die Lage, im Vergleich zu früheren Ansätzen tiefere kontextuelle Informationen und Feinheiten der menschlichen Sprache zu erfassen. Infolgedessen haben LLMs die Leistung in einem breiten Spektrum von NLP-Aufgaben erheblich verbessert, einschließlich Textübersetzung, Stimmungsanalyse, Beantwortung von Fragen und vielem mehr.

Heutige LLMs und frühere NLP-Modelle unterscheiden sich zudem dadurch, dass frühere NLP-Modelle in der Regel für bestimmte Aufgaben wie Textkategorisierung, Sprachübersetzung usw. konzipiert wurden. Diese früheren NLP-Modelle konnten zwar in ihren eng gefassten Anwendungen brillieren, doch LLMs erweisen sich als kompetenter in einem breiten Spektrum von NLP-Aufgaben.

Der Erfolg der LLMs lässt sich auf die Transformer-Architektur zurückführen, die vielen LLMs zugrunde liegt, sowie auf die riesigen Datenmengen, mit denen LLMs trainiert wurden, sodass sie eine umfangreiche Palette an sprachlichen Nuancen, Kontexten und Mustern erfassen können, die manuell nur schwer zu codieren wären.

Dieser Übergang zur Implementierung von Modellen, die auf der Transformer-Architektur basieren und große Trainingsdatensätze verwenden, um LLMs zu trainieren, hat NLP grundlegend verändert, sodass jetzt leistungsfähigere Tools verfügbar sind, um menschliche Sprache zu verstehen und damit zu interagieren.

Die folgende Erörterung umreißt den Ausgangspunkt, um das primäre Ziel dieses Buchs zu erreichen: Verstehen von LLMs durch schrittweise Implementierung des Codes eines ChatGPT-ähnlichen LLM, das auf der Transformer-Architektur basiert.

## 1.1 Was ist ein LLM?

Ein LLM ist ein neuronales Netz, das darauf ausgelegt ist, Klartext zu verstehen, zu erzeugen und darauf zu reagieren. Diese Modelle sind tiefe neuronale Netze (Deep Neural Networks), die mit riesigen Mengen an Textdaten trainiert wurden, die manchmal große Teile des gesamten öffentlich zugänglichen Texts im Internet umfassen.

Das »Large« in »Large Language Models« bezieht sich sowohl auf die Größe des Modells in Bezug auf die Parameter als auch auf den riesigen Datensatz, mit dem es trainiert wurde. Derartige Modelle haben oft Dutzende oder sogar Hunderte von Milliarden an Parametern, d.h. die anpassbaren Gewichte im Netz, die während des Trainings optimiert werden, um das nächste Wort in einer Sequenz vorherzusagen. Die Vorhersage des nächsten Worts ist sinnvoll, weil sie die inhärente sequenzielle Natur der Sprache nutzt, um Modelle für das Verstehen von Kontext, Struktur und Beziehungen im Text zu trainieren. Da es sich um eine sehr

einfache Aufgabe handelt, überrascht es viele Forscher, dass sie dennoch derart leistungsfähige Modelle hervorbringen kann. In späteren Kapiteln werden wir den Ablauf für das Training mit dem nächsten Wort Schritt für Schritt erläutern und implementieren.

LLMs setzen auf eine als *Transformer* bezeichnete Architektur, die es ihnen ermöglicht, Aufmerksamkeit selektiv auf verschiedene Teile der Eingabe zu richten, um Vorhersagen zu erstellen, sodass sie speziell dafür geeignet sind, die Nuancen und Komplexitäten der menschlichen Sprache zu berücksichtigen.

Da LLMs in der Lage sind, Text zu generieren, betrachtet man sie oftmals auch als eine Form der generativen künstlichen Intelligenz, kurz GenAI (für *Generative Artificial Intelligence*). Wie Abbildung 1.1 zeigt, umfasst künstliche Intelligenz die Entwicklung von Maschinen, die Aufgaben ausführen können, für die eine menschliche Intelligenz erforderlich ist – einschließlich Sprache verstehen, Muster erkennen und Entscheidungen treffen –, und Teilbereiche wie Machine Learning oder Deep Learning.

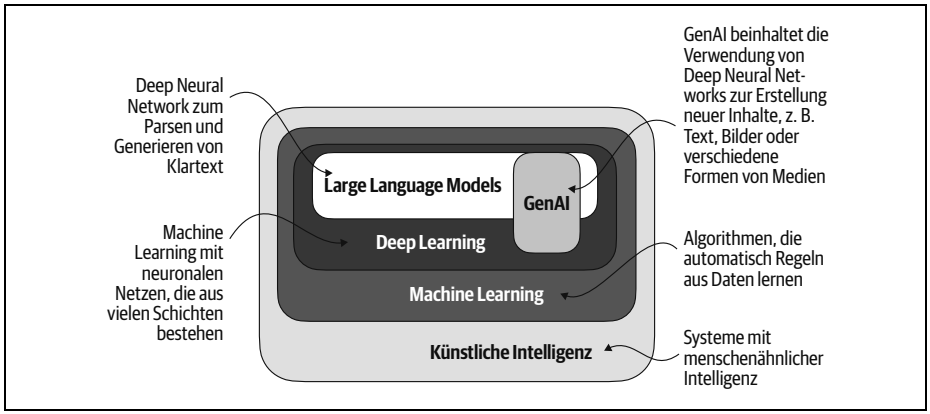


***Abb. 1.1*** *Wie diese hierarchische Darstellung der Beziehungen zwischen den verschiedenen Bereichen zeigt, verkörpern LLMs eine spezifische Anwendung von Deep-Learning-Techniken, indem sie deren Fähigkeit nutzen, menschenähnlichen Text zu verarbeiten und zu erzeugen. Deep Learning ist ein spezialisierter Zweig des Machine Learning, der sich auf mehrschichtige neuronale Netze stützt. Machine Learning und Deep Learning sind Bereiche mit dem Ziel, Algorithmen zu implementieren, die Computer in die Lage versetzen, aus Daten zu lernen und Aufgaben durchzuführen, die normalerweise menschliche Intelligenz erfordern.*

Die Algorithmen, die KI implementieren sollen, stehen im Mittelpunkt des Machine Learning. Insbesondere geht es bei Machine Learning um die Entwicklung von Algorithmen, die anhand von Daten lernen und Vorhersagen oder Entscheidungen treffen können, ohne explizit programmiert zu werden. Um dies zu veranschaulichen, stellen Sie sich einen Spam-Filter als praktische Anwendung des

Machine Learning vor. Anstatt Spam-E-Mails mithilfe von manuell formulierten Regeln zu identifizieren, wird ein Algorithmus für Machine Learning mit Beispielen von E-Mails gefüttert, die als Spam- und Nicht-Spam-E-Mails gekennzeichnet sind. Indem man den Fehler des Modells in seinen Vorhersagen auf einem Trainingsdatensatz minimiert, lernt es, Muster und Charakteristika von Spam zu erkennen, sodass es in die Lage versetzt wird, neue E-Mails entweder als Spam oder als Nicht-Spam zu klassifizieren.

Wie Abbildung 1.1 zeigt, bildet Deep Learning einen Teilbereich des Machine Learning, bei dem es darum geht, komplexe Muster und Abstraktionen in den Daten durch neuronale Netze mit drei oder mehr Schichten (auch als Deep Neural Networks bezeichnet) zu modellieren. Im Gegensatz zum Deep Learning ist beim herkömmlichen Machine Learning eine manuelle Merkmalsextraktion erforderlich. Das heißt, dass menschliche Experten die relevantesten Features für das Modell identifizieren und auswählen müssen.

Der Bereich der künstlichen Intelligenz wird heute von Machine Learning und Deep Learning dominiert, umfasst aber auch andere Ansätze – zum Beispiel regelbasierte Systeme, genetische Algorithmen, Expertensysteme, Fuzzy-Logik oder Computeralgebra (symbolische Manipulation algebraischer Ausdrücke).

Kommen wir auf das Beispiel der Spam-Klassifizierung zurück: Beim traditionellen Machine Learning könnten menschliche Experten manuell Merkmale aus dem E-Mail-Text herausziehen, wie zum Beispiel die Häufigkeit bestimmter Trigger-Wörter (etwa »Preis«, »Gewinn«, »kostenlos«), die Anzahl der Ausrufezeichen, die Verwendung von Wörtern in Großbuchstaben oder das Vorhandensein verdächtiger Links. Mit diesem Datensatz, der auf der Grundlage der von menschlichen Experten definierten Merkmale erstellt wurde, wird dann das Modell trainiert. Im Unterschied zum herkömmlichen Machine Learning ist beim Deep Learning kein manuelles Extrahieren erforderlich. Für ein Deep-Learning-Modell müssen also keine menschlichen Experten die relevantesten Merkmale identifizieren und auswählen. (Allerdings müssen sowohl beim herkömmlichen Machine Learning als auch beim Deep Learning für die Spam-Klassifizierung immer noch Labels erfasst werden, zum Beispiel ob es sich um Spam oder Nicht-Spam handelt, die entweder von einem Experten oder von den Usern bestimmt werden.)

Schauen wir uns nun an, für welche Probleme LLMs heute infrage kommen, welche Herausforderungen LLMs angehen können und wie die allgemeine LLM-Architektur aussieht, die wir später implementieren werden.

## 1.2 Anwendungen von LLMs

Dank ihrer fortgeschrittenen Fähigkeiten, unstrukturierte Textdaten zu analysieren und zu verstehen, sind LLMs in einem breiten Spektrum von Anwendungen in verschiedenen Bereichen zu finden. Heute nutzt man LLMs für die maschinelle Übersetzung, das Generieren von Prosatexten (siehe Abbildung 1.2) sowie Stim-

mungsanalysen, Textzusammenfassungen und viele andere Aufgaben. Seit Kurzem werden LLMs auch für das Erstellen von Inhalten verwendet, um beispielsweise Romane, Artikel oder sogar Computercode zu schreiben.

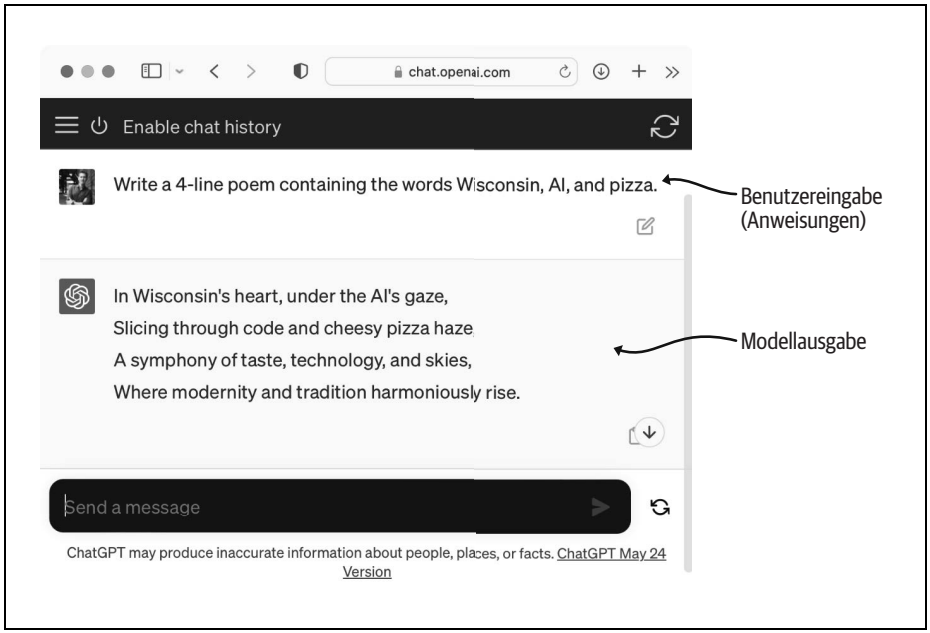


***Abb. 1.2*** *LLM-Schnittstellen ermöglichen die Kommunikation zwischen Benutzern und KI-Systemen in natürlicher Sprache. Dieser Screenshot zeigt, wie ChatGPT ein Gedicht nach den Vorgaben eines Benutzers schreibt.*

LLMs können auch anspruchsvolle Chatbots und virtuelle Assistenten antreiben, wie es zum Beispiel bei ChatGPT von OpenAI und Gemini (früher Bard genannt) von Google der Fall ist. Derartige Anwendungen können Benutzeranfragen beantworten und herkömmliche Suchmaschinen wie Google Search oder Microsoft Bing ergänzen.

Darüber hinaus eignen sich LLMs zur effektiven Wissensabfrage aus riesigen Textmengen in Spezialgebieten wie Medizin oder Recht. Dazu gehören das Durchsuchen von Dokumenten, das Zusammenfassen langer Passagen und die Beantwortung technischer Fragen.

Kurz gesagt, LLMs sind von unschätzbarem Wert für die Automatisierung fast aller Aufgaben, die das Parsen und Generieren von Text beinhalten. Die Anwendungsmöglichkeiten sind schier unendlich, und da wir weiterhin Innovationen entwickeln und neue Wege zur Nutzung dieser Modelle erforschen, ist klar, dass LLMs das Potenzial besitzen, unsere Beziehung zur Technologie neu zu definieren, indem sie sie dialogfähiger, intuitiver und zugänglicher machen.

Uns geht es in erster Linie darum, die prinzipielle Funktionsweise von LLMs zu verstehen. Zu diesem Zweck programmieren wir ein LLM, das Texte erzeugen kann. Außerdem lernen Sie Techniken kennen, die es LLMs ermöglichen, Abfragen durchzuführen, die von der Beantwortung von Fragen über die Zusammenfassung von Text bis hin zur Übersetzung von Text in verschiedene Sprachen reichen – und vieles mehr. Sie werden mit anderen Worten lernen, wie komplexe LLM-Assistenten à la ChatGPT funktionieren, indem Sie einen solchen Schritt für Schritt aufbauen.

## 1.3 Phasen beim Erstellen und Verwenden von LLMs

Warum sollten wir unsere eigenen LLMs erstellen? Ein LLM von Grund auf zu codieren, ist eine ausgezeichnete Übung, um dessen Mechanismen und Grenzen zu verstehen. Außerdem erhalten wir so das nötige Wissen, um vorhandene Open-Source-LLM-Architekturen für unsere domänenspezifischen Datensätze oder Aufgaben vorab zu trainieren oder feinzutunen.

**Hinweis**

Die meisten LLMs werden heute mithilfe der Deep-Learning-Bibliothek PyTorch implementiert, die wir ebenfalls verwenden. In Anhang A finden Sie eine umfassende Einführung in PyTorch.

Wie die Forschung in Bezug auf die Modellierungsleistung gezeigt hat, können benutzerdefinierte LLMs – solche, die auf spezifische Aufgaben oder Bereiche zugeschnitten sind – allgemeine LLMs – solche, wie sie von ChatGPT bereitgestellt und für ein breites Anwendungsspektrum konzipiert sind – übertreffen. Beispiele hierfür sind BloombergGPT (spezialisiert auf Finanzen) und LLMs, die auf die Beantwortung medizinischer Fragen zugeschnitten sind (siehe Anhang B für weitere Details).

Maßgeschneiderte LLMs bieten mehrere Vorteile, insbesondere im Hinblick auf den Datenschutz. Zum Beispiel können Unternehmen darauf bestehen, keine sensiblen Daten mit Drittanbietern von LLMs wie OpenAI zu teilen, da sie Bedenken hinsichtlich der Vertraulichkeit haben. Darüber hinaus ermöglicht die Entwicklung kleinerer benutzerdefinierter LLMs ein direktes Deployment auf Kundengeräten wie Laptops und Smartphones, was von Unternehmen wie Apple derzeit erforscht wird.

Diese lokale Implementierung kann die Latenzzeit erheblich senken und die serverbezogenen Kosten verringern. Darüber hinaus gewähren benutzerdefinierte LLMs den Entwicklerinnen und Entwicklern völlige Autonomie, sodass sie Aktualisierungen und Änderungen des Modells nach Bedarf steuern können.

Der allgemeine Ablauf beim Erstellen eines LLM umfasst das Vortraining und das Feintuning. Das »Vor« in Vortraining bezieht sich auf die Anfangsphase, in der ein Modell wie ein LLM mit einem großen, breit gefächerten Datensatz trainiert wird, um ein umfassendes Verständnis von Sprache zu entwickeln. Dieses vortrainierte Modell dient dann als grundlegende Ressource, die sich durch ein Feintuning weiterentwickeln lässt. Bei diesem Prozess wird das Modell speziell mit einem engeren Datensatz trainiert, der für bestimmte Aufgaben oder Bereiche spezifischer ist. Abbildung 1.3 veranschaulicht diesen zweistufigen Trainingsansatz, der aus Vortraining und Feintuning besteht.

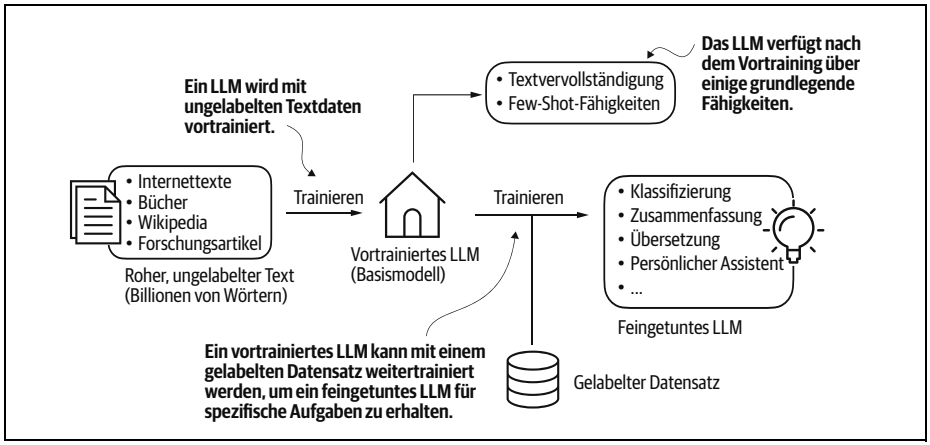


***Abb. 1.3*** *Das Vortraining eines LLM beinhaltet die Vorhersage des nächsten Worts auf großen Textdatensätzen. Ein vortrainiertes LLM kann dann mit einem kleineren gelabelten Datensatz feingetunt werden.*

Um ein LLM zu erstellen, trainiert man es im ersten Schritt mit einem großen Korpus von Textdaten, den man auch als *Rohtext* bezeichnet. Hier bezieht sich »roh« auf die Tatsache, dass es sich bei diesen Daten lediglich um normalen Text ohne irgendwelche Beschriftungsinformationen handelt. (Es kann aber ein Filter angewendet werden, um beispielsweise Formatierungszeichen oder Dokumente in unbekannten Sprachen zu entfernen.)

**Hinweis**

Leser, die sich bereits mit Machine Learning auskennen, werden feststellen, dass für herkömmliche Modelle des Machine Learning und Deep Neural Networks, die über die konventionellen überwachten Lernparadigmen trainiert werden, normalerweise Beschriftungsinformationen erforderlich sind. Dies ist jedoch nicht der Fall für die Vortrainingsphase von LLMs. In dieser Phase verwenden LLMs Self-supervised Learning (selbstüberwachtes Lernen), bei dem das Modell seine eigenen Labels aus den Eingabedaten generiert.

In der als *Vortraining* bezeichneten ersten Trainingsphase eines LLM wird ein erstes vortrainiertes LLM erstellt, das oft als *Basis-* oder *Grundmodell* bezeichnet wird. Ein typisches Beispiel für ein derartiges Modell ist das GPT-3-Modell (der Vorläufer des in ChatGPT angebotenen Originalmodells). Dieses Modell ist in der Lage, Text zu vervollständigen – d.h., einen halb geschriebenen Satz, den der Benutzer bereitstellt, zu beenden. Zudem besitzt es beschränkte Few-Shot-Fähigkeiten, kann also neue Aufgaben auf der Grundlage von nur wenigen Beispielen erlernen, anstatt umfangreiche Trainingsdaten zu benötigen.

Ist ein LLM mit großen Textdatensätzen für die Vorhersage des nächsten Worts im Text vortrainiert worden, können wir das LLM mit gelabelten Daten weitertrainieren, was auch als *Feintuning* (Feinabstimmung) bezeichnet wird.

Die beiden populärsten Kategorien des Feintunings von LLMs sind das *Feintuning per Anweisung* und das *Feintuning per Klassifizierung*. Beim Feintuning per Anweisung besteht der gelabelte Datensatz aus Anweisungs-Antwort-Paaren, wie zum Beispiel die Anfrage zur Übersetzung eines Texts, die vom korrekt übersetzten Text begleitet wird. Beim Feintuning per Klassifizierung besteht der gelabelte Datensatz aus Texten und zugeordneten Klassenlabels – zum Beispiel E-Mails, denen die Labels »Spam« und »Nicht-Spam« zugeordnet sind.

Wir werden die Codeimplementierungen für das Vortraining und das Feintuning eines LLM behandeln und uns nach dem Vortraining eines grundlegenden LLM eingehender mit den Besonderheiten des Feintunings – sowohl per Anweisung als auch per Klassifizierung – befassen.

## 1.4 Einführung in die Transformer-Architektur

Die meisten modernen LLMs stützen sich auf die *Transformer*-Architektur, eine Architektur für Deep Neural Networks, die 2017 im Paper »Attention Is All You Need« (*https://arxiv.org/abs/1706.03762*) vorgestellt wurde. Um LLMs zu verstehen, müssen wir den ursprünglichen Transformer verstehen, der für die maschinelle Übersetzung von englischen Texten ins Deutsche und Französische entwickelt wurde. Abbildung 1.4 stellt eine vereinfachte Version der Transformer-Architektur dar.

Die Transformer-Architektur besteht aus zwei Teilmodulen: einem Encoder und einem Decoder. Das Encoder-Modul verarbeitet den Eingabetext und codiert ihn in eine Folge von numerischen Darstellungen oder Vektoren, die die kontextuelle Information der Eingabe erfassen. Dann übernimmt das Decoder-Modul diese codierten Vektoren und generiert den Ausgabetext. Zum Beispiel würde in einer Übersetzungsaufgabe der Encoder den Text aus der Quellsprache in Vektoren codieren, und der Decoder würde diese Vektoren decodieren, um Text in der Zielsprache zu generieren. Sowohl der Encoder als auch der Decoder bestehen aus vielen Schichten, die durch einen sogenannten Self-Attention-Mechanismus miteinander verbunden sind. Möglicherweise haben Sie viele Fragen dazu, wie

die Eingaben vorverarbeitet und codiert werden. Diese Fragen klären wir in den folgenden Kapiteln anhand einer schrittweisen Implementierung.

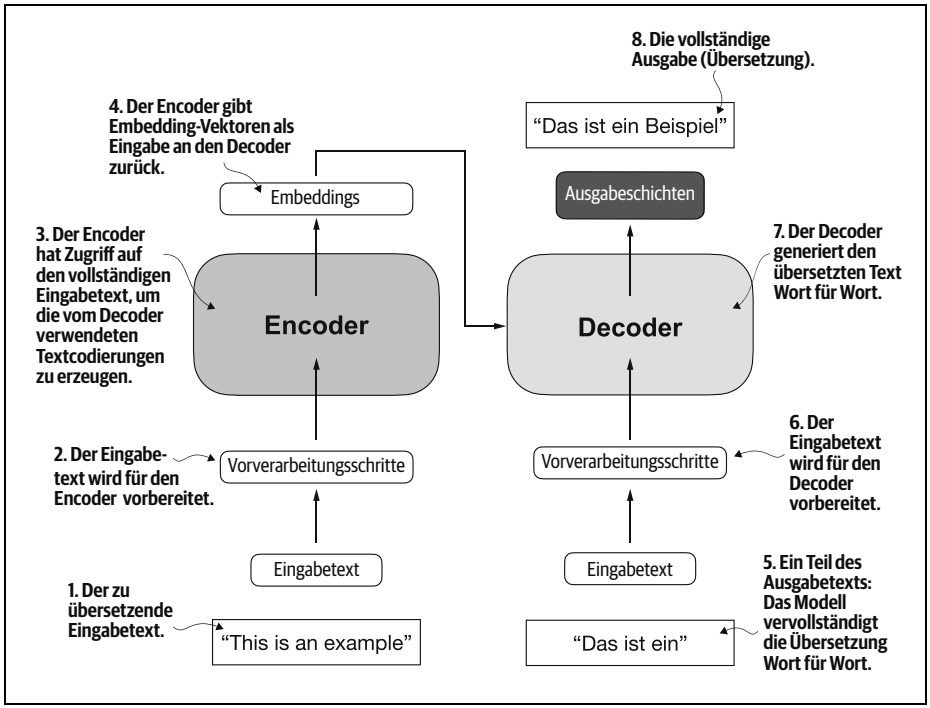


***Abb. 1.4*** *Eine vereinfachte Darstellung der ursprünglichen Transformer-Architektur, die ein Deep-Learning-Modell für die Sprachübersetzung ist. Der Transformer besteht aus zwei Teilen: einem Encoder (links), der den Eingabetext verarbeitet und eine Embedding-Repräsentation des Texts erzeugt (eine numerische Darstellung, die viele verschiedene Faktoren in verschiedenen Dimensionen erfasst), die der Decoder (rechts) verwenden kann, um den übersetzten Text Wort für Wort zu erzeugen. Die Abbildung zeigt die letzte Phase des Übersetzungsprozesses, in der der Decoder für den ursprünglichen Eingabetext (»This is an example«) und einen teilweise übersetzten Satz (»Das ist ein«) nur noch das letzte Wort (»Beispiel«) erzeugen muss, um die Übersetzung abzuschließen.*

Eine Schlüsselkomponente von Transformern und LLMs ist der Self-Attention-Mechanismus (hier nicht gezeigt), der es dem Modell ermöglicht, die Bedeutung verschiedener Wörter oder Tokens in einer Sequenz relativ zueinander zu gewichten. Dieser Mechanismus ermöglicht dem Modell, weitreichende Abhängigkeiten und kontextuelle Beziehungen innerhalb der Eingabedaten zu erfassen. Dies erweitert seine Fähigkeiten, kohärent und kontextuell relevante Ausgaben zu erzeugen. Allerdings verschieben wir die weitere Erläuterung aufgrund der Komplexität auf Kapitel 3, wo wir ihn Schritt für Schritt diskutieren und implementieren werden.

Spätere Varianten der Transformer-Architektur, wie BERT (kurz für *Bidirectional Encoder Representations from Transformers*) und die verschiedenen GPT-Modelle (kurz für *Generative Pretrained Transformers*), bauten auf diesem Konzept auf, um diese Architektur für verschiedene Aufgaben anzupassen. In Anhang B finden Sie hierzu weitere Literaturempfehlungen.

BERT, das auf dem Encoder-Submodul des ursprünglichen Transformers aufbaut, unterscheidet sich in seinem Trainingsansatz von GPT. Während GPT für generative Aufgaben entwickelt wurde, sind BERT und seine Varianten auf die Vorhersage maskierter Wörter spezialisiert, wobei das Modell maskierte oder versteckte Wörter in einem gegebenen Satz vorhersagt, wie Abbildung 1.5 zeigt. Diese einzigartige Trainingsstrategie verleiht BERT Stärken bei Textklassifizierungsaufgaben, einschließlich Stimmungsvorhersage und Dokumentkategorisierung. Als Beispiel für die Anwendung seiner Fähigkeiten verwendet X (vormals Twitter) BERT, um schädliche Inhalte zu erkennen.

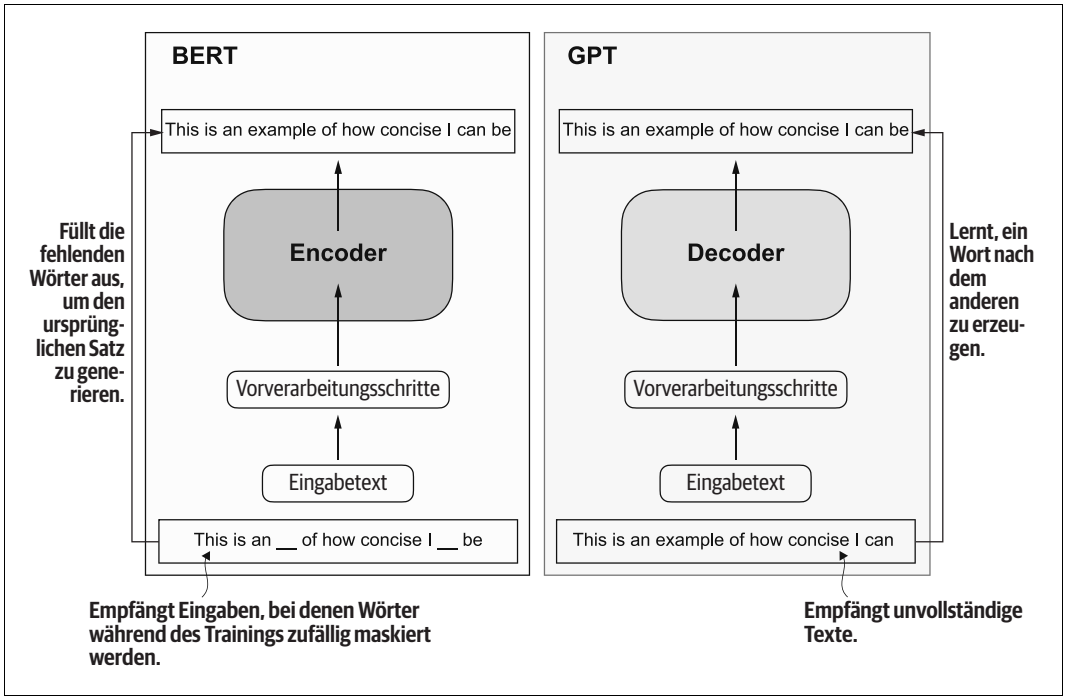


***Abb. 1.5*** *Eine visuelle Darstellung der Encoder- und Decoder-Submodule des Transformers. Die linke Seite zeigt exemplarisch BERT-ähnliche LLMs, die auf die Vorhersage maskierter Wörter ausgelegt sind und hauptsächlich für Aufgaben wie Textklassifizierung verwendet werden. Rechts zeigt das Decoder-Segment GPT-ähnliche LLMs, die für generative Aufgaben und die Erzeugung kohärenter Textsequenzen konzipiert sind.*

GPT hingegen konzentriert sich auf den Decoder-Teil der ursprünglichen Transformer-Architektur und ist für Aufgaben konzipiert, bei denen Texte erstellt werden müssen. Dazu gehören maschinelle Übersetzungen, Textzusammenfassungen, das Schreiben von Belletristik, das Schreiben von Computercode und vieles mehr.

GPT-Modelle, die hauptsächlich dafür entwickelt und trainiert wurden, Texte zu vervollständigen, zeigen ebenfalls eine bemerkenswerte Vielseitigkeit in ihren Fähigkeiten. Diese Modelle sind in der Lage, sowohl Zero-Shot- als auch Few-Shot-Learning-Aufgaben auszuführen. Zero-Shot-Learning bezieht sich auf die Fähigkeit, ohne vorherige spezifische Beispiele auf völlig unbekannte Aufgaben zu generalisieren. Andererseits geht es beim Few-Shot-Learning darum, aus einer minimalen Anzahl von Beispielen zu lernen, die der Benutzer als Eingabe zur Verfügung stellt (siehe Abbildung 1.6).

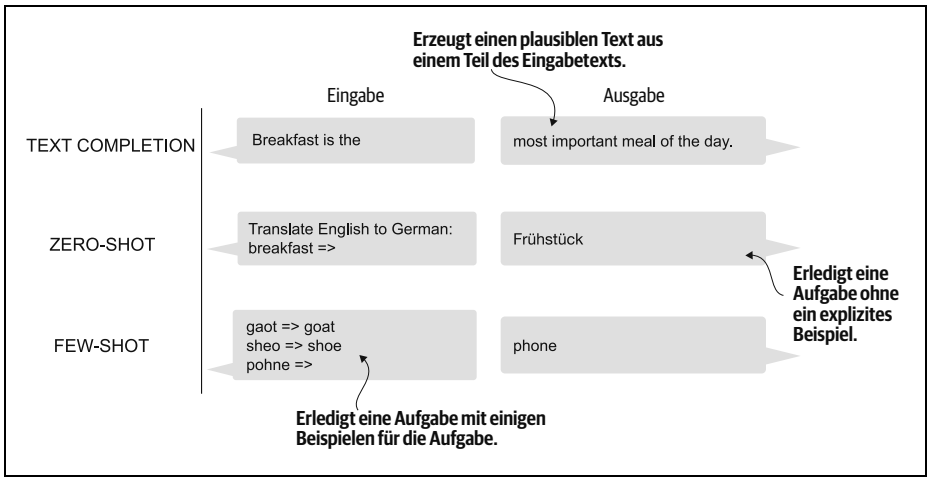


***Abb. 1.6*** *Zusätzlich zur Textvervollständigung können GPT-ähnliche LLMs verschiedene Aufgaben anhand ihrer Eingaben lösen, ohne dass Neutraining, Feintuning oder aufgabenspezifische Änderungen der Modellarchitektur erforderlich sind. Manchmal ist es hilfreich, Beispiele des Ziels innerhalb der Eingabe bereitzustellen, was als »Few-Shot-Setting« bekannt ist. Allerdings sind GPT-ähnliche LLMs auch in der Lage, Aufgaben ohne ein konkretes Beispiel zu realisieren, was man als »Zero-Shot-Setting« bezeichnet.*

**Transformer- vs. LLM-Architekturen**

Die heutigen LLMs basieren auf der Transformer-Architektur. Daher sind Transformer und LLMs Begriffe, die in der Literatur oft synonym verwendet werden. Beachten Sie aber, dass nicht alle Transformer LLMs sind, da Transformer auch für Computervision verwendet werden können. Zudem sind nicht alle LLMs Transformer, da LLMs auf rekurrenten und konvolutionalen Architekturen basieren. Die Hauptmotivation hinter diesen

→

alternativen Ansätzen ist die Verbesserung der Berechnungseffizienz von LLMs. Es bleibt abzuwarten, ob diese alternativen LLM-Architekturen mit den Fähigkeiten von Transformer-basierten LLMs konkurrieren können und ob sie sich in der Praxis durchsetzen werden. Der Einfachheit halber verwende ich den Begriff »LLM«, um mich auf Transformer-basierte LLMs ähnlich wie GPT zu beziehen. (Interessierte Leser finden in Anhang B Hinweise auf Quellen, die diese Architekturen beschreiben.)

## 1.5 Große Datensätze nutzen

Die großen Trainingsdatensätze für populäre GPT- und BERT-ähnliche Modelle stellen vielfältige und umfassende Textkorpora dar, die Milliarden von Wörtern umfassen und eine große Bandbreite an Themen sowie natürliche und Computersprachen beinhalten. Um ein konkretes Beispiel zu geben, fasst Tabelle 1.1 den Datensatz zusammen, der für das Vortraining von GPT-3 verwendet wurde, das als Basismodell für die erste Version von ChatGPT diente.

| Datensatzname | Datensatzbeschreibung | Anzahl der Tokens | Anteil an den Trainingsdaten |
|---|---|---|---|
| CommonCrawl (gefiltert) | Web-Crawl-Daten | 410 Milliarden | 60% |
| WebText2 | Web-Crawl-Daten | 19 Milliarden | 22% |
| Books1 | internetbasierter Buchkorpus | 12 Milliarden | 8% |
| Books2 | internetbasierter Buchkorpus | 55 Milliarden | 8% |
| Wikipedia | hochwertiger Text | 3 Milliarden | 3% |

***Tab. 1.1*** *Der Datensatz für das Vortraining des populären GPT-3-LLM*

Tabelle 1.1 gibt die Anzahl der Tokens an, wobei ein Token eine Texteinheit ist, die ein Modell liest. Die Anzahl der Tokens im Datensatz entspricht ungefähr der Anzahl der Wörter und Satzzeichen im Text. Kapitel 2 befasst sich mit der Tokenisierung, d.h. mit der Umwandlung von Text in Tokens.

Die wichtigste Erkenntnis ist, dass Umfang und Vielfalt dieses Trainingsdatensatzes es diesen Modellen ermöglichen, bei verschiedenen Aufgaben, einschließlich Sprachsyntax, Semantik und Kontext, gut abzuschneiden – sogar bei solchen, die Allgemeinwissen erfordern.

**Details zum GPT-3-Datensatz**

Tabelle 1.1 zeigt den Datensatz, der für GPT-3 verwendet wurde. Die Tabellenspalte mit den Anteilen summiert sich zu 100% der Beispieldaten, wobei Rundungsfehler zu berücksichtigen sind. Obwohl die Teilmengen in der Spalte »Anzahl der Tokens« insge-

→

samt 499 Milliarden ergeben, wurde das Modell nur mit etwa 300 Milliarden Tokens trainiert. Die Autoren des Papers zu GPT-3 haben nicht angegeben, warum das Modell nicht mit allen 499 Milliarden Tokens trainiert wurde.

Man bedenke die Größe des Datensatzes CommonCrawl, der allein aus 410 Milliarden Tokens besteht und etwa 570 GB Speicherplatz benötigt. Im Vergleich dazu haben spätere Iterationen von Modellen wie GPT-3 – zum Beispiel Llama von Meta – ihren Trainingsumfang erweitert, um zusätzliche Datenquellen wie die arXiv-Forschungspapers (92 GB) und die codebezogenen Fragen und Antworten von StackExchange (78 GB) einzubeziehen.

Die Autoren des GPT-3-Papers haben den Trainingsdatensatz nicht veröffentlicht, aber ein vergleichbarer und öffentlich zugänglicher Datensatz ist »Dolma: An Open Corpus of Three Trillion Tokens for LLM Pretraining Research« von Soldaini et al. 2024 (*https://arxiv.org/abs/2402.00159*). Allerdings kann die Sammlung urheberrechtlich geschützte Werke enthalten, und die genauen Nutzungsbedingungen können vom beabsichtigten Verwendungszweck und vom Land abhängen.

Wegen ihres Vortrainings sind diese Modelle unglaublich vielseitig für ein weiteres Feintuning bei Downstream-Aufgaben. Deshalb bezeichnet man sie auch als Basis- oder Grundmodelle. Das Vortraining von LLMs setzt den Zugang zu erheblichen Ressourcen voraus und ist sehr teuer. So werden beispielsweise die Kosten für das Vortraining von GPT-3 auf 4,6 Millionen Dollar in Form von Cloud Computing Credits geschätzt (*https://mng.bz/VxEW*).

Die gute Nachricht ist, dass sich viele vortrainierte LLMs, die als Open-Source-Modelle verfügbar sind, als Allzweckwerkzeuge eignen, um Texte, die nicht Teil der Trainingsdaten waren, zu schreiben, zu extrahieren und zu bearbeiten. Außerdem lassen sich LLMs für spezifische Aufgaben mit relativ kleinen Datensätzen feintunen, was die erforderlichen Rechenressourcen reduziert und die Performance verbessert.

Wir werden den Code für das Vortraining implementieren und ihn nutzen, um ein LLM für Lehrzwecke vorab zu trainieren. Alle Berechnungen sind auf Consumer-Hardware ausführbar. Nachdem Sie den Code für das Vortraining implementiert haben, lernen Sie, wie sich offen verfügbare Modellgewichte wiederverwenden und in die von uns implementierte Architektur laden lassen, um die teure Vortrainingsphase zu überspringen, wenn wir unser LLM feintunen.

## 1.6 Die GPT-Architektur unter der Lupe

GPT wurde ursprünglich im Paper »Improving Language Understanding by Generative Pre-Training« (*https://mng.bz/x2qg*) von Radford et al. bei OpenAI vorgestellt. GPT-3 ist eine vergrößerte Version dieses Modells, das mehr Parameter

umfasst und mit einem größeren Datensatz trainiert wurde. Darüber hinaus wurde das ursprüngliche Modell in ChatGPT durch Feintuning von GPT-3 auf einem großen Anweisungsdatensatz mit einer Methode aus dem InstructGPT-Paper von OpenAI (*https://arxiv.org/abs/2203.02155*) erzeugt. Wie Abbildung 1.6 zeigt, sind diese Modelle kompetente Textvervollständigungsmodelle, die auch andere Aufgaben wie Rechtschreibkorrektur, Klassifizierung oder Sprachübersetzung übernehmen können. Dies ist tatsächlich sehr bemerkenswert, wenn man bedenkt, dass die GPT-Modelle mit einer relativ einfachen Aufgabe zur Vorhersage des nächsten Worts trainiert werden, wie Abbildung 1.7 zeigt.

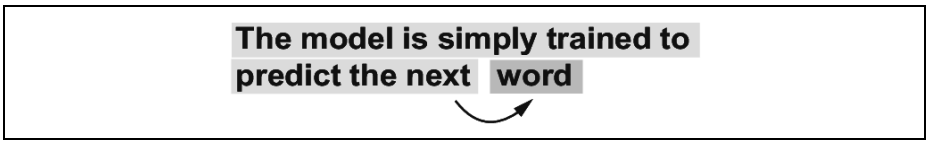


***Abb. 1.7*** *Beim Vortraining der Vorhersage des nächsten Worts für GPT-Modelle lernt das System, das nächste Wort in einem Satz vorherzusagen, indem es sich die Wörter betrachtet, die davor standen. Dieser Ansatz hilft dem Modell, zu verstehen, wie Wörter und Sätze in der Sprache typischerweise zusammenpassen, und bildet eine Grundlage, die auf verschiedene andere Aufgaben angewendet werden kann.*

Die Aufgabe, das nächste Wort vorherzusagen, ist eine Form des selbstüberwachten Lernens (Self-supervised Learning), d.h. eine Form der Selbstbeschriftung. Das bedeutet, dass wir Labels für die Trainingsdaten nicht explizit sammeln müssen, sondern die Struktur der Daten selbst nutzen können: indem wir das nächste Wort in einem Satz oder Dokument als das Label verwenden, das das Modell vorhersagen soll. Da uns diese Aufgabe der Vorhersage des nächsten Worts erlaubt, Labels »während des Betriebs« zu erstellen, ist es möglich, LLMs mit riesigen ungelabelten Textdatensätzen zu trainieren.

Verglichen mit der ursprünglichen Transformer-Architektur, die Abschnitt 1.4 behandelt hat, ist die allgemeine GPT-Architektur relativ einfach. Im Wesentlichen handelt es sich nur um den Decoder-Teil ohne den Encoder (siehe Abbildung 1.8). Da Decoder-ähnliche Modelle wie GPT den Text generieren, indem sie ein Wort nach dem anderen vorhersagen, betrachtet man sie als eine Art *autoregressives Modell*. Autoregressive Modelle beziehen ihre vorherigen Ausgaben als Eingaben für zukünftige Vorhersagen ein. Folglich wird bei GPT jedes neue Wort auf der Grundlage der ihm vorausgehenden Sequenz ausgewählt, was die Kohärenz des resultierenden Texts verbessert.

Architekturen wie GPT-3 sind auch erheblich größer als das ursprüngliche Transformer-Modell. So werden im ursprünglichen Transformer die Encoder- und Decoder-Blöcke sechsmal wiederholt. GPT-3 umfasst 96 Transformer-Schichten und insgesamt 175 Milliarden Parameter.

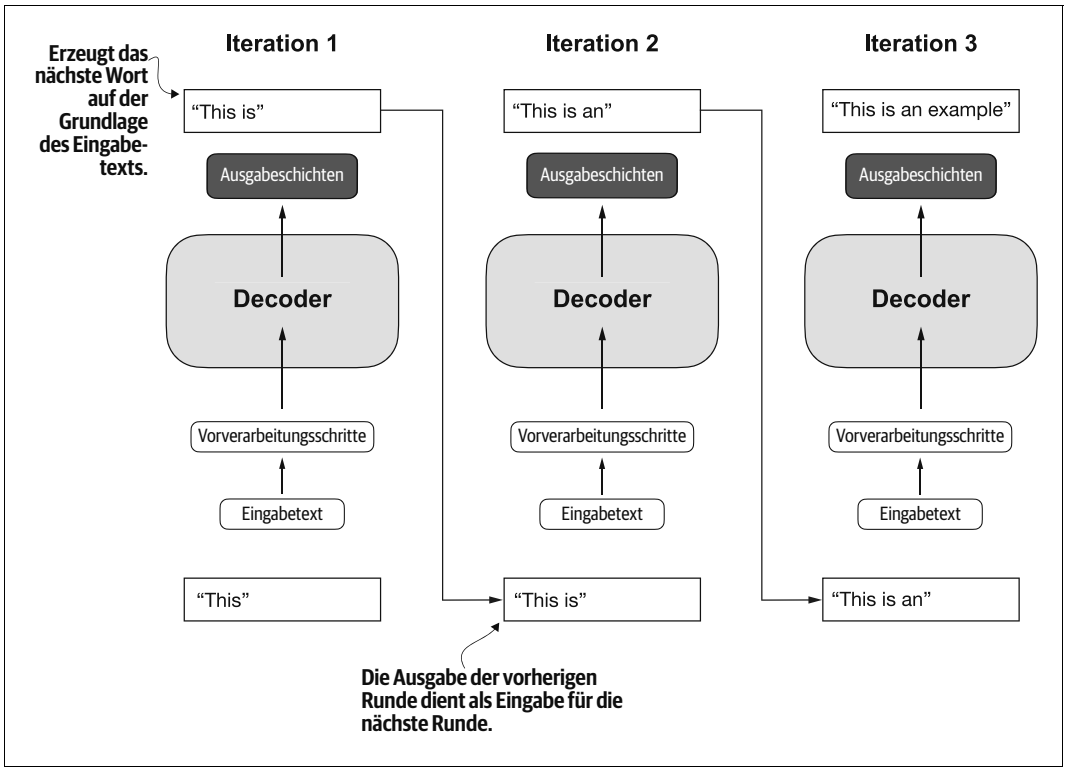


***Abb. 1.8*** *Die GPT-Architektur nutzt nur den Decoder-Teil des ursprünglichen Transformers. Konzipiert ist diese Architektur für unidirektionale Verarbeitung von links nach rechts, sodass sie gut geeignet ist für Aufgaben wie Textgenerierung und die Vorhersage des nächsten Worts, um Text auf iterative Weise Wort für Wort zu erzeugen.*

GPT-3 wurde im Jahr 2020 eingeführt, was nach den Maßstäben von Deep Learning und der Entwicklung großer Sprachmodelle als vor langer Zeit zu betrachten ist. Allerdings basieren die neueren Architekturen wie die Llama-Modelle von Meta immer noch auf denselben zugrunde liegenden Konzepten und weisen nur geringfügige Änderungen auf. Daher ist das Verständnis von GPT so wichtig wie eh und je, und ich konzentriere mich auf die Implementierung der prominenten Architektur hinter GPT, während ich Hinweise auf spezifische Anpassungen gebe, die von alternativen LLMs genutzt werden.

Obwohl das ursprüngliche Transformer-Modell, das aus Encoder- und Decoder-Blöcken besteht, ausdrücklich für die Sprachübersetzung entwickelt wurde, sind GPT-Modelle – trotz ihrer zwar größeren, aber auch einfacheren reinen Decoder-Architektur, die auf die Vorhersage des nächsten Worts abzielt – ebenfalls in der Lage, Übersetzungsaufgaben zu erfüllen. Diese Fähigkeit war für die Forschenden zunächst unerwartet, da sie aus einem Modell hervorging, das in erster Linie

für die Vorhersage des nächsten Worts trainiert wurde, also für eine Aufgabe, die nicht speziell auf die Übersetzung abzielte.

Die Fähigkeit, Aufgaben auszuführen, für die das Modell nicht explizit trainiert wurde, bezeichnet man als *emergentes Verhalten*. Diese Fähigkeit wird dem Modell nicht explizit während des Trainings beigebracht, sondern ergibt sich als natürliche Konsequenz aus dem Umgang des Modells mit großen mehrsprachigen Daten in unterschiedlichen Kontexten. Die Tatsache, dass GPT-Modelle die Übersetzungsmuster zwischen Sprachen »erlernen« und Übersetzungsaufgaben ausführen können, obwohl sie nicht speziell dafür trainiert wurden, zeigt die Vorteile und Fähigkeiten dieser großen, generativen Sprachmodelle. Wir können verschiedene Aufgaben erfüllen, ohne für jede Aufgabe unterschiedliche Modelle zu verwenden.

## 1.7 Ein großes Sprachmodell aufbauen

Nachdem wir nun die Grundlagen für das Verständnis von LLMs geschaffen haben, wollen wir eines von Grund auf programmieren. Wir greifen die fundamentale Idee hinter GPT als Blaupause auf und realisieren diese Aufgabe in drei Phasen, wie sie Abbildung 1.9 veranschaulicht.

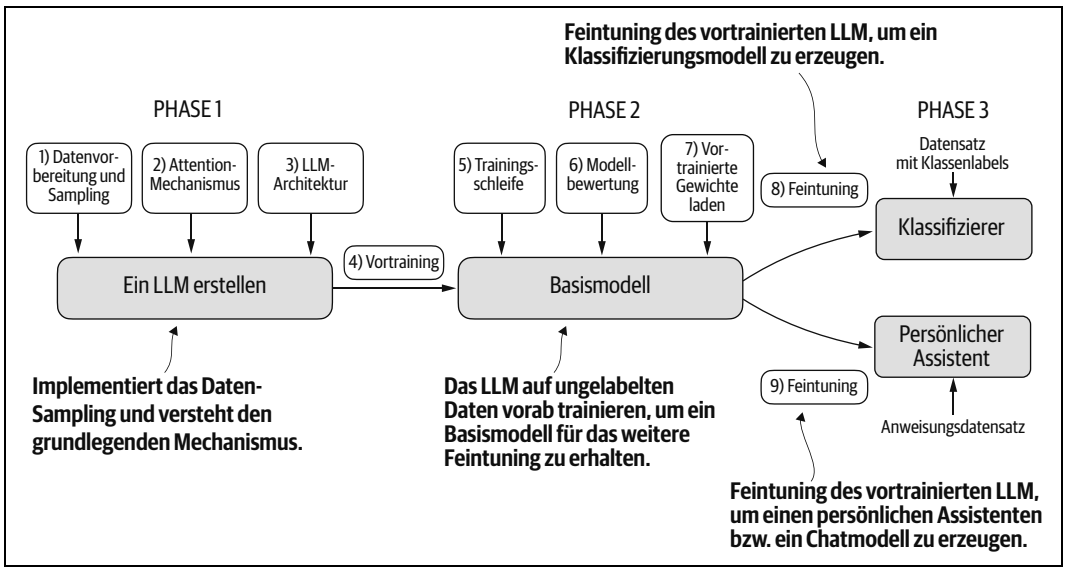


***Abb. 1.9*** *Die drei Hauptphasen für die Programmierung eines LLM. PHASE 1: Implementierung der LLM-Architektur und Datenvorbereitungsprozess, PHASE 2: Vortraining eines LLM, um ein Basismodell zu erstellen, und PHASE 3: Feintuning des Basismodells, um ein persönlicher Assistent oder Klassifizierer zu werden.*

In Phase 1 lernen Sie die grundlegenden Schritte der Datenvorverarbeitung kennen und codieren den Attention-Mechanismus, das Herzstück jedes LLM. Als Nächstes erfahren Sie in Phase 2, wie man ein GPT-ähnliches LLM codiert und trainiert, das in der Lage ist, neue Texte zu generieren. Außerdem werden wir uns mit den Grundlagen der Bewertung von LLMs beschäftigen, die für die Entwicklung leistungsfähiger NLP-Systeme unerlässlich ist.

Das Vortraining eines LLM von Grund auf ist ein bedeutendes Unterfangen, das Tausende oder Millionen von Dollar an Rechenkosten für GPT-ähnliche Modelle verschlingt. Daher liegt der Schwerpunkt bei Phase 2 auf der Implementierung des Trainings für Lehrzwecke anhand eines kleinen Datensatzes. Darüber hinaus bringe ich auch Beispiele für Code, mit dem sich frei verfügbare Modellgewichte laden lassen.

Schließlich nehmen wir in Phase 3 ein vortrainiertes LLM und feintunen es, um Anweisungen zu befolgen, wie zum Beispiel Fragen zu beantworten oder Texte zu klassifizieren – die häufigsten Aufgaben in vielen realen Anwendungen und in der Forschung.

Ich hoffe, Sie freuen sich auf diese spannende Reise!

## 1.8 Zusammenfassung

- LLMs haben das Gebiet der Verarbeitung natürlicher Sprache umgewandelt, das sich bis dahin vorwiegend auf explizit regelbasierte Systeme und einfachere statistische Methoden gestützt hat. Das Aufkommen von LLMs förderte auch neue auf Deep Learning basierende Ansätze zutage, die zu Fortschritten beim Verstehen, Erzeugen und Übersetzen menschlicher Sprache geführt haben.
- Moderne LLMs werden in zwei Hauptschritten trainiert:
  - Zunächst werden sie auf einem großen Korpus von ungelabeltem Text trainiert, indem die Vorhersage des nächsten Worts in einem Satz als Label verwendet wird.
  - Dann werden sie mit einem kleineren, gelabelten Zieldatensatz feingetunt, um Anweisungen zu befolgen oder Klassifizierungsaufgaben durchzuführen.
- LLMs basieren auf der Transformer-Architektur. Die Schlüsselidee der Transformer-Architektur ist ein Attention-Mechanismus, der dem LLM selektiven Zugriff auf die gesamte Eingabesequenz gibt, wenn die Ausgabe Wort für Wort generiert wird.
- Die ursprüngliche Transformer-Architektur besteht aus einem Encoder, der den Text parst, und einem Decoder, der Text generiert.

- LLMs wie GPT-3 und ChatGPT, die Text generieren und Anweisungen befolgen sollen, implementieren nur Decoder-Module, was die Architektur vereinfacht.
- Große Datensätze, die aus Milliarden von Wörtern bestehen, sind für das Vortraining von LLMs unerlässlich.
- Während die allgemeine Vortrainingsaufgabe für GPT-ähnliche Modelle darin besteht, das nächste Wort in einem Satz vorherzusagen, weisen derartige LLMs emergente Eigenschaften auf, wie zum Beispiel die Fähigkeit, Texte zu klassifizieren, zu übersetzen oder zusammenzufassen.
- Sobald ein LLM vortrainiert ist, kann das resultierende Basismodell für verschiedene Downstream-Aufgaben effizienter feingetunt werden.
- LLMs, die auf benutzerdefinierten Datensätzen feingetunt wurden, können allgemeine LLMs bei spezifischen Aufgaben übertreffen.

# Inhalt

# Large Language Models selbst programmieren

- Ein LLM selbst erstellen und von Grund auf verstehen
- US-Bestseller – der ideale Praxiseinstieg in das Thema LLMs
- Sebastian Raschka erklärt jeden Entwicklungsschritt sehr gut verständlich

Dieses Buch ist eine spannende Reise in die Blackbox der Generativen KI: Ohne auf bestehende LLM-Bibliotheken zurückzugreifen, programmieren Sie ein LLM-Basismodell im GPT-Stil auf dem eigenen Rechner. Sie entwickeln es zu einem Textklassifikator weiter und erstellen schließlich einen Chatbot, der Ihren Anweisungen folgt und den Sie als persönlichen KI-Assistenten verwenden können. Jeder Schritt wird mit klaren Beschreibungen, Diagrammen und Beispielen erklärt.

Auf diese Weise eignen Sie sich aktiv und ganz praktisch grundlegendes Wissen zur aktuell wichtigsten KI-Technologie an – denn Sie haben Ihren Chatbot selbst gebaut! Während Sie die einzelnen Phasen der LLM-Erstellung durchlaufen, entwickeln Sie eine klarere Vorstellung davon, wie LLMs unter der Haube funktionieren.

Sie erfahren, wie Sie

- alle Bestandteile eines LLMs planen und programmieren
- einen für das LLM-Training geeigneten Datensatz vorbereiten
- das LLM mit Ihren eigenen Daten optimieren
- Feedback nutzen, um sicherzustellen, dass das LLM Ihren Anweisungen folgt
- vortrainierte Gewichte in das LLM laden

*»Wirklich inspirierend! Das Buch motiviert Sie, Ihre neuen Fähigkeiten in die Tat umzusetzen.«*

**Benjamin Muskalla**
**Senior Engineer, GitHub**

**Sebastian Raschka**, PhD, arbeitet seit mehr als einem Jahrzehnt im Bereich Machine Learning und KI. Er ist Staff Research Engineer bei Lightning AI, wo er LLM-Forschung betreibt und Open-Source-Software entwickelt. Sebastian ist nicht nur Forscher, sondern hat auch eine große Leidenschaft für die Vermittlung von Wissen. Bekannt ist er für seine Bestseller zu Machine Learning und seine Beiträge zu Open Source.

**€ 39,90 (D)**

Gedruckt in Deutschland
Mineralölfreie Druckfarben
Zertifiziertes Papier

**ISBN 978-3-98889-044-3**