KAPITEL 2

Wie KI-Coding-Technologie funktioniert

In diesem Kapitel schrauben wir die KI-gestützten Programmiertools auf und schauen uns an, was unter der Motorhaube passiert. Wir werden einen kurzen Ausflug in die Geschichte unternehmen, uns mit Transformer-Modellen und LLMs beschäftigen und den OpenAI Playground vorstellen. Dann werden wir auch noch Ratschläge zum Bewerten von LLMs geben.

Wenn Sie verstehen, was diese leistungsfähige Technologie kann – und was nicht –, werden Sie KI-gestützte Programmiertools auch sinnvoller in realen Softwareprojekten einsetzen können.

Zentrale Features

Es finden sich viele KI-gestützte Programmiertools im Markt, wie zum Beispiel GitHub Copilot, Tabnine, CodiumAI oder Amazon CodeWhisperer. Die Firmen hinter den Produkten versuchen, ihres als das jeweils Beste mit besonderen Fertigkeiten zu verkaufen. Aber ein Großteil der Features sind in den meisten Tools vorhanden. In Tabelle 2-1 sind einige der zentralen Features aufgeführt.

Tabelle 2-1: Gemeinsame Features von KI-gestützten Programmiertools

Feature	Beschreibung
Codevorschläge	Macht Codevorschläge anhand von Kommentaren und dem Dateikontext; schlägt einzelne Zeilen oder ganze Funktionen vor.
Kontextsensitive Vervollständigung	Bietet kontextsensitive Codevervollständigung an, die auf der ganzen Codebasis oder auf Teilen davon basiert; macht auch Vorschläge, die den Code besser machen sollen.

→

Tabelle 2-1: Gemeinsame Features von KI-gestützten Programmiertools (Fortsetzung)

Feature	Beschreibung
Testerzeugung	Analysiert Code, um sinnvolle Tests zu erzeugen, Codeverhalten abzubilden und Spezialfälle zu berücksichtigen, die die Zuverlässigkeit der Software vor der Auslieferung sicherstellen.
User-IDE-Interaktion	Aktiviert automatisch Ratschläge, wenn Code in der IDE eingetippt wird – User können mit dem Code per Chat interagieren.
Codeanalyse	Analysiert Codeschnipsel, Docstrings und Kommentare, um zuverlässige Vorhersagen über den Code zu treffen und fragwürdigen Code zu kennzeichnen.
Bugs erkennen und beheben	Erkennt mögliche Fehler im Code und schlägt Lösungsmöglichkeiten vor.
Autodokumentation von Code	Fügt automatisch Docstrings hinzu und verbessert die Codedokumentation.
Automatisieren von Routineaufgaben	Hilft bei der Codeerstellung für Routineaufgaben oder Zeitfresser, unbekannte APIs oder SDKs und andere gängigen Coding-Szenarien, zum Beispiel Dateioperationen oder Bildverarbeitung.
Optimieren des API- und SDK-Einsatzes	Hilft dabei, APIs und SDKs korrekt und effektiv einzusetzen.
Finden und Verwenden von Open Source	Hilft bei der Suche nach Open-Source-Code und -Bibliotheken und deren Einsatz.

Die Liste in Tabelle 2-1 ist bei Weitem nicht vollständig – die Tools entwickeln sich rasend schnell weiter. Diese Systeme können einem beim Entwickeln sehr stark weiterhelfen, vor allem durch Codevorschläge und kontextsensitive Vervollständigung. Diese Features schauen wir uns gleich genauer an.

Codevorschläge und kontextsensitive Vervollständigung versus Smart Code Completion

Die Magie der *Smart Code Completion*, auch bekannt als Autovervollständigen oder unter dem Microsoft-Begriff IntelliSense, ist in vielen IDEs vorhanden. Sie hilft beim Entwickeln, indem sie kleine Codeschnipsel vorschlägt, einsetzt und hervorhebt, während man etwas eintippt. Diese Technologie gibt es im Prinzip schon seit den späten 1950ern mit dem Aufkommen von Rechtschreibprüfungen.

Der Durchbruch kam Mitte der 1990er-Jahre. Microsoft Visual Basic 5.0 brachte Vorschläge und Vervollständigen in Echtzeit mit, wobei es vor allem um einfache Syntax und Funktionssignaturen ging. Das hat die Produktivität deutlich verbessert und dabei geholfen, Fehler zu vermeiden.

Sie fragen sich jetzt vielleicht, wie sich so etwas wie IntelliSense gegen KI-gestützte Programmiertools behaupten kann. Schließlich greift IntelliSense auch in gewissem Maße auf KI und ML (Machine Learning) zurück.

Trotzdem gibt es einen wichtigen Unterschied. KI-gestützte Tools werden durch generative KI befeuert. Sie liefern nicht nur Code zurück, sondern viel mehr – Dokumentation, Planungsdokumente, hilfreiche Ratschläge und anderes. Dank der generativen KI können diese Tools für Menschen gedachte Texte auf Basis des bestehenden Kontexts erstellen, anpassen und verstehen. Das macht sie so gut beim Übersetzen, Zusammenfassen, Analysieren, Modellieren von Themen und Beantworten von Fragen. Die Interaktion mit diesen Tools fühlt sich manchmal so an, als spräche man mit seinem Code. Mit einem LLM im Bauch können die Tools aus Ihren Eingaben Kontextverschiebungen und Absichten erkennen.

Compiler versus KI-gestützte Programmiertools

Um KI-gestützte Programmiertools besser zu verstehen, hilft es, sich anzuschauen, was Compiler alles tun. Dies sind die wichtigsten Schritte eines Compilers:

Lexikalische Analyse (Tokenization)
: Der Compiler agiert wie ein Sprachlehrer, er teilt Ihren Code in Tokens auf.

Syntaxanalyse
: Hier prüft der Compiler, wie Ihre Tokens gruppiert sind. Er stellt sicher, dass Ihr Coding nicht nur die richtigen Befehle, sondern auch die richtige Struktur besitzt.

Semantische Analyse (Fehlerprüfung)
: Der Compiler stellt sicher, dass Ihr Code im Kontext der Programmiersprache sinnvoll ist. Es geht nicht nur um die korrekte Syntax, sondern auch um die korrekte Bedeutung.

Zwischencodeerzeugung
: Hier beginnt der Code mit seiner Transformation. Der Compiler übersetzt Ihren High-Level-Code in eine Zwischenform. Das ist noch keine Maschinensprache, aber schon ein Schritt in diese Richtung.

Codeoptimierung
: In diesem Schritt dient der Compiler als Personal Trainer für Ihren Code – er macht ihn schlanker und effizienter. Der Zwischencode wird so angepasst, dass er schneller läuft und weniger Platz braucht.

Codegenerierung
: Dies ist die finale Transformation. Der Compiler wandelt den optimierten Zwischencode in Maschinencode oder Assemblercode um, den Ihre CPU verstehen kann.

Linken und Laden
: Das wird manchmal als Teil des Kompilierungsprozesses angesehen. Beim *Linken* werden verschiedene Codemodule und Bibliotheken in einem einzelnen, ausführbaren Programm miteinander verbunden. Beim *Laden* wird das Programm dann zum Ausführen in den Speicher geladen.

KI-gestützte Programmiertools wie Copilot gehen ganz anders vor. Sie »verstehen« Programmiersprachen nicht so, wie es Compiler tun. Das ist schon okay. Dafür ist der Compiler ja da. Stattdessen kommt KI zum Einsatz, um Vermutungen zu treffen und Codeschnipsel vorzuschlagen – basierend auf Tonnen von bestehendem Code. Da die Tools auf Wahrscheinlichkeiten aufbauen, können die Vorschläge sehr unterschiedlich sein. Der Compiler wird dann diesen Code nehmen und ihn so umwandeln, dass der Rechner das Programm ausführen kann.

Manchmal übersehen KI-Tools so etwas Einfaches wie eine eckige Klammer, die ein Mensch oder ein Compiler auf Anhieb erkennt. Das liegt daran, dass LLMs auf der Vorhersage von Mustern basieren und nicht auf einer Compiler-Engine. Wenn etwas im Training nicht häufig vorkommt, wird es vielleicht auch nicht erkannt. Zudem schießen diese Tools eventuell über das Ziel hinaus und machen abhängig von der Situation komplexe Codevorschläge. Ja – KI-gestützte Programmiertools können ebenfalls abgelenkt werden.

Beim Erkennen von Fehlern sind KI-gestützte Programmiertools im Allgemeinen sehr effektiv, reichen aber bei Weitem nicht an die Ninja-ähnlichen Fähigkeiten eines Compilers heran. Trotzdem sind sie ausgesprochen leistungsfähig. Sie können zum Beispiel dabei helfen, nervige Syntaxfehler zu finden – ein fehlendes Semikolon, Tippfehler in Funktionsnamen, nicht zusammenpassende Klammern – und nebenbei auch noch die richtige Korrektur vorschlagen. Außerdem glänzen sie darin, Sie um verbreitete Coding-Fallstricke herumzuführen. Sei es, dass Sie daran erinnert werden, eine Datei auch wieder zu schließen, nachdem Sie sie geöffnet haben, oder dass effizientere Wege vorgeschlagen werden, ein Array zu durchlaufen – dieses Tool hält Ihnen den Rücken frei. Und bei logischen Fehlern können KI-gestützte Programmiertools überra-

schend aufschlussreich sein. Sie lösen vielleicht nicht jedes komplexe Problem, aber oft können sie alternative Wege oder Lösungen vorschlagen, an die Sie nicht gedacht haben, sodass Sie eher in die richtige Richtung weiterlaufen.

KI-Tools sind also hilfreich, um den Code besser zu machen, aber sie sind kein Ersatz für die umfassende Prüfung, die ein Compiler vornimmt, oder das scharfe Auge eines menschlichen Programmierers oder einer Programmiererin.

Diese Nachteile unterstreichen noch einmal, wie wichtig es ist, die Hilfe von KI-gestützten Tools mit der umfassenden Prüfung durch den Compiler und den Menschen vor der Tastatur zu verbinden. Denn schließlich wollen Sie sicher sein, dass Ihr Code nicht nur gut genug ist, sondern wirklich exakt und korrekt.

Fertigkeitsstufen

Im Oktober 2023 veröffentlichte Quinn Slack, CEO und Mitbegründer von Sourcegraph, einen interessanten Blogpost (*https://oreil.ly/SoDqc*). Er teilte die Welt der KI-gestützten Programmiertools wie Copilot auf und kam zu einer interessanten Art und Weise, über sie nachzudenken – die »Stufen von Code-KI«. Sein Schritt-für-Schritt-Framework macht es jedem leichter, zu erkennen, was diese KI-Tools können, und zu prüfen, ob die beeindruckenden Werbesprüche der Firmen auch halten, was sie versprechen. Abbildung 2-1 zeigt die verschiedenen Coding-Stufen.

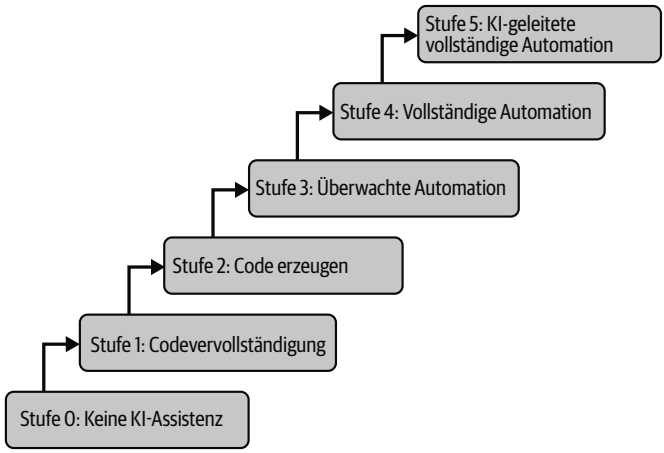

Abbildung 2-1: Programmiersysteme besitzen unterschiedliche KI-Fertigkeitsstufen.

Die ersten drei Stufen konzentrieren sich auf menschengemachten Code, bei denen die Programmierenden die Hauptdarsteller sind. Bei Stufe 0 gibt es kei-

nerlei KI-Assistenz – gutes, altes Coding. Man macht alles per Hand, und es ist keine KI in Sicht. Das ist die Grundlage, von der aus die KI loslegen kann.

Damit kommen wir zu Stufe 1 – Codevervollständigung. Hier kann KI mit ihrer Arbeit beginnen und einzelne Zeilen oder Codeschnipsel abhängig vom umgebenden Code vorschlagen. Dabei ist man beim Entwickeln immer die Hauptperson, steuert das Gesamtprogramm und nutzt KI zum Beschleunigen typischer Programmieraufgaben.

Mit Stufe 2 – Code erzeugen – bekommt die KI mehr zu tun. Hier werden längere Codeabschnitte generiert. Die KI kann beispielsweise APIs designen und sogar bestehenden Code korrigieren. Natürlich geschieht das alles, während ein Mensch ein Auge darauf hat. Bei dieser Stufe muss die KI die Codebasis und den Kontext kennen, sodass sie Code vorschlagen kann, der nicht nur korrekt ist, sondern auch gut passt.

Beginnend mit Stufe 3 – überwachte Automation – verschiebt sich die Hauptrolle bei der Programmierung hin zur KI. In dieser Stufe kümmert sich die KI um eine Reihe von Aufgaben, um vom Menschen vorgegebene allgemeiner formulierte Ziele zu erreichen. Dabei muss nicht bei jedem Schritt nachgefragt werden. Diese Stufe fühlt sich so an, als delegierte man Arbeit an einen Junior-Entwickler. Die KI ist bei dieser Stufe klug genug, um Fehler auszumerzen, neue Features zu bauen und Systeme miteinander zu verbinden. Dabei wird das menschliche Gegenüber immer wieder einbezogen, um Aspekte zu klären.

Bei Stufe 4 – vollständige Automation – geht die KI noch einen Schritt weiter. Hier kümmert sie sich ganz allein um komplexe Aufgaben, ohne dass Menschen abschließend ihre Zustimmung zum Code geben müssen. Denken Sie an das Vertrauen, dass Sie einem richtig guten Entwickler oder einer erfahrenen Entwicklerin entgegenbringen, wenn Sie eine CEO oder ein Produktmanager wären. Das ist die Art von Beziehung, die diese Stufe anstrebt. Die KI reagiert nicht mehr einfach nur. Sie kümmert sich proaktiv um den Code und erkennt und behebt Fehler, sobald diese entstehen.

Und schließlich gibt es auch noch Stufe 5 – KI-geleitete vollständige Automation. Diese Stufe ist noch mal etwas ganz anderes. Die KI folgt nicht mehr nur menschlichen Anweisungen, sondern sie hat ihre eigenen Ziele und versucht nur noch, eine Belohnungsfunktion zu erfüllen. Stellen Sie sie sich als Agent in einer Welt vor, in der sich die KI gegen andere Agenten behaupten muss. Klar, das klingt ein bisschen wie Science-Fiction, aber so schnell, wie es im Moment vorangeht, ist nicht auszuschließen, dass wir diese Stufe noch erleben werden.

Aktuell schaffen es Tools wie Copilot mehr oder weniger auf Stufe 3. Es ist schwierig, die genaue Stufe festzulegen, aber das Framework von Quinn Slack ist ein ziemlich guter Rahmen, um die Technologie und ihre wichtigsten Interaktionen einzuordnen. Und eines ist sicher: Die Weiterentwicklung der Technologie verlangsamt sich nicht – es geht immer noch ziemlich schnell voran.

Generative KI und Large Language Models (LLMs)

Für den Einsatz KI-gestützter Programmiertools müssen Sie die Technik der generativen KI nicht bis ins letzte Detail durchschaut haben. Aber es kann ziemlich praktisch sein, zumindest ein grobes Verständnis davon zu haben. So können Sie die Antworten, Möglichkeiten und Beschränkungen dieser Tools besser einschätzen.

Transparenz ist hier nicht nur ein Modewort. Damit Sie neue Technologie wirklich sinnvoll einsetzen können, ist es wichtig, sich darüber im Klaren zu sein, was unter der Motorhaube passiert. Denn bei ihrem Einsatz geht es immer auch um Vertrauen. In der Programmierwelt sind Zuverlässigkeit und Verantwortlichkeit nicht nur nette Extras – sie sind von grundlegender Wichtigkeit.

In den folgenden Abschnitten werden wir uns generative KI und LLMs (oberflächlich) anschauen, um Ihnen ein klareres Bild zu vermitteln.

Evolution

Die Wurzeln der generativen KI liegen viele Jahrzehnte in der Vergangenheit – eines der frühesten Beispiele ist ELIZA, der richtungsweisende Chatbot, der am Massachusetts Institute of Technology von Professor Joseph Weizenbaum Mitte der 1960er-Jahre zum Leben erweckt wurde. ELIZA wurde so entworfen, dass er Gespräche mit einem Psychotherapeuten simuliert (Sie finden ihn immer noch online (*https://oreil.ly/MbLP8*)). Klar, er war sehr simpel, lief auf einem regelbasierten Algorithmus und spiegelte größtenteils nur die menschlichen Eingaben wider.

Trotzdem fanden viele Menschen eine Unterhaltung mit ELIZA angenehmer als mit einem echten Therapeuten oder einer Therapeutin, und manche wurden sogar dazu gebracht, zu glauben, sie würden wirklich mit einem lebenden Wesen kommunizieren. Diese kuriose Situation, auch als »ELIZA-Effekt« bekannt, zeigte, wie schnell Menschen glauben, dass ein Computerprogramm ein menschenähnliches Verständnis erlangt.

Der Zug der generativen KI ist allerdings nicht wirklich schnell vorangekommen. Die technische Ausrüstung in ihrem Kern war ziemlich rudimentär, und es ging nur langsam voran. Aber in den 2010ern kam dann die Wende: Die Technologiewelt bot nun massive Rechenleistung, schicke Hardwaresysteme, beispielsweise GPUs (*Graphics Processing Units*), Unmengen wertvoller Daten und das Optimieren ausgefeilter Modelle wie zum Beispiel beim Deep Learning. Und damit konnte generative KI auf die Überholspur wechseln. Während ihrer Weiterentwicklung entstanden verschiedene Methoden:

Variational Autoencoders (VAEs)

Diese Technologie debütierte im Jahr 2013 durch Diederik P. Kingma und Max Welling in ihrem Artikel »Auto-Encoding Variational Bayes« (*https://arxiv.org/abs/1312.6114*). Ihr VAE-Modell besteht aus einem niedrigdimensionalen latenten Raum aus komplexeren, höherdimensionalen Daten ganz ohne Überwachung. Auch eine Encoder-Decoder-Struktur gehört dazu. Wenn wir von *höherdimensionalen Daten* reden, meinen wir Daten mit vielen Features, die jeweils eine eigene Dimension bilden – stellen Sie sich ein 28×28-Pixel-Bild in einem 784-dimensionalen Raum vor. Der niedrigdimensionale latente Raum ist dann wie eine kompakte Version dieser Daten, in der die wichtigen Informationen stecken, während die zusätzlichen Dimensionen weggeschnitten wurden. Das ist wichtig, weil damit die Rechenlast geringer wird, man den Fluch der Dimensionalität bekämpft und die Daten leichter visualisiert und interpretiert werden können. Dieser Sprung von einem höherdimensionalen zu einem niedrigdimensionaleren Raum wird als *Dimensionsreduktion* bezeichnet, und die Daten werden auf ihren Kern reduziert. Anders als bei den verwandten klassischen Autoencodern, die einen einzelnen Wert für jedes latente Attribut ausgeben, liefert Ihnen der Encoder in einem VAE eine Wahrscheinlichkeitsverteilung. Der Decoder wählt dann Samples aus dieser Verteilung, um die Daten wieder aufzubauen. Dieser Kniff, statt eines einzelnen Werts einen Bereich von Daten im latenten Raum anzubieten, öffnet die Tür zum Erstellen neuer Daten oder Bilder.

Generative Adversarial Networks (GANs)

Generative Adversarial Networks wurden 2014 von Ian Goodfellow und seinem Team vorgestellt (*https://arxiv.org/abs/1406.2661*). Dabei handelt es sich um eine Klasse von KI-Algorithmen, die beim unüberwachten maschinellen Lernen (*Unsupervised Machine Learning*) eingesetzt werden. Im Kern eines GAN stecken zwei neuronale Netze, *Generator* und *Discriminator*, die sich wie in einem Spiel ein Wettrennen liefern. Der Generator gibt neue Datensätze aus, und der Discriminator muss entscheiden, ob diese real oder gefälscht sind. Mit jeder Runde wird der Generator besser und schafft Daten, die von realen Instanzen weniger zu unterscheiden sind. Dieses clevere Setup hat viele neue Möglichkeiten eröffnet und KIs geschaffen, die realistische Bilder, Stimmaufzeichnungen und vieles mehr erzeugen kommen.

Diese Arten generativer KI waren wichtige Bausteine für das Transformer-Modell – den wirklichen Durchbruch, der die mächtigen LLMs hat Realität werden lassen.

Das Transformer-Modell

Bevor Transformer ihren großen Auftritt hatten, waren *Recurrent Neural Networks* (RNNs) für die natürliche Sprachverarbeitung (*Natural Language Processing*, NLP) das Mittel der Wahl. RNNs wurden dazu entwickelt, sequenzielle oder Zeitseriendaten zu verarbeiten. Sie enthalten einen verborgenen Status, um sich an Informationen aus vorherigen Schritten einer Sequenz zu erinnern – sehr praktisch für Sprachmodellierung, Spracherkennung und Sentiment-Analyse. Die RNNs gehen das Ganze Schritt für Schritt an, verarbeiten immer nur ein Element der Sequenz gleichzeitig und aktualisieren ihren verborgenen Status abhängig vom aktuellen Eingabeparameter und den zuvor verarbeiteten Elementen – daher der Begriff *Recurrent*. Wenn sie es aber mit langen Sequenzen zu tun haben, bekommen sie Probleme durch verschwindende oder explodierende Gradienten. Das machte es für sie schwierig, langfristige Beziehungen in den Daten zu berücksichtigen.

Da kommen nun die Transformer ins Spiel, die alles auf den Kopf stellen. Statt sich wie RNNs Schritt für Schritt vorzuarbeiten, gehen Transformer die Daten parallel durch und nutzen Attention-Mechanismen, um Beziehungen zwischen verschiedenen Elementen der Eingabesequenz im Griff zu behalten – egal wo diese erwähnt sind. Durch diesen Architekturwechsel können Transformer problemlos sowohl mit kurzen als auch mit langen Sequenzen umgehen. Zudem werden die Gradientenprobleme umgangen, und ihre Fähigkeit zur parallelen Verarbeitung passt gut zu ausgefeilten Chiparchitekturen wie denen von GPUs (*Graphics Processing Units*) oder TPUs (*Tensor Processing Units*).

Ashish Vaswani und seine Kollegen und Kolleginnen bei Google haben den Transformer geschaffen und den Kern der Architektur im Jahr 2017 im bahnbrechenden Artikel »Attention Is All You Need« (*https://arxiv.org/abs/1706.03762*) veröffentlicht. Abbildung 2-2 zeigt die wichtigsten Elemente des Modells.

Das Transformer-Modell ist wie ein brillanter Linguist, der gut darin ist, die Feinheiten von Sprache aufzudecken. Seine Magie entfaltet sich in zwei zentralen Phasen: Encoding und Decoding. Jede besitzt ihren eigenen Satz an Schichten. Während der *Encoding*-Phase liest und versteht das Modell den Eingabetext so ähnlich, wie eine Linguistin einen Satz in einer fremden Sprache verstehen würde. In der *Decoding*-Phase generiert das Modell dann einen neuen Textabschnitt oder eine Übersetzung basierend auf dem Verständnis, das es in der Encoding-Phase gewonnen hat – so wie ein Linguist diesen Satz in Ihre Muttersprache übersetzen würde.

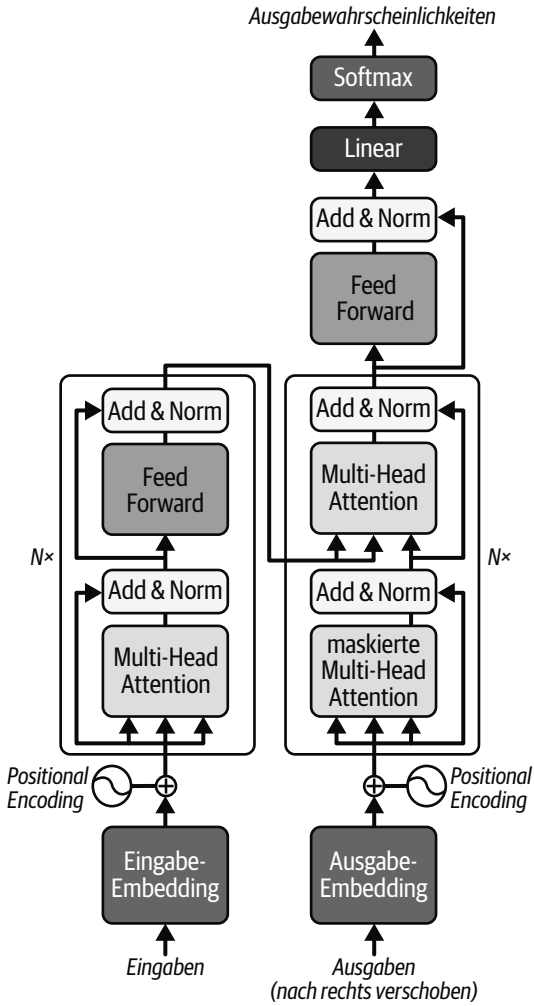

Abbildung 2-2: Die Architektur des Transformer-Modells als Kern von LLMs

Im Kern des Transformer steckt ein Mechanismus namens *Attention*, mit dem man die Relevanz jedes Worts eines Satzes im Verhältnis zu den anderen Wörtern bewerten kann. Er weist jedem Wort einen Attention-Score zu. Nehmen wir beispielsweise den Satz »Die Katze saß auf der Matte«. Konzentriert sich das Modell auf das Wort *saß*, erhalten die Wörter *Katze* und *Matte* aufgrund ihrer direkten Beziehung zur Aktivität des Sitzens eventuell höhere Attention-Scores.

Ein bemerkenswertes Feature dieses Modells ist der *Self-Attention-Mechanismus*. Dieser erlaubt es, einen gesamten Satz zu betrachten, die Beziehungen zwischen Wörtern zu verstehen und sich diese Beziehungen über lange Textab-

schnitte zu merken. Damit erhält der Transformer eine Form von Langzeitgedächtnis, indem sich das Modell auf alle bisher aufgetauchten Wörter oder *Tokens* (ganze Wörter oder Teile eines Worts) fokussieren und damit den größeren Rahmen verstehen kann.

Trotz dieser Fähigkeiten fehlte dem Transformer zunächst die Möglichkeit, die Reihenfolge der Wörter in einem Satz zu erkennen, was für ein Verständnis der Bedeutung entscheidend ist. Hier kommt das *Positional Encoding* ins Spiel. Es agiert wie ein GPS, um das Modell mit Informationen über die Position jedes Worts im Satz zu versorgen, und es hilft damit, einen Satz wie »Die Katze jagt die Maus« von »Die Maus jagt die Katze« unterscheiden zu können.

Der Transformer besitzt nun zudem noch einen *Multi-Head-Attention-Mechanismus*. Stellen Sie ihn sich vor, als würde das Modell mehrere Augenpaare besitzen, von denen jedes den Satz aus einem anderen Blickwinkel betrachtet und sich auf andere Aspekte oder Beziehungen zwischen den Wörtern konzentriert. So fokussiert sich beispielsweise ein Paar auf das Verstehen von Aktionen, ein anderes identifiziert Charaktere, und ein drittes erkennt Orte. Dieser Multi-View-Ansatz ermöglicht dem Transformer, den Text besser zu verstehen.

Außerdem greift jede Phase des Transformer auf Schichten eines *Feedforward Neural Network* zurück – eines einfachen Netzwerks, das beim Verarbeiten der Beziehungen zwischen Wörtern hilft. Damit wird das Verständnis des Texts und das Generieren weiter verbessert.

Ein Transformer ist ein vortrainiertes Modell. Es wurde schon mit wirklich vielen Daten trainiert und kann dann genutzt oder weiter optimiert werden. Nach dem Vortraining lässt sich das Modell über eine API ansprechen, sodass es in diversen Aufgaben der Sprachverarbeitung direkt eingesetzt werden kann. Firmen oder Einzelpersonen können das Modell schnell in ihre Systeme einbinden, wie zum Beispiel KI-gestützte Programmieranwendungen. Zudem lässt sich das vortrainierte LLM weiter in spezifischen Domains verbessern, wie zum Beispiel in der Analyse von medizinischen oder juristischen Texten, indem man es mit weiteren domainspezifischen Daten füttert. Damit muss nicht mehr ein komplexes Sprachmodell von Grund auf neu erstellt werden, was viel Zeit, Aufwand und Ressourcen spart. Das vortrainierte Modell mit seinem grundsätzlichen Sprachverständnis agiert als Sprungbrett für die Entwicklung generativer KI-Anwendungen.

Die zwei wichtigsten Arten von Transformer-Systemen sind *Generative Pretrained Transformer* (GPT) und *Bidirectional Encoder Representations from Transformers* (BERT). GPT ist ein Tool von OpenAI, das ideal geeignet ist für das Erzeugen von Text, das Zusammenfassen von Informationen und das

Übersetzen von Sprachen. Es basiert auf einer autoregressiven LLM-Architektur. Der Text wird also generiert, indem bei jedem Wort sorgfältig die bisherige Ausgabe betrachtet wird – wie ein Erzähler, der ein Narrativ Wort für Wort erschafft. Das Modell ist so gut, weil es mit einer gigantischen Menge an Textdaten trainiert wurde. GPT nutzt den Decoder zum Generieren von Inhalten.

Das Bauen und Betreiben eines LLM ist teuer. Anfang 2023 sorgte GitHub Copilot laut *Wall Street Journal* im Durchschnitt für einen Verlust von mehr als 20$ pro Monat und User (*https://oreil.ly/D2NiB*). In manchen Situationen sorgten einige User sogar für einen Verlust von 80 $ pro Monat. Wenn aber die Infrastruktur in den kommenden Jahren für generative KI skaliert wird, sollten die Kosten pro User sinken.

BERT verwendet andererseits einen Autoencoding-Ansatz. Damit lässt sich der Kontext von Wörtern in einem Satz sehr gut verstehen, was dazu führt, dass die Nuancen und Bedeutungen von Sprache besonders gut erfasst werden können. Google hat BERT 2018 als Open-Source-Projekt entwickelt. Seitdem sind viele Variationen und Verbesserungen des Kernmodells entstanden.

Bei KI-gestützten Programmieranwendungen ist GPT das wichtigste Transformer-Modell. Es hat gezeigt, dass es Code basierend auf dem vom Programmierer gelieferten Kontext effizient vorhersagen und automatisch vervollständigen kann.

OpenAI Playground

Der OpenAI Playground (*https://platform.openai.com*) ist eine generative KI-Sandbox, die den Zugriff auf diverse von OpenAI entwickelte Modelle ermöglicht. Diese Modelle lassen sich über eine intuitive grafische Oberfläche anpassen.

Der OpenAI Playground erleichtert es, die Stärken und Schwächen der verschiedenen LLMs zu verstehen. Zudem sind so Echtzeittests und Anpassungen des Modells als Reaktion auf unterschiedliche Eingaben möglich – zum Beispiel die Temperatur.

Allerdings möchte OpenAI für das Verwenden der Plattform Geld sehen. Die Kosten basieren auf der Menge der genutzten Tokens (siehe Tabelle 2-2). Beachten Sie, dass sich die Preise regelmäßig ändern. Bisher sind sie aber immer nur gesunken.

Tabelle 2-2: Die Kosten für OpenAI-LLMs

Modell	Eingabe	Ausgabe
GPT-4/8K Context	$0,03/1K Tokens	$0,06/1K Tokens
GPT-4/32K Context	$0,06/1K Tokens	$0,12/1K Tokens
GPT-3.5-Turbo/4K Context	$0,0015/1K Tokens	$0,002/1K Tokens
GPT-3.5-Turbo/16K Context	$0,003/1K Tokens	$0,004/1K Tokens

Nehmen wir beispielsweise an, wir nutzen das LLM »GPT-4/8K Context«. Sie haben einen Prompt mit 1.000 Tokens, und die Antwort des Modells besteht aus 2.000 Tokens. Dann wird Sie das 3 US-Cent für die Eingabe und 12 US-Cent für die Ausgabe kosten.

Registrieren Sie sich erstmals für einen OpenAI-Account, erhalten Sie ein Guthaben von 5 $, das für den OpenAI Playground oder API-Aufrufe eingesetzt werden kann.

Tokens

Befassen wir uns ein wenig genauer mit Tokens. OpenAI bietet ein Tool namens Tokenizer (*https://platform.openai.com/tokenizer*) an. In Abbildung 2-3 sehen Sie ein Beispiel, bei dem ich den folgenden Text zur Analyse eingegeben habe:

Input: ChatGPT ist unglaublich! ? Ich liebe es.

Abbildung 2-3: Der OpenAI Tokenizer zeigt die Tokens für einen Textausschnitt an.

Beim Tokenizing – hervorgehoben durch die Farben – wird das Wort *ChatGPT* in drei Tokens aufgeteilt (Chat, G und PT). Das Wort unglaublich besteht aus vier Tokens, dazu das Ausrufezeichen. Das Emoji wird auf drei Tokens umbrochen. Jedes Satzzeichen ist ein Token. Leerzeichen gehören zum folgenden Wort.

Der Tokenizer ist für GPT-3, GPT-3.5 und GPT-4 nutzbar. Denken Sie daran, dass das Tokenizing bei LLMs oft sehr unterschiedliche Ergebnisse liefern kann.

Als Faustregel gilt, dass 1.000 Tokens ungefähr 750 (englischen) Wörtern entsprechen.

Die Plattform einsetzen

Rufen Sie den OpenAI Playground auf, erhalten Sie Zugriff auf ein Dashboard (siehe Abbildung 2-4).

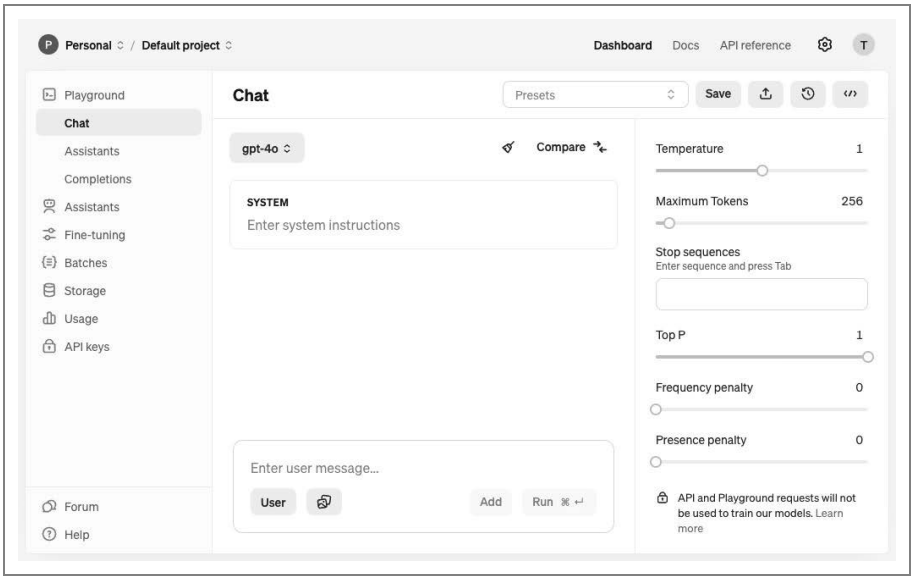

Abbildung 2-4: Der OpenAI Playground bietet ein Dashboard mit Tipps, Ressourcen und Interaktionsbereichen.

In der Mitte des Bildschirms findet sich der zentrale Bereich für die Interaktion mit einem LLM:

System
: Hier liefern Sie Kontext für das LLM, zum Beispiel: »Du bist ein Experte in Python-Programmierung.« Der System-Prompt ist die erste Nachricht in einer Session, und diese gibt den Rahmen für die Interaktion vor. Ein Anpassen des System-Prompts erlaubt eine genauere Steuerung des Verhaltens des Modells in der Unterhaltung – das kann besonders nützlich sein, um sicherzustellen, dass es sich innerhalb der gewünschten Parameter oder Kontexte bewegt.

User
: Dies ist die zentrale Anweisung des Prompts. Hier können Sie das LLM beispielsweise bitten, eine Programmieraufgabe zu erledigen.

Add
: Auf diesem Weg können Sie eine fortlaufende Unterhaltung mit dem LLM führen.

Probieren wir es mit einem Beispiel. Stellen Sie sich vor, Sie arbeiten an einem Python-Projekt und haben Probleme damit, zu verstehen, wie Sie die Tkinter-Bibliothek implementieren sollen, um eine Benutzereingabe zu erhalten. Sie können Folgendes eingeben:

> *System-Nachricht: Du bist ein Python-Experte, der auf Tkinter spezialisiert ist.*
>
> *User-Nachricht: Ich will mit Tkinter eine einfache Oberfläche erstellen, um den Namen und das Alter des Benutzer zu erhalten. Wie kann ich das machen?*

Das LLM wird den Code dafür generieren. Nun wollen Sie aber noch sicherstellen, dass der Benutzer mindestens 18 Jahre alt ist. Also drücken Sie auf *Add* und geben ein: »Wie kann ich sicherstellen, dass der Benutzer älter als 18 ist?«

Das LLM wird auch dafür Code liefern, der das Alter prüft und eine Meldung ausgibt, wenn es unter 18 liegt.

Das ist zugegebenermaßen das Gleiche wie der Einsatz von ChatGPT – nur mit mehr Struktur. Und Sie haben hier mehr Anpassungsmöglichkeiten. Auf der rechten Seite und oberhalb des zentralen Bereichs finden Sie folgende Möglichkeiten:

Model

Sie können aus einer Vielzahl von Modellen auswählen und sogar Ihre eigenen optimierten LLMs nutzen, um sicherzustellen, dass sich das Modell an den spezifischen Bedürfnissen Ihres Codes orientiert. Mehr zum Optimieren eines Modells finden Sie in der OpenAI API Documentation (*https://oreil.ly/L3y09*).

Temperature

Damit können Sie an der Zufälligkeit oder der Kreativität der erzeugten Inhalte drehen. Der Wertebereich liegt zwischen 0 und 2. Je niedriger der Wert, desto deterministischer und fokussierter sind die Antworten. In Tabelle 2-3 sehen Sie vorgeschlagene Temperaturwerte für verschiedene Entwicklungsaufgaben.

Nutzen Sie einen recht hohen Wert für die Temperatur, können die Ergebnisse unsinnig werden. Hier ein Beispiel-Prompt bei einem Wert von 2:

Prompt: Wie kann man in Python die Daten aus einer CSV-Datei in eine MySQL-Datenbank übertragen?

Abbildung 2-5 zeigt die Ausgabe. Wie Sie sehen, ist das ziemlich unsinnig!

Tabelle 2-3: Vorgeschlagene Temperaturwerte für bestimmte Arten von Programmieraufgaben

Aufgabenbereich	Temperaturwert	Beschreibung
Code generieren	0.2 bis 0.3	Deterministischeren und exakteren Code sicherstellen, der sich an die üblichen Konventionen für zuverlässige und verständliche Ergebnisse hält.
Code Review	0.2 oder weniger	Fokus auf bekannte Best Practices und Standards für ein genaues Feedback.
Bug Fixing	0.2 oder weniger	Generiert genauere und klarere Lösungen für gefundene Probleme.
Kreative Problemlösung	0.7 bis 1.0	Sucht in einem größeren Raum möglicher Lösungen – nützlich zum Brainstorming oder für das innovative Lösen von Problemen.
Lernen und Experimentieren	0.7 bis 1.0	Liefert eine größere Auswahl an Beispielen und Lösungen zum Verstehen verschiedener Ansätze.
Datenanalyse und Visualisierung	0.2 oder weniger	Erzeugt genauere und sinnvollere Visualisierungen oder Analysen.
Optimierungsaufgaben	variierend	Erlaubt das Finden einer Balance zwischen Erforschen (höhere Temperatur) und Ausnutzen (niedrigere Temperatur) für eine effiziente Lösung.

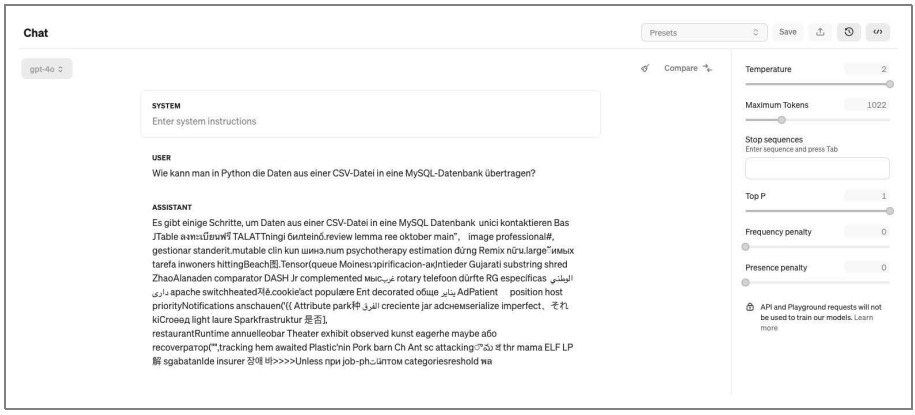

Abbildung 2-5: Mit einer Temperatur von 2 sind die Ergebnisse des LLM meist unsinnig.

Schauen wir uns noch die anderen Möglichkeiten zum Optimieren an:

Maximum Tokens
: Dies ist die maximale Anzahl von Tokens, die für den generierten Inhalt genutzt wird. Darin enthalten sind die Tokens für den Prompt und für die Antwort. Das Verhältnis von Tokens und Inhalt hängt vom eingesetzten Modell ab.

Stop sequences
: Das gibt einen Punkt an, an dem das LLM keinen weiteren Text mehr erzeugen sollte. Sie können einen spezifischen String oder eine Folge von Zeichen angeben, die dem Modell signalisieren, den Prozess zu beenden, wenn sie im generierten Text gefunden werden.

Top P
: Diese Technik, auch als *Nucleus Sampling* bekannt, wählt Wörter aufgrund einer kumulierten Wahrscheinlichkeitsgrenze *p* aus, die zwischen 0 und 1 liegen kann. Einfacher gesagt: Statt immer aus den wenigen wahrscheinlichsten nächsten Wörter zu wählen, berücksichtigt das Modell einen größeren oder kleineren Bereich möglicher nächster Wörter abhängig vom angegebenen *p*-Wert. Ein niedrigerer *p*-Wert führt zu einem kleineren, fokussierteren Satz an Wörtern, aus denen gewählt wird, was einen vorhersagbareren und einheitlicheren Text schafft. Ein höherer *p*-Wert erlaubt andererseits mehr mögliche nächste Wörter, was zu abwechslungsreicheren und kreativeren Texten führt.

Frequency penalty
: Das hilft dabei, ein häufiges Problem bei LLMs zu bekämpfen – sich wiederholende Phrasen oder Sätze. Der Wert kann zwischen 0 und 2 liegen. Je höher der Wert ist, desto seltener kommt es zu Wiederholungen. Werte größer als 1 können aber zu unvorhersagbaren und sogar unsinnigen Texten führen.

Presence penalty
: Auch dieser Wert liegt zwischen 0 und 2. Ein höherer Wert erlaubt dem LLM, eine größere Spannbreite von Tokens aufzunehmen, was wiederum zu einem abwechslungsreicheren Vokabular und mehr unterschiedlichen Konzepten führt.

Bei *Frequency penalty*, *Presence penalty* und *Top P* empfiehlt OpenAI die Wahl nur eines der Parameter, um das Modell für Ihre Aufgabe zu optimieren. Aber seien Sie ruhig experimentierfreudig. Es gibt dabei aufgrund der ungeheuren Komplexität keine festen Regeln.

LLMs bewerten

Das Bewerten von LLMs ist keine leichte Aufgabe. Diese Ungetüme sind oft so undurchsichtig, dass es unmöglich scheint, sie zu verstehen. Der Wettbewerb unter den KI-Firmen macht das häufig noch schlimmer. Es ist üblich geworden, nur wenige Details zu den Datensätzen herauszurücken, mit denen die Modelle trainiert wurden, ebenso erfährt man nicht viel über die Anzahl der Parameter zum Optimieren des Verhaltens und zur Hardware, auf der das alles läuft.

Es gibt aber dank einiger Forschenden in Stanford auch Licht am Ende des Tunnels. Sie haben ein Scoring-System (*https://oreil.ly/FoVAr*) namens *Foundation Model Transparency Index* geschaffen, um die Offenheit von LLMs zu fördern. Dieser Maßstab, gebildet aus rund 100 Kriterien, ist ein Versuch, die trüben Gewässer der LLM-Transparenz etwas klarer zu machen.

Das Ranking basiert auf einer Prozentskala, und in Tabelle 2-4 sehen Sie die Ergebnisse. Leider sind diese nicht allzu vielversprechend. Kein großes LLM erreicht laut der Forschungsgruppe auch nur annähernd »adäquate Transparenz«, und der Durchschnittswert liegt bei nur 37%.

Die Flexibilität von LLMs in Bezug auf die diversen Bereiche und Aufgaben – wie zum Beispiel Softwareentwicklung – ist ein bemerkenswerter Vorteil. Sie verkompliziert aber den Bewertungsprozess, da domainspezifische Bewertungsmetriken und Benchmarks erforderlich sind, um die Effektivität und Sicherheit eines Modells in jeder spezifischen Anwendung sicherstellen zu können.

Tabelle 2-4: Rankings der besten LLMs in Bezug auf die Transparenz ihrer Modelle[1]

Firma	Modell	Rang
Meta	LLaMA 2	54%
BigScience	BLOOMZ	53%
OpenAI	GPT-4	48%
Stability.ai	Stable Diffusion 2	47%
Google	PaLM 2	40%
Anthropic	Claude 2	36%
Cohere	Command	34%
AI21Labs	Jurassic-2	25%
Inflection	Inflection-1	21%
Amazon	Titan Text	12%

Trotz all der Hindernisse hier ein paar Metriken, die Sie sich zum Bewerten von LLMs anschauen können:

BERTScore
: Diese Metrik ist dazu gedacht, Textgenerierungsmodelle zu bewerten, indem generierter Text über BERT-Embeddings mit Referenztext verglichen wird. Auch wenn sie vor allem für natürlichsprachigen Text gedacht ist, lässt sich die Metrik für Codegenerierungsaufgaben erweitern beziehungsweise anpassen – insbesondere wenn der Code in natürlicher Sprache annotiert oder kommentiert ist.

Perplexity
: Das ist eine gebräuchliche Metrik für das Bewerten von Wahrscheinlichkeitsmodellen wie LLMs. Sie ermittelt, wie gut die vom Modell vorhergesagte Wahrscheinlichkeitsverteilung zur tatsächlichen Verteilung der Daten passt. In Bezug auf das Generieren von Code bedeutet ein niedrigerer Perplexity-Wert, dass das Modell das nächste Token in einer Codefolge besser vorhersagen kann.

BLEU (Bilingual Evaluation Understudy)
: Ursprünglich für maschinelle Übersetzung entwickelt, wird BLEU auch bei der Codegenerierung genutzt, um den erzeugten Code mit Referenzcode zu vergleichen. Die Metrik berechnet *n*-gram-Genauigkeitswerte, um

1 Center for Research on Foundation Models, Foundation Model Transparency Index Total Scores 2023, *https://crfm.stanford.edu/fmti*

die Ähnlichkeit zwischen den generierten und Referenztexten zu vergleichen. Das wiederum kann dabei helfen, die syntaktische Korrektheit des erzeugten Codes zu bewerten. Ein höherer *n*-gram-Genauigkeitswert deutet auf eine bessere Übereinstimmung zwischen dem generierten und dem Referenztext für diese spezifische Folge von *n* Wörtern hin.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)
: Dies ist eine weitere Metrik, die aus der NLP übernommen wurde und dazu genutzt werden kann, Codegenerierungsmodelle zu bewerten. Sie berechnet die Überlappung von *n*-Grammen zwischen den generierten und den Referenztexten, was Einblicke darin geben kann, wie gut der generierte Code mit der erwarteten Ausgabe übereinstimmt.

MBXP (Most Basic X Programming Problems)
: Diese Benchmark ist spezifisch für das Bewerten von Codegenerierungsmodellen für mehrere Programmiersprachen entworfen worden. Sie nutzt ein skalierbares Umwandlungs-Framework, um Prompts und Test Cases aus den Originaldatensätzen in Zielsprachen zu transpilieren und damit eine umfassende mehrsprachige Bewertung von Codegenerierungsmodellen zu ermöglichen.

HumanEval
: Dies ist eine Benchmark zum Bewerten der Codegenerierungsmöglichkeiten von LLMs, indem ihre funktionale Korrektheit beim Synthetisieren von Programmen aus Docstrings ermittelt wird. Diese Benchmark ist für das fortlaufende Entwickeln und Verbessern von KI-Modellen zum Generieren von Code sehr wichtig. Unterschiedliche Modelle haben sich bei HumanEval auch unterschiedlich gut geschlagen, und mit einer erweiterten Version namens HUMANEVAL+ konnte vorher unerkannter falsch generierter Code von verbreiteten LLMs identifiziert werden.

Multilingual HumanEval (HumanEval-X)
: Dies ist eine Erweiterung der ursprünglichen HumanEval-Benchmark. Multilingual HumanEval bewertet die Codegenerierungs- und Übersetzungsfähigkeiten von LLMs für mehr als zehn Programmiersprachen. Dabei kommt ein Umwandlungs-Framework zum Transpilieren von Prompts und Test Cases aus Python in entsprechende Daten der Zielsprachen zum Einsatz, was eine umfassendere Benchmark für mehrsprachige Codegenerierung und Übersetzung ermöglicht.

Eine andere Möglichkeit zum Bewerten von LLMs ist ein Blick auf die Anzahl der Parameter – die in die Hunderte von Milliarden gehen können. Also je mehr, desto besser? Nicht unbedingt. Beim Bewerten sollte man etwas nuan-

cierter vorgehen. Die Kosten für das Skalieren der Parameter können in Bezug auf Rechenleistung und Energieeinsatz enorm sein. Das kann ein LLM zum Monetarisieren von Anwendungen unökonomisch machen. Zudem steigt mit der Anzahl der Parameter auch die Komplexität des Modells, was potenziell zu einem Overfitting führen kann. Ein *Overfitting* tritt dann ein, wenn das Modell viel zu gut lernt, sich an die Trainingsdaten anzupassen, dann aber beim Kontakt mit unbekannten Daten ins Stolpern gerät. Das beeinträchtigt seine Fähigkeit zum Generalisieren.

Ein weiteres Thema ist der Bedarf an großen und variantenreichen Trainingsdatensätzen, mit denen der unersättliche Hunger dieser Modelle nach Daten gestillt werden kann. Das Beschaffen und Kuratieren solch riesiger Datensätze ist nicht nur ressourcenintensiv, es bringt auch Herausforderungen in Bezug auf Datenschutz und Voreingenommenheiten mit sich. Zudem wird das Bewerten dieser Datenmonster mit einer zunehmenden Menge an Parametern immer schwieriger. Die Bewertungsmetriken müssen umfassender und variantenreicher sein, um die Performance des Modells für eine Heerschar an Aufgaben gut bestimmen zu können.

Das Optimieren eines Modells kann letztlich ein besserer Weg sein, um mehr aus ihm herauszuholen, ohne die Parametermenge des zugrunde liegenden LLM noch weiter steigern zu müssen.

Arten von LLMs

Es gibt diverse Arten von LLMs, und eine bekannte Kategorie sind die Open-Source-LLMs. Jeder kann sie verwenden, anpassen oder bereitstellen. Durch ihre Transparenz können Sie sehen, wie sie funktionieren. Zudem erlauben Open-Source-LLMs Entwicklerinnen und Entwicklern, gemeinsam an Innovationen zu arbeiten, Add-ons zu erstellen und natürlich nervige Bugs zu beheben.

Und das Beste ist, dass sie nichts kosten.

Aber Open-Source-LLMs sind nicht nur eitel Sonnenschein. Es gibt meist kein dediziertes Team, das nur dafür da ist, Fehler zu beheben oder für regelmäßige Updates zu sorgen. Stoßen Sie auf ein Problem, müssen Sie eventuell selbst die Ärmel hochkrempeln und in Foren nach Hilfe suchen.

Qualität und Performance von Open-Source-Modellen können stark schwanken. Und es gibt immer wieder Sicherheitsprobleme. Da alles einsehbar ist, finden Hacker eher einen Weg, böswilligen Code einzuschmuggeln. Seien Sie also vorsichtig.

Schließlich sind Anleitungen für die Benutzenden sowie eine Dokumentation nicht unbedingt die Stärke von Open-Source-LLMs. Manchmal fühlt es sich so an, als wären sie in Hieroglyphen geschrieben worden.

Tabelle 2-5 stellt einige der besten Open-Source-LLMs vor.

Tabelle 2-5: Gute Open-Source-LLMs

Modell	Entwickler	Parameter (in Milliarden)	Erwähnenswerte Features
GPT-NeoX-20B	EleutherAI	20	Mit dem »The Pile«-Datensatz trainiert; kann diverse NLP-Aufgaben handhaben, z. B. Story Generation, Chatbots oder Zusammenfassungen.
LLaMA 2	Meta	7 bis 70	Mit zwei Billionen Tokens trainiert; verdoppelt die Kontextlänge von LLaMA 1.
OPT-175B	Meta	175	Teil einer ganzen Modell-Suite; beim Training wurde weniger CO_2 verbraucht als bei GPT-3.
BLOOM	BigScience	176	Mit dem ROOTS-Korpus[2] trainiert; auf Transparenz designt mit offengelegten Trainingsdaten und Bewertungsmethoden.
Falcon-40B	Technology Innovation Institute (TII)	40	Mit einer Billion Tokens trainiert.
Dolly 2.0	Databricks	12	Basiert auf der Pythia-Modellfamilie von EleutherAI; bietet ChatGPT-ähnliche Interaktivität.
Mistral 7B	Mistral AI	7,3	Nutzt Grouped-Query und Sliding Window Attention; trainiert auf einem großen Datensatz und sticht besonders bei längeren Sequenzen hervor.
Mixtral 8X7B	Mistral AI	46,7	Sparse-Mischung von Expertenmodellen; führt die Inferenz wie ein 12,9-Milliarden-Modell durch, unterstützt mehrere Sprachen und ist bei vielen Aufgaben besonders gut, unter anderem beim Generieren von Code und beim Begründen.

Closed-Source- oder proprietäre LLMs agieren andererseits viel verschlossener. Sie behalten ihren Code, ihre Trainingsdaten und Modellstrukturen größtenteils für sich. Die Firmen, die diese komplexen Systeme entwickeln, haben aber im Allgemeinen Zugriff auf viel Geld. Tabelle 2-6 führt das Kapital auf, das von diesen Firmen im Jahr 2023 eingeworben wurde.

2 Responsible Open-science Open-collaboration Text Sources

Tabelle 2-6: Venture-Kapital von den größten LLM-Entwicklern

Firma	Funding
Anthropic	$1,25 Milliarden
OpenAI	$10 Milliarden
Cohere	$270 Millionen
Inflection AI	$1,3 Milliarden

Mit solchen Ressourcen können diese Firmen die besten Data Scientists an Bord holen und sich eine ausgefeilte Infrastruktur hinstellen. Daher ist die Performance ihrer LLMs oft State of the Art. Zudem sind sie auf eine entsprechende Größe hin gebaut, und sie berücksichtigen die Bedürfnisse von Unternehmen – zum Beispiel in Bezug auf Sicherheit und Datenschutz.

Der Nachteil ist die Sache mit dem Vertrauen. Woher holen diese Modelle ihre Antworten? Wie ist es mit Halluzinationen und Voreingenommenheiten? Antworten auf diese Fragen sind oft nicht ausgesprochen detailliert.

Und dann gibt es noch das Risiko, dass diese Riesen-KI-Firmen zu Monopolisten werden. In dem Fall wäre man als Kundin oder Kunde in deren Ökosystem gefangen. Und der letzte Punkt: Bei Closed-Source-LLMs besteht eher die Gefahr von Stillstand als bei Open-Source-Projekten, da sie eventuell nicht von den variantenreichen Einträgen und Kontrollen profitieren, die bei Open-Source-Projekten so interessant sind.

Bewerten von KI-gestützten Programmiertools

Es ist eventuell gar nicht so einfach, herauszufinden, mit welchem KI-gestützten Programmiertool man arbeiten soll. Sie müssen viele Faktoren gegeneinander abwägen – Genauigkeit, Chat-Features, Sicherheit, Geschwindigkeit und Benutzerfreundlichkeit. Manchmal kommt es letztendlich darauf an, was sich beim Arbeiten »richtig« anfühlt. Aber auch dann sind Ihnen vielleicht die Hände gebunden, wenn Ihr Arbeitgeber auf einem bestimmten System besteht.

Um ein Gespür dafür zu bekommen, was aktuell gerade in ist, können Sie sich den 2023 Developer Survey von Stack Overflow (*https://oreil.ly/nvqKY*) anschauen. Dabei wurden Bewertungen von nahezu 90.000 Programmierern und Programmiererinnen zu den beliebtesten Tools eingesammelt – zu sehen in Tabelle 2-7.

Tabelle 2-7: Das Ranking beliebter KI-gestützter Programmiertools[3]

KI-gestützte Programmiertools	Prozentwert
GitHub Copilot	54,77 %
Tabnine	12,88 %
Amazon CodeWhisperer	5,14 %
Snyk Code	1,33 %
Codeium	1,25 %
Wispr AI	1,13 %
Replit Ghostwriter	0,83 %
Mintlify	0,52 %
Adrenaline	0,43 %
Rubberduck AI	0,37 %

Hier sehen Sie, welche Tools es überhaupt so gibt. Wollen Sie eines auswählen, ist es sinnvoll, sich umzuhören und selbst ein paar auszuprobieren. Zum Glück bieten die meisten dieser Tools kostenlose Proberunden, sodass Sie mit ihnen experimentieren können, ohne sich gleich festlegen zu müssen.

Ein weiterer Aspekt, den Sie berücksichtigen sollten, ist die finanzielle Stabilität des Anbieters. Gibt es Venture-Kapital? Ohne dieses Kapital hat eine Firma eventuell nicht nur Probleme beim Wachsen, sondern auch, ihre Plattform innovativ zu halten. Es mussten schon einige KI-gestützte Programmierfirmen ihre Services abschalten, was beim Entwickeln zu echten Problemen führen kann. Kite war beispielsweise 2014 eine der ersten Firmen im Bereich KI-gestützter Programmiertools. Dennoch hat sie sich 2022 dafür entschieden, aus dem Projekt auszusteigen (*https://oreil.ly/Bnz9U*). Der Silberstreif am Horizont? Ein Großteil der Tool-Codebasis ist jetzt Open Source.

3 Stack Overflow, 2023 Developer Survey (*https://oreil.ly/0u7WZ*)

Zusammenfassung

In diesem Kapitel haben wir gezeigt, was bei generativer KI und LLMs hinter den Kulissen geschieht. Sie haben ein bisschen die faszinierende Geschichte kennengelernt (zum Beispiel ELIZA), und wir haben uns dann auf einen der größten Durchbrüche in der KI konzentriert – das Transformer-Modell. Außerdem haben wir den OpenAI Playground ausprobiert und gezeigt, wie man das LLM anpasst.

Zu den zentralen Punkten dieses Kapitels gehören Tokens, die Vorteile der Verwendung von vortrainierten Modellen, die Vor- und Nachteile des Skalierens von LLMs, Metriken wie Perplexity und BLEU-Scores sowie die Gegenüberstellung von Open Source und proprietären Modellen.

Inhalt