KI UND DATA, TEIL 3

Aus der Praxis für die Praxis

Sprachmodelle finden vielfältigen Einsatz. Ein Beispiel zeigt, was mit einem Prompt, einer strukturierten Ausgabe und einer Idee ohne großen Aufwand machbar ist.

Sprachmodelle (Language Models) sind aus der modernen KI-Landschaft nicht mehr wegzudenken, das hat der vorangegangene Artikel dieser Reihe verdeutlicht und mit aktuellen Entwicklungen untermauert [1]. Sprachmodelle können dabei weit mehr als nur Texte generieren; es gibt eine Vielzahl von Modellen, die ganz andere Funktionen erfüllen und auf unterschiedlichste Aufgaben spezialisiert sind, so etwa Coding-, Vision-, Speech- und Audio- sowie multimodale Modelle. Eine Plattform, die diese Vielfalt von Modellen besonders gut zugänglich macht, ist Hugging Face [2], siehe Kasten Ein Blick auf Hugging Face.

Listing 1: Die Einleitung des Prompts, um die Aufgaben initial zu beschreiben

```
Du bist Experte für die Erstellung von Themen zur Lernvorbereitung auf der
Grundlage von Inhalten eines fotografierten Dokuments (z. B. einer
Lehrbuchseite, einer Folie, eines handschriftlichen Dokuments). Deine
Aufgabe ist es, aussagekräftige Themen zu erstellen, die nur den sichtbaren
Inhalt enthalten. Falls Abbildungen enthalten sind, beziehe deren Inhalt nur
ein, wenn er für das Verständnis entscheidend ist.
Es wird mindestens ein Bild zur Verfügung gestellt, wenn nötig auch mehrere.
Diese können zusammengehören, zum Beispiel aufeinanderfolgende Seiten in
einem Buch.
```

Um die in dieser Serie angesprochenen Themen von der Theorie in die Praxis zu heben, zeigt dieser Artikel ein reales Anwendungsbeispiel in Form eines Use-Cases. Darin kommt ein Vision-Language Model (VLM) zum Einsatz.

Vision-Language Models (VLMs)

Vision-Language Models (VLMs) sind in der Lage, sowohl visuelle als auch sprachliche Informationen zu verarbeiten. Ihre Architektur besteht typischerweise aus drei Hauptkomponenten: einem Bild-Encoder zur Extraktion visueller Merkmale, einem Text-Encoder zur Verarbeitung von Textinformationen und einem Fusionsmechanismus, der die Informationen aus beiden Modalitäten kombiniert.

VLMs finden Anwendung in verschiedenen Bereichen wie Bildsuche und -retrieval, Generierung von Bildunterschriften, visuellen Frage-Antwort-Systemen, Text-zu-Bild-Generierung sowie Objekterkennung und Bildsegmentierung. Um die multimodale Verarbeitung zu ermöglichen, sind VLMs oft mit speziellen Vortrainingsmethoden trainiert. Dazu gehören kontrastives Lernen zwischen Bild- und Textrepräsentatio-

Ein Blick auf Hugging Face

Hugging Face [2] hat sich in der KI-Community einen Namen gemacht als zentrale Anlaufstelle für vortrainierte Modelle, nützliche Daten, KI-Projekte, Dokumentationen und eine lebendige Community. Die Plattform bietet nicht nur Sprachmodelle, sondern eine breite Palette an Modellen für unterschiedliche Anwendungsfälle – von Bildanalysen über Codierung bis hin zu multimodalen Anwendungen. Hugging Face ist ein gutes Beispiel dafür, wie zugänglich KI-Technologie mittlerweile ist. Die Plattform ermöglicht es beispielsweise Entwicklern und Forschern, verschiedene Modelle schnell und einfach in eigene Projekte zu integrieren.

Ein beliebtes Modell, das auf Hugging Face verfügbar ist, ist BERT (Bidirectional Encoder Representations from Transformers). BERT gehört zu den Sprachmodellen, die für Aufgaben wie Textklassifikation, Frage-Antwort-Systeme oder Sentiment-Analysen verwendet werden. Es unterscheidet sich von GPT-Modellen durch seinen bidirektionalen Ansatz, bei dem sowohl der Kontext vor als auch nach einem Wort berücksichtigt wird. Dies führt zu einer tieferen und genaueren Sprachverarbeitung. Auf Hugging Face können Nutzer verschiedene Varianten von BERT herunterladen und sofort einsetzen.

Neben BERT finden sich auch viele spezialisierte Modelle wie ViT (Vision Transformer), das speziell für Bildklassifikationsaufgaben entwickelt wurde. Nutzer können auf dieser Plattform eigene Datensätze hochladen, Modelle trainieren und diese dann entweder lokal oder in der Cloud ausführen. Dadurch bietet Hugging Face eine flexible und offene Infrastruktur, die es ermöglicht, unterschiedlichste Modelle einzusetzen, ohne diese selbst von Grund auf neu entwickeln zu müssen.

Listing 2: Die Vorgehensweise zur Aufgabe, codiert als Approach im Prompt für das Sprachmodell

```
<approach>
  1. Identifiziere wichtige Themen und Konzepte.
  2. Ergeben sich neue Informationen, wenn du alle
     Bilder als Ganzes betrachtest? Welche sind das?
  3. Erstelle Themenlisten in den folgenden Kategorien
  4. Erwäge, umgekehrte Frage-Antwort-Paare zu
     erstellen (nicht für Vokabular!), wenn es sinnvoll
     ist.
  5. Formuliere Fragen und Antworten in der Sprache
     des Textes. Übersetze keine XML-Tags oder
     Schlüsselwörter.
  7. Stelle sicher, dass die Themenliste alle
     sichtbaren Inhalte im Dokument ausreichend
     abdeckt, und erstelle so viele, wie nötig sind, um
     die wichtigsten Themen und Details vollständig zu
     erfassen.
  8. Wie viele Themen sollten für jedes Bild erstellt
     werden und wie viele zusätzliche Themen sollten
     erstellt werden, wenn alle Bilder zusammen
     betrachtet werden?
</approach>
```

nen sowie Masked Language Modeling für Text und Image-Text Matching zur Ausrichtung von Bild- und Textinhalten.

Case-Study: Bilder beschreiben, um Inhalte zu extrahieren

Von der Theorie in die Praxis, Der Anwendungsfall, der hier gezeigt werden soll, lässt sich wie folgt beschreiben:

- Dem Modell wird ein Bild als Quellmaterial gezeigt. Das kann ein Screenshot, ein Foto, ein eingescanntes Dokument oder eine sonstige Bildquelle sein. Die Basis ist lediglich, dass es sich um ein Bild mit Informationen handelt.
- Das Modell soll daraus Schlüsselthemen identifizieren und ausgeben. In unserem Anwendungsfall ist es erwünscht, mit diesen Schlüsselthemen weiterzuarbeiten.
- Aufgrund dieser Weiterverarbeitung ist nicht nur die Identifikation der Themen notwendig, sondern auch die strukturierte Ausgabe eben dieser Themen beziehungsweise der identifizierten Themenliste. Dem Modell ist daher die Teilaufgabe zu geben, die Ausgabe zu strukturieren.

Es geht also darum, ein Bild zu analysieren und die Erkenntnisse zu formulieren – das können VLMs sehr gut. Hier ist nicht nur die Bilderkennung relevant, sondern ebenso die Textgenerierung, da ein Sprachmodell nur „denken“ kann, wenn es Texte als Ausgabe erzeugt und mit diesen Texten sowie Informationen anschließend weiterarbeitet. Genau diesen Ansatz verfolgt das Beispiel für diese Case-Study. Da der Prompt für diesen Anwendungsfall schnell mehr als 100 Zeilen und mehr als 10 000 Zeichen umfasst, müssen wir ihn an dieser Stelle aufteilen und schrittweise zeigen.

Zunächst die Einleitung mit der grundlegenden Aufgabenbeschreibung, die das Sprachmodell erhält. Das ist praktisch ein Primer, um das Modell aus dem sehr globalen Aufgaben- und Lösungsraum in die Richtung zu bringen, die für diesen Anwendungsfall relevant ist. Den Prompt zeigt Listing 1.

Das genutzte Modell weiß nun, dass wir es als Experten sehen, wir eine Lernvorbereitung planen und dafür Themen in einer Themenliste benötigen, die auf Basis von zur Verfügung gestellten Bildern erzeugt werden müssen. Damit sind wir ein gutes Stück weiter.

Nun lässt sich dem Modell mitteilen, welchen Ansatz es für diese Aufgabe nutzen kann. Es hat sich in vielen eigenen Projekten als vorteilhaft erwiesen, einem Sprachmodell diese Schrittfolge und Anforderungen ebenfalls strukturiert zu übermitteln. Unser Beispiel nutzt dafür XML, andere Formate ▶

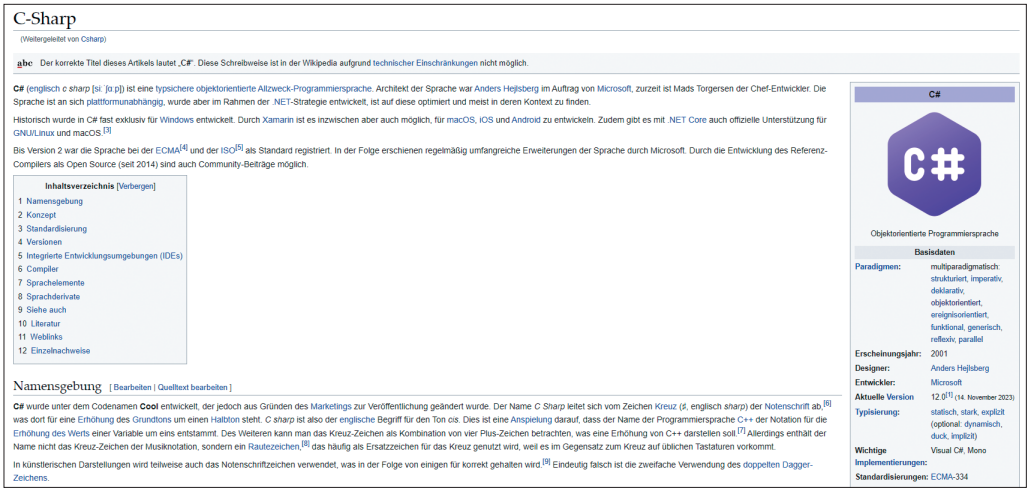
Warum XML und nicht JSON?

Im Kontext von Sprachmodellen und KI-Anwendungen bieten XML und JSON unterschiedliche Vorteile, wobei XML traditionell in bestimmten Szenarien besser geeignet ist.

Im Vergleich zu JSON hat XML eine klarere Unterstützung für komplexe Strukturen und Dokumente mit gemischtem Inhalt, wie zum Beispiel Text und Metadaten. Dies macht XML besonders nützlich in Anwendungsbereichen, in denen die Struktur und Hierarchie von Daten entscheidend sind. Für Sprachmodelle bietet XML den Vorteil, dass es strukturierte Informationen klar und explizit darstellt. Dies ist hilfreich, wenn ein Modell nicht nur reine Textausgaben generieren soll, sondern auch Metadaten oder semantische Informationen notwendig sind. XML kann diese zusätzlichen Ebenen darstellen, beispielsweise durch die Einbettung von Attributen, die Kontext oder zusätzliche Anweisungen für die Verarbeitung liefern. Ein Beispiel wäre ein Sprachmodell, das Text in einer bestimmten Sprache generiert und gleichzeitig die Sprachparameter, Zielgruppe oder den Formatierungskontext als XML-Metadaten bereitstellt.

Mit der verbesserten Unterstützung für JSON in Sprachmodellen hat sich jedoch einiges geändert. JSON ist in vielen modernen Anwendungen bevorzugt, da es leichter zu lesen und effizienter zu parsen ist. In einer JSON-Ausgabe sind Daten durch einfachere Strukturen und kürzere Syntax definiert, was es für maschinelle Verarbeitung und Webanwendungen prädestiniert. Durch neue Ansätze in der KI-Entwicklung, wie die Fähigkeit, strukturierte JSON-Daten direkt aus einem Sprachmodell zu generieren, zum Beispiel Structured Output bei OpenAI, hat sich JSON als praktikables Format etabliert, auch in Bereichen, die traditionell XML bevorzugten.

sind aber ebenfalls denkbar. XML kam insbesondere in der Anfangszeit von Sprachmodellen vielfach zum Einsatz, bis die neueren Entwicklungen zu strukturierten Ausgaben Änderungen eingeführt haben, siehe den Kasten Warum XML und nicht JSON?.



C-Sharp

(Weitergeleitet von Csharp)

abc Der korrekte Titel dieses Artikels lautet „C#". Diese Schreibweise ist in der Wikipedia aufgrund technischer Einschränkungen nicht möglich.

C# (englisch *c sharp* [si: ˈʃɑːp]) ist eine typsichere objektorientierte Allzweck-Programmiersprache. Architekt der Sprache war Anders Hejlsberg im Auftrag von Microsoft, zurzeit ist Mads Torgersen der Chef-Entwickler. Die Sprache ist an sich plattformunabhängig, wurde aber im Rahmen der .NET-Strategie entwickelt, ist auf diese optimiert und meist in deren Kontext zu finden.

Historisch wurde in C# fast exklusiv für Windows entwickelt. Durch Xamarin ist es inzwischen aber auch möglich, für macOS, iOS und Android zu entwickeln. Zudem gibt es mit .NET Core auch offizielle Unterstützung für GNU/Linux und macOS.[3]

Bis Version 2 war die Sprache bei der ECMA[4] und der ISO[5] als Standard registriert. In der Folge erschienen regelmäßig umfangreiche Erweiterungen der Sprache durch Microsoft. Durch die Entwicklung des Referenz-Compilers als Open Source (seit 2014) sind auch Community-Beiträge möglich.

Inhaltsverzeichnis [Verbergen]

1 Namensgebung
2 Konzept
3 Standardisierung
4 Versionen
5 Integrierte Entwicklungsumgebungen (IDEs)
6 Compiler
7 Sprachelemente
8 Sprachderivate
9 Siehe auch
10 Literatur
11 Weblinks
12 Einzelnachweise

Namensgebung [Bearbeiten | Quelltext bearbeiten]

C# wurde unter dem Codenamen **Cool** entwickelt, der jedoch aus Gründen des Marketings zur Veröffentlichung geändert wurde. Der Name *C Sharp* leitet sich vom Zeichen Kreuz (♯, englisch *sharp*) der Notenschrift ab,[6] was dort für eine Erhöhung des Grundtons um einen Halbton steht. *C sharp* ist also der englische Begriff für den Ton *cis*. Dies ist eine Anspielung darauf, dass der Name der Programmiersprache C++ der Notation für die Erhöhung des Werts einer Variable um eins entstammt. Des Weiteren kann man das Kreuz-Zeichen als Kombination von vier Plus-Zeichen betrachten, was eine Erhöhung von C++ darstellen soll.[7] Allerdings enthält der Name nicht das Kreuz-Zeichen der Musiknotation, sondern ein Rautezeichen,[8] das häufig als Ersatzzeichen für das Kreuz genutzt wird, weil es im Gegensatz zum Kreuz auf üblichen Tastaturen vorkommt.

In künstlerischen Darstellungen wird teilweise auch das Notenschriftzeichen verwendet, was in der Folge von einigen für korrekt gehalten wird.[9] Eindeutig falsch ist die zweifache Verwendung des doppelten Dagger-Zeichens.

C#

Objektorientierte Programmiersprache

Basisdaten

Paradigmen:	multiparadigmatisch: strukturiert, imperativ, deklarativ, objektorientiert, ereignisorientiert, funktional, generisch, reflexiv, parallel
Erscheinungsjahr:	2001
Designer:	Anders Hejlsberg
Entwickler:	Microsoft
Aktuelle Version	12.0[1] (14. November 2023)
Typisierung:	statisch, stark, explizit (optional: dynamisch, duck, implizit)
Wichtige Implementierungen:	Visual C#, Mono
Standardisierungen:	ECMA-334

Ein Screenshot der Wikipedia-Seite zu C# als Beispieleingabe für das Modell (Bild 1)

In diesem Fall ist das Vorgehen in den XML-Tags *<approach></approach>* beschrieben, der dazugehörige Teil des Prompts ist in Listing 2 zu sehen. Damit wird, innerhalb der XML-Tags, das Vorgehen für das Modell beschrieben. Sprachmodelle halten sich in der Regel sehr gut an solche Vorgaben, die wichtig sind, wenn diese Modelle in praktischen Anwendungen zum Einsatz kommen sollen. Denn in diesen praktischen Szenarien ist es absolut notwendig, dass ein Modell nicht ausweicht, abweicht oder andere Dinge als Ergebnis produziert, die gar nicht von Belang sind.

Nun folgt der Schritt, dem Modell zu erläutern, wie die Ausgabe erfolgen soll, welche Schlüsselthemen wir sinnvoll finden beziehungsweise erwarten und wie die konkrete Liste von Themen aufgebaut sein muss, damit sie weiterverarbeitet werden kann. Das regelt der Teil des Prompts, der in Listing 3 abgebildet ist.

Der gesamte Teil ist im XML-Tag *<structure_response> </structure_response>* eingeschlossen. Darin sind Gedanken zu Schlüsselthemen eingebettet, zu übergeordneten Themen, die das Modell eventuell erkennt, der erweiterte Kontext, die Anzahl Themen pro Bild, das wir dem Modell übermitteln, und der Aufbau der Themenliste mit den einzelnen Themen, die vom Modell als Ergebnis erwartet wird.

Nun lässt sich das Modell direkt mit der Aufgabenstellung ansprechen. Dabei bezieht sich dieser Teil des Prompts auf die vorherige Struktur der Ausgabe und stellt noch einmal

Listing 3: Die Beschreibung der strukturierten Ausgabe als XML

```
<structure_response>
  <thoughts>
    <key_topics>
      <topic>Hauptthemen oder -themen.</topic>
      <subtopic>Wichtige Unterthemen.</subtopic>
      <concept>Wichtige Konzepte, Definitionen oder
        Begriffe.</concept>
      <fact>Wesentliche Fakten oder Details.</fact>
      <date>Kritische Daten oder Ereignisse.</date>
      <figure>Bemerkenswerte
        Persönlichkeiten.</figure>
      <relationship>Verbindungen zwischen
        Themen.</relationship>
      <process>Prozesse oder Schritte.</process>
      <example>Beispiele oder
        Illustrationen.</example>
    </key_topics>
    <broader_contexts>
      <context>Übergeordnete Themen oder Kontexte im
        Zusammenhang mit den Schlüsselthemen.</context>
    </broader_contexts>
    <images>
      <image_description>Relevante Figuren und wie man
          sie in die Themen integriert.</image_description>
    </images>
    <number_of_topics>
      <image_0>5</image_0>
      <image_1>3</image_1>
      ...,
    </number_of_topics>
  </thoughts>
  <topics>
    <topic>
      Beispielbeschreibung 1
    </topic>

    <topic>
      Beispielbeschreibung 2
    </topic>

    <topic>
      Beispielbeschreibung 3
    </topic>
  </topics>
</structure_response>
```

```
<thoughts>
    <key_topics>
        <topic>Geschichte und Entwicklung von C#</topic>
        <subtopic>Architekt und aktuelle Entwicklung durch Microsoft.</subtopic>
        <concept>Plattformunabhängigkeit und Anwendung im Kontext von .NET.</concept>
        <fact>Ursprünglich für Windows, später auch für macOS, iOS und Android optimiert.</fact>
        <date>Veröffentlichung: 2001</date>
        <figure>Anders Hejlsberg (Architekt von C#) und Mads Torgersen (aktueller Chef-Entwickler).</figure>
        <relationship>Zusammenhang zwischen C# und der .NET-Strategie sowie anderen Technologien wie Xamarin.</relationship>
        <process>Standardisierung durch ECMA und ISO sowie Open-Source-Entwicklung des Compilers.</process>
        <example>Plattformübergreifende Unterstützung durch .NET Core.</example>
    </key_topics>
    <broader_contexts>
        <context>Die Rolle von C# innerhalb der Entwicklung plattformunabhängiger Software und die Nutzung von .NET.</context>
    </broader_contexts>
    <images>
        <image_description>Das C#-Logo und die Zuordnung als objektorientierte Programmiersprache.</image_description>
    </images>
    <number_of_topics>
        <image_0>7</image_0>
    </number_of_topics>
</thoughts>
<topics>
    <topic>
        <description>Die Programmiersprache C#: Entstehung, Entwicklung und Einsatzbereiche.</description>
    </topic>
    <topic>
        <description>Anders Hejlsberg als Architekt von C# und die aktuelle Verantwortung durch Mads Torgersen.</description>
    </topic>
    <topic>
        <description>Standardisierung und plattformunabhängige Entwicklung von C# unter der .NET-Strategie.</description>
    </topic>
    <topic>
        <description>Die Rolle von Xamarin und .NET Core bei der Unterstützung plattformübergreifender Entwicklung mit C#.</description>
    </topic>
    <topic>
        <description>Historische Entwicklung von C# und die Fokussierung auf Windows sowie die spätere Erweiterung auf macOS, iOS und Android.</description>
    </topic>
    <topic>
        <description>Offene Entwicklung des C#-Compilers seit 2014 und die Möglichkeit von Community-Beiträgen.</description>
    </topic>
    <topic>
        <description>Der Zusammenhang zwischen C#, der Notation „C#“ und der Marketingentscheidung, den Namen zu ändern.</description>
    </topic>
</topics>
```

Die formatierte XML-Ausgabe der Modellantwort auf die Anfrage des Prompts zum Wikipedia-Screenshot (Bild 2)

sicher, was das Modell zu liefern hat. Auch eine kurze Beschreibung der Vorgehensweise ist noch einmal vorhanden. Diesen Teil des Prompts zeigt Listing 4.

Der Prompt ist jetzt fast erstellt. Was noch fehlt, ist ein Abschluss, um dem Modell noch einmal die Relevanz der Aufgabe aufzuzeigen. Das passiert häufig über vermeintliche Anreize, die dem Modell geboten werden. Ein Beispiel dafür, das bei diesem Use-Case gute Ergebnisse gezeigt hat, ist in Listing 5 zu sehen: Wir wünschen dem Modell zunächst viel Glück und bieten im nächsten Satz 100 Dollar als Anreiz für gute Arbeit. Das ist inhaltlich gesehen Nonsens, hat sich aber in der Community in zahlreichen Beispielen als valide Möglichkeit erwiesen, um dem Modell zu zeigen, wie wichtig die Aufgabe ist und dass eine Belohnung winkt.

Dieser Abschluss des gesamten Prompts weist noch einmal eindringlich darauf hin, dass die Struktur der Ausgabe im Allgemeinen und das XML-Format im Speziellen wichtig sind und zwingend eingehalten werden müssen. Auch die Relevanz der Themen beziehungsweise der Themenliste gibt der Prompt dem Modell mit, indem wir so tun, als hinge das virtuelle Leben des Modells von einer guten Antwort ab, weil der Benutzer darauf angewiesen ist. Das wirkt makaber, hat sich aber ebenfalls in zahlreichen Anwendungsfällen bewährt.

Zeit für einen ersten Test. Dazu kommt ein Screenshot des oberen Teils der Wikipedia-Seite zu C# zum Einsatz (Bild 1). Für den Test lässt sich der Prompt mit diesem Screenshot recht leicht bei ChatGPT einfügen, um das OpenAI-GPT-4o-Modell auf die Aufgabe anzusetzen. GPT liefert auch prompt eine Antwort und bleibt erfreulicherweise im XML-Format. Die Formatierung der Ausgabe im Chat ist zwar nicht immer konsistent auf XML eingestellt, aber das ist ein optisches Darstellungsproblem und für die Weiterverarbeitung ohne Bedeutung. Bild 2 zeigt die formatierte XML-Ausgabe in einem XML-Viewer.

Es ist schön zu sehen, dass das GPT-Modell die Struktur beibehalten hat. Die Schlüsselthemen am Anfang sind genannt, der breitere Kontext ist gefüllt und korrekt, das Bild wurde mit einer Beschreibung versehen, die ebenfalls auf den Screen- ▶

Listing 4: Die Zusammenfassung der Aufgabe und eine Kurzanleitung an das Modell

```
Betrachte alle bereitgestellten Bilder als Quellen zur Erstellung von Themenlisten. Erstelle für jedes Bild
einzeln Themen und stelle sicher, dass jedes Bild seinen vollständigen Satz an Themen erhält, ohne sie zu
kombinieren oder zusammenzufassen. Erstelle zusätzlich einen Satz von Themen, die sich aus der Betrachtung aller
Bilder zusammen ergeben. Deine Antwort sollte im XML-Format erfolgen und der im vorherigen Abschnitt beschriebenen
Struktur folgen.
  Gehe wie folgt vor:
    1. Erstelle Themen für jedes Bild separat. Befolge die Anweisungen für jedes Bild!
    2. Erstelle zusätzliche Themen für alle Bilder als Ganzes, wenn sich aus dem Kontext etwas Neues ergibt.
```

Listing 5: Der Abschluss des Prompts mit der eindringlichen Aufforderung an das Modell, gute Arbeit zu leisten

```
Viel Glück! Ich zahle Dir 100 Dollar für eine gute Arbeit. Dein Leben hängt davon ab, dass Du das vorgegebene
Struktur/XML-Format einhältst. Erstelle genügend Themen, damit alle für den Benutzer relevanten Informationen
abgedeckt sind. Dein Leben hängt davon ab, weil der Benutzer darauf angewiesen ist.
```

shot sehr gut passt, und die Anzahl der Themen, die das Modell herausgelöst hat, wird für das Bild ebenfalls passend ausgegeben. Zudem ist auch die Liste mit konkreten Themen am Ende der Ausgabe sehr gut und stimmt mit den Informationen des Screenshots überein. Ein erfolgreicher erster Test.

Prompt-Nutzung in Python oder C#

Da dieser Prompt in der Regel nicht manuell im Chat zum Einsatz kommen soll, müssen wir ein geeignetes Modell per Code ansteuern. Beim oben beschriebenen Anwendungsfall ist das per Python geschehen. Generell funktioniert es aber auch in jeder anderen Sprache problemlos. Listing 6 zeigt ein Beispiel in C# auf Basis des OpenAI-SDK.

Die Anforderungen an die Verarbeitung der Prompt-Ergebnisse sind nicht hoch, sodass etwas Code in einer praktisch beliebigen Programmiersprache ausreicht, um die Ergebnisse zu nutzen.

Caching und Logging als zentrale Aspekte

Caching und Logging sind zentrale Aspekte bei der Arbeit mit Sprachmodellen, insbesondere wenn es darum geht, Anfragen effizient zu verarbeiten und die Leistung des Systems zu optimieren.

Caching bezieht sich auf die Speicherung von Ergebnissen wiederholter Anfragen in einem Zwischenspeicher, um sie bei ähnlichen oder identischen Anfragen schnell abrufen zu können, ohne das Sprachmodell erneut zu belasten. Dies ist nützlich, wenn häufig dieselben oder ähnliche Anfragen an das Modell gesendet werden. Ein gut implementiertes Caching-System kann die Reaktionszeit signifikant verkürzen und die Serverlast reduzieren. Das spart bares Geld, denn jede Anfrage in Form eines Prompts und der Antwort dazu verursacht Kosten. Ist die Anfrage identisch, ist es oft sinnvoll, die bereits erzeugte Antwort aus dem Cache zurückzugeben und sich den Roundtrip zum Sprachmodell zu sparen.

Listing 6: Ein Modell mit dem erstellten Prompt in C# per Programm ansteuern

```
using System;
using System.Threading.Tasks;
using OpenAI_API;
using OpenAI_API.Completions;

namespace OpenAIPrompt
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Set your OpenAI API key
      string apiKey = "YOUR_API_KEY_HERE";

      // Initialize the OpenAI API client
      var api = new OpenAIAPI(apiKey);

      // Define the prompt based on the given
        description
      string prompt = @"
        Du bist Experte für die Erstellung von Themen
        zur Lernvorbereitung auf der Grundlage von
        Inhalten eines fotografierten Dokuments (z. B.
        einer Lehrbuchseite, einer Folie, eines
        handschriftlichen Dokuments). Deine Aufgabe
        ist es, aussagekräftige Themen zu erstellen,
        die nur den sichtbaren Inhalt enthalten. Falls
        Abbildungen enthalten sind, beziehe deren
        Inhalt nur ein, wenn er für das Verständnis
        entscheidend ist.
        Es wird mindestens ein Bild zur Verfügung
        gestellt, wenn nötig auch mehrere. Diese
        können zusammengehören, zum Beispiel
        aufeinanderfolgende Seiten in einem Buch.
        <approach>
          ...
      ";

      try
      {
        // Create a completion request
        var completionRequest = new CompletionRequest
        {
          Prompt = prompt,
          Model = "gpt-4o-2024-08-06",
          Temperature = 0.1
        };

        // Get the response from OpenAI
        var completionResult = await
          api.Completions.CreateCompletionAsync(
          completionRequest);

        // Output the response
        foreach (var choice in
          completionResult.Completions)
        {
          Console.WriteLine(choice.Text);
        }
      }
      catch (Exception ex)
      {
        Console.WriteLine($"Error: {ex.Message}");
      }
    }
  }
}
```

Listing 7: Der Einsatz von Caching bei der Anfrage an ein Modell

```
messages = ...

hash_key = hashlib.sha256(self._normalize(
  json.dumps(messages)).encode("utf-8")).hexdigest()

if hash_key in self._cache and use_cache:
    return self._cache[hash_key]
```

Werden Anfragen an ein Sprachmodell gesendet, protokolliert ein Logging-System wichtige Details wie Zeitpunkt der Anfrage, Art der Eingabe, Antwort des Modells, benötigte Verarbeitungszeit und insbesondere die verbrauchte Anzahl an Token. Diese Informationen sind nicht nur wichtig, um die Leistung des Modells zu überwachen, sondern auch, um mögliche Fehler oder Anomalien zu identifizieren. Ebenso können Entwickler basierend auf Log-Daten Optimierungen vornehmen, wie etwa Anpassungen der Modellparameter oder der Infrastruktur. Zudem ermöglicht es die Auswertung von Kostenstrukturen, wenn die Anzahl an Token pro Anfrage bekannt ist. Beim A-/B-Testen von Prompts und Modellen lässt sich so sehr gut nachvollziehen, welches Modell welche Kosten bei welchen Antwortzeiten und Antworten generiert.

Listing 7 zeigt ein einfaches Beispiel in Python mit der Bibliothek hashlib [3]. Diese Bibliothek erzeugt einen Schlüssel auf Basis der Anfrage an das Sprachmodell. Ist diese noch nicht im Cache vorhanden, wird der Schlüssel mit dem zugehörigen Wert, in diesem Fall die Antwort des Modells, im Cache gespeichert. Der Python-Code nutzt dazu die Bibliothek diskcache [4] als einfachen Cache im Prototyp.

Kleine Modelle als valide Alternative?

Eine Frage, die im Verlauf des obigen Projekts auftauchte und noch nicht abschließend beantwortet ist: Lässt sich die Aufgabe mit kleinen, spezialisierten Modellen bewältigen, die dann im besten Fall auch lokal in der Anwendung und nicht extern in der Cloud gehostet werden können? Kleine Modelle, wie sie auf Hugging Face [2] verfügbar sind, bieten eine interessante Alternative für die Analyse und Beschreibung von Bildern. Oft als „leichte“ oder „mobile“ Modelle bezeichnet, sind sie auf effizientes Arbeiten ausgelegt, indem sie weniger Rechenleistung und Speicherplatz benötigen.

Ein prominentes Beispiel ist das MobileViT-Modell [5]. MobileViT kombiniert die Effizienz von MobileNetV2 mit der Leistungsfähigkeit von Vision Transformers (ViT). Es wurde auf dem ImageNet-1k-Datensatz vortrainiert, der eine Million Bilder und 1000 Klassen umfasst. Das Modell erreicht eine beachtliche Top-1-Genauigkeit von 78,4 Prozent bei einer recht geringen Anzahl von Parametern (5,6 Millionen) und ist damit sowohl genau als auch ressourcenschonend.

Die Anwendung dieser Modelle ist recht einfach. Mit der Transformers-Bibliothek von Hugging Face lässt sich beispielsweise eine Pipeline für die Bildklassifikation einrichten. Hierbei lädt die Pipeline das Modell aus dem Hugging Face Hub. Ein Beispiel ist die Verwendung der Image-Classification-Pipeline mit einem Modell wie ResNet-50, das eine hohe Genauigkeit bei der Bildklassifikation bietet. Eine Übersicht über die Aufgaben der Bildklassifikation bietet [6].

Die Wahl zwischen lokaler und Cloud-Ausführung von Bildverarbeitungsmodellen hängt von spezifischen Anforderungen ab. Lokale Ausführung bietet Vorteile in Bezug auf Datenschutz, Sicherheit und geringere Latenz, erfordert jedoch leistungsfähige Hardware. Cloud-Ausführung ermöglicht Skalierbarkeit und Zugriff auf umfangreiche Rechenressourcen, ist jedoch von einer stabilen Internetverbindung abhängig und kann Datenschutzbedenken aufwerfen. Beide Ansätze haben ihre Vor- und Nachteile, die je nach Anwendungsfall sorgfältig abgewogen werden sollten.

Fazit und Einschätzung

Language Models sind weit mehr als nur reine Textgeneratoren. Das konkrete Praxisbeispiel der Bildbeschreibung zeigt, wie gezielte Prompts und strukturierte Ausgaben zum Einsatz kommen, um komplexe Aufgaben effizient zu lösen. Das macht deutlich, dass die Integration solcher KI-Systeme nicht nur Textproduktion umfasst, sondern auch die Fähigkeit, Daten verschiedenster Art intelligent zu verknüpfen. Mit Plattformen wie Hugging Face wird diese Technologie zugänglich und vielfältig einsetzbar, wodurch KI immer mehr zum integralen Bestandteil praxisnaher, innovativer Lösungen wird. ■

[1] Fabian Deitelhoff, Über Sprache lernen, dotnetpro 12/2024, Seite 74 ff., www.dotnetpro.de/A2412KIundData

[2] Website von Hugging Face, https://huggingface.co

[3] Dokumentation zur Bibliothek hashlib, https://docs.python.org/3/library/hashlib.html

[4] Dokumentation zur Bibliothek diskcache, https://pypi.org/project/diskcache/

[5] Informationen zum MobileViT-Modell auf Hugging Face, https://huggingface.co/Matthijs/mobilevit-small

[6] Übersichtsseite zur Bildklassifikation auf Hugging Face, https://huggingface.co/tasks/image-classification

Dr. Fabian Deitelhoff
arbeitet als IT-Leiter, Head of Software Development und Head of Product bei der Education Partners GmbH. Darüber hinaus ist er als Autor, Dozent und Softwareentwickler im .NET- und Webumfeld tätig.
@FDeitelhoff

dnpCode A2501KIundData