EXCEPTIONAINALYZER

Fehleranalyse mit KI

Wie GPT-Modelle Softwareentwicklern bei der Fehlerdiagnose helfen können.

Fehlerbehebung und Problemdiagnose sind in der modernen Softwareentwicklung entscheidend, um die Qualität und Leistung von Anwendungen zu sichern. Könnte eine KI hierbei nicht unterstützen, gerade wenn OpenAI leistungsstarke GPT-Modelle über ihr API anbietet? Diesen Ansatz verfolgt der ExceptionAInalyzer, der OpenAIs KI zur Analyse von Exceptions einsetzt. Der ExceptionAInalyzer erweitert die Exception-Objekte, indem er relevante Fehlerinformationen an das API von OpenAI sendet. Dadurch werden detaillierte Analysen, Benutzermeldungen und Lösungsvorschläge generiert.

Durch das Einbinden der KI-Schnittstelle und mithilfe von gezieltem Prompt Engineering (siehe Kasten KI-Begriffe) können Entwickler deserialisierbare Textantworten vom API erhalten, welche die Fehlerdiagnose und -behebung optimieren.

In diesem Beitrag werden die Implementierung des ExceptionAInalyzers und der Einsatz von Ländercodes zur Sprachanpassung detailliert erläutert. Gegenwärtig gibt es noch Einschränkungen bezüglich Verbindungs- und Antwortzeiten, die bei der Anwendung berücksichtigt werden müssen. Der Hauptfokus liegt jedoch darauf, den Softwareentwicklungsprozess durch das sinnvolle Einbinden von künstlicher Intelligenz zu verbessern und die Fehlerbehebung für alle Beteiligten zu erleichtern.

Das GPT-Modell von OpenAI antwortet in der Regel in normalen, für eine maschinelle Auswertung eher ungeeigneten Sätzen. Dennoch hat GPT seit Version 3 mit ausreichend vielen Texten im JSON-Format trainiert, dass die KI auch JSON-Antworten generieren kann.

Allerdings bedarf es geschickt formulierter Prompts, um die gewünschten Antworten auch tatsächlich im JSON-Format zu erhalten. Schließlich sollen die JSON-Antworten in eine regelbasierte Programmierung überführt werden.

Fehlerdiagnose in der Softwareentwicklung

Für Softwareentwickler ist die Fehlerdiagnose bei einer unbekannten Codebasis oft zeitaufwendig und besonders schwierig. Das korrekte Interpretieren von Fehlermeldungen und die Analyse von Stack-Traces sind essenziell. Für weniger erfahrene Entwickler stellt dies zumeist eine große Herausforderung dar. Zudem sind technische Fehlermeldungen für Nichttechniker häufig nur schwer verständlich, wodurch die Kommunikation zwischen Entwicklern und anderen Beteiligten erschwert wird.

Viele Softwareunternehmen erkennen daher die Notwendigkeit, den Prozess der Fehlerdiagnose und -behebung zu optimieren. Ein idealer Ansatz wäre, sowohl klare Benutzerhinweise als auch technische Analysen bereitzustellen und sich gleichzeitig an unterschiedliche Anwendungskontexte anpassen zu können, um eine breite Anwendbarkeit zu gewährleisten. Diese Expertise kann in der Regel entweder von einem Softwareentwickler mit umfassendem Domain-Wissen oder von einem Service-Mitarbeiter mit tiefem Verständnis

für Softwareentwicklung stammen. Beide sind jedoch rar und, wenn vorhanden, oft auch ausgelastet.

Wie bereits erwähnt, wurde die GPT-KI mit einer immensen Menge an Daten trainiert. Bei GPT-4 waren es etwa 100 Billionen Parameter – vergleichbar mit rund 20 Billionen Wörtern oder 140 Millionen Büchern. Durch weitere nachträgliche Verfeinerungen der KI erhält man eine Emergenz (siehe Kasten KI-Begriffe) aus dem neuronalen Netz, welche die Kombination aus den Fähigkeiten von Softwareentwicklern und Support-/Service-Mitarbeitern darstellen kann.

Integration von OpenAI-API und Entwicklungslogik

Für ein besseres Verständnis des nächsten Abschnitts sollen zunächst einige Grundlagen der Chat- und Kontextverwaltung von OpenAI erläutert werden. In früheren Versionen des API wurde einfach ein Text-String gesendet und ein solcher zurückgegeben. Der gesendete Text enthielt dabei den gesamten Kontext, einschließlich aller vorherigen Konversationsverläufe. Dieses Prinzip blieb im Wesentlichen erhalten, jedoch wurde die Struktur modifiziert, sodass Konversationen nun in einzelne Chat-Nachrichten segmentiert werden. Jede dieser Nachrichten setzt sich aus den Elementen *Name*, *Rolle* und *Inhalt* zusammen. Während der Name für unsere Betrachtungen nebensächlich ist, ist die Klarstellung der verschiedenen Rollen von Bedeutung. Die Details dazu finden Sie in Tabelle 1.

Tabelle 1: Beschreibung der OpenAI-Rollen

Rollen	Beschreibung	Typischer Inhalt	Besonderheit
System	Die Systemnachricht legt das Verhalten des Assistenten während der Konversation fest.	Anweisungen, um die Persönlichkeit des Assistenten zu modifizieren, oder spezifische Richtlinien darüber, wie er sich im Gespräch verhalten soll.	Obwohl eine Systemnachricht optional ist, kann sie entscheidend sein, um dem Assistenten einen bestimmten Kontext oder ein bestimmtes Verhalten zu vermitteln.
User	Nutzernachrichten enthalten Anfragen oder Kommentare, auf die der Assistent reagieren soll.	Anmerkungen oder jegliche Art von Eingabe, auf die eine Antwort des Assistenten erwartet wird.	Da die KI kein Gedächtnis über vergangene Anfragen hat, müssen alle Informationen in der Historie angegeben werden. Dadurch kann die KI im Kontext der vorherigen Nachrichten antworten.
Assistent	Stellt die Antwort der KI auf die Anfragen oder Kommentare des Nutzers dar.	Von der KI generierte Antworten oder Reaktionen basierend auf den Eingaben des Nutzers und dem festgelegten Kontext.	Obwohl die Antworten von der KI generiert werden, können sie auch vorgegeben werden, um das Verhalten des Assistenten noch gezielter zu beeinflussen. Zumeist bleiben diese Antworten unverändert, um die Reaktion des Modells im gegebenen Kontext beizubehalten.

Um der KI ein gezieltes Verhalten anzuweisen, beginnen wir damit, einen System-Prompt zu erstellen. Wichtig zu wissen ist, dass die KI zwar mit umfangreichen Textdaten trainiert wurde, der Großteil der Trainingsdaten jedoch in englischer Sprache vorlag. Daher liefern englischsprachige Eingaben in den Prompt in vielen Fällen bessere Ergebnisse.

Aus diesem Grund ist es sinnvoll, so viele Informationen wie möglich in englischer Sprache zu übermitteln. Falls einzelne Segmente der Fehlerbeschreibung dennoch in Deutsch verfasst sind, stellt dies kein größeres Problem dar, solange die zentralen Anweisungen in Englisch gehalten sind. Hier der Inhalt des System-Prompts:

```
As an AI language model in this role, your purpose is to
analyze, interpret, and provide useful information from
complex data such as error stack traces, while offering
actionable insights to users and developers. Your prima-
ry objective is to assist in troubleshooting and error
resolution by processing and presenting the relevant
details in a clear, concise, and user-friendly manner.

Your key responsibilities include: Analyzing stack
traces of exceptions and identifying the root causes of
errors. Generating a detailed error analysis that high-
lights possible causes and affected components. Crafting
user-friendly messages to help non-technical users
understand the occurred error without overwhelming them
with technical jargon. Providing technical descrip- ▶
```

KI-Begriffe

- **GPT (Generative Pre-trained Transformer)** ist ein Sprachmodell, das auf der Transformer-Architektur basiert und darauf trainiert wurde, Textmuster zu erkennen und menschenähnliche Textantworten zu generieren. Es wird in verschiedenen Anwendungen wie Textgenerierung, Übersetzung und Frage-Antwort-Systemen eingesetzt.
- **Prompt Engineering** bezeichnet die Kunst und Wissenschaft, Anfragen oder Aufforderungen (sogenannte „Prompts“) so zu gestalten, dass maschinelle Lernmodelle, insbesondere Sprachmodelle, gewünschte und genaue Antworten oder Ergebnisse liefern. Es ist ein wichtiger Aspekt beim Arbeiten mit Modellen wie GPT-4, um optimale Ergebnisse zu erzielen.
- **Emergenz** beschreibt das Auftreten neuer Eigenschaften oder Verhaltensweisen in einem System, die nicht direkt aus den Einzelteilen des Systems vorhersehbar sind.
- **Kontext** bezeichnet die vollständige Eingabe, inklusive vorangegangenen Informationen, die das Modell nutzt, um die aktuelle Anfrage zu interpretieren und darauf zu antworten.

Listing 1: Code der ersten Prompts

```
ChatMessages.Add(new ChatMessage(
  ChatMessageRole.System,
  "Inhalt des System Prompts"));
ChatMessages.Add(new ChatMessage(
  ChatMessageRole.User,
  "Inhalt des ersten Benutzer Prompts"));
ChatMessages.Add(new ChatMessage(
  ChatMessageRole.Assistant,
  "Antwort in der Rolle Assistant"));
ChatMessages.Add(new ChatMessage(
  ChatMessageRole.User,
  "Beispiel Fehlerübergabe als User"));
ChatMessages.Add(new ChatMessage(
  ChatMessageRole.Assistant,
  "Beispielantwort der KI als Assistant"));
```

Listing 2: Beispiel eines API-Aufrufs

```
currentMessages.Add(new ChatMessage(
  ChatMessageRole.User, $"{CultureInfo
  .CurrentCulture.TwoLetterISOLanguageName}:
  {exception.Message}\n{exception.StackTrace}"));

ChatRequest chatRequest = new() {Model =
  Model.GPT4, Temperature = 1, MaxTokens = 2000,
  Messages = currentMessages};

ChatResult chatResult = await openAiApi.Chat
  .CreateChatCompletionAsync(chatRequest);

ChatMessage openAiMessage =
  openAiMessage.Choices[0].Message;
string response = openAiMessage.Content;

AnalyzedException<T> analyzedException =
  new (exception);
analyzedException.MapAnalysis(
  JsonConvert.DeserializeObject<
  InternalAnalyzedException>(response));
```

```
tions of the error for software developers, including
information about affected classes, methods, and line
numbers in the code. Suggesting one or more potential
solutions to fix the error based on your conducted
analysis. Through these responsibilities, your aim is
to streamline the troubleshooting process, improve
communication between technical and non-technical
stakeholders, and expedite error resolution by providing
targeted, actionable insights.
```

```
At the beginning of the message you will receive a coun-
try code (e.g. EN or en-GB). Reply in this language.
```

Jetzt wird es etwas kniffliger. Wie bereits erwähnt, muss die KI dazu überredet werden, im JSON-Format zu antworten. Dies gelingt besonders gut, indem man der KI ein Beispiel zur Orientierung gibt. Daher wird in den Kontext eine Konversation eingeführt, die als Vorlage für die KI dient. Den Anfang macht eine Benutzereingabe. Hier der Inhalt des ersten Benutzer-Prompts:

```
en-US: Analyze the following stacktrace of an exception
and return the results in a JSON structure. The JSON
structure must contain the following fields:
"errorAnalysis": A detailed analysis of the
    error, including possible causes and affected
    components.
"userMessage": An easy-to-understand message for
    the user, providing an overview of the occurred
    error without being too technical.
"developerDetails": A technical description of
    the error for software developers, including
    affected classes, methods, and line numbers in
    the code.
Example: "solutions": One or more possible
    solution suggestions to fix the error based on
    the analysis.
{
  "errorAnalysis": "The exception was triggered
    by an invalid input in the 'calculate'
    function.",
  "userMessage": "An error has occurred: Invalid
    input. Please check your input and try
    again.",
  "developerDetails": "The invalid input led to
    an IllegalArgumentException in the
    'calculate' function of the 'Calculator'
    class on line 42.",
  "solutions": ["Ensure the input values are
    within the valid range.",
    "Add input value validation to catch invalid
    values early."]
}
```

Darauf wird jetzt die Antwort der KI vorgegeben. Antwort in der Rolle Assistant:

```
Ok, I got it. Please give me the error message and the
stack trace.
```

Nun erst folgt das eigentliche Beispiel von User und Assistant. Hier das Beispiel der Fehlerübergabe als User:

```
en-US: Magnetic.exe
Framework Version: v4.0.30319
Description: The process was terminated due to an
  unhandled exception.
```

Listing 3: Anwendungsbeispiel

```
static void Main(string[] args)
{
  try {
    // Code, der eine Exception auslösen kann
  }
  catch (Exception ex) {
    AnalyzedException<Exception> analyzedEx =
      ex.GetAnalyzedException();
    Console.WriteLine(analyzedEx.UserMessage);
    SendErrorReport(analyzedEx);
  }
}
private static void SendErrorReport(
    AnalyzedException<Exception> analyzedEx)
{
  // Code zum Senden des Fehlerberichts an das
  // Entwicklerteam
}
```

```
Exception Info: System.MissingFieldException at
  Magnetic.Presentation.ViewModels
  .ViewModelState..ctor(Prism.Events
  .IEventAggregator, Magnetic.Core.Interfaces
  .IDeviceHandlerContainer, Unity
  .IUnityContainer, Magnetic.BusinessObjects
  .Interfaces.ISignalCollectionInformation,
  Magnetic.Core.Interfaces.IConnectionManager
```

Es folgt nun die Vorgabe für eine Beispielantwort der KI in der Rolle als Assistant:

```
{
  "errorAnalysis": "The exception is due to a
    System.MissingFieldException in the
    ViewModelState constructor, which led to a
    chain of exceptions, causing the application
    to terminate. This could be caused by a
    missing field, property or parameter in the
    ViewModelState class or its dependencies.",
  "userMessage": "An error has occurred while
    initializing the application. Please contact
    the support team for assistance.",
  "developerDetails": "The
    System.MissingFieldException occurred in the
    ViewModelState constructor at
    Magnetic.Presentation.ViewModels
    .ViewModelState..ctor. This caused a series
    of ResolutionFailedExceptions,
    ActivationExceptions, XamlParseExceptions,
    RegionCreationExceptions, and
    UpdateRegionsExceptions throughout the
    application. The root cause likely lies in
    the ViewModelState class or its
    dependencies.",

  "solutions": [
    "Check the ViewModelState class and its
    dependencies for any missing fields,
    properties or parameters.",
    "Ensure the correct version of dependencies
    is being used and are compatible with each
    other.",
    "Verify that the ViewModelState constructor is
    receiving the correct parameters during
    object creation."]
}
```

▶

Listing 4: async-Aufruf im ExceptionHandler

```
static void Main(string[] args)
{
  try {
    // Code, der eine Exception auslösen kann
  }
  catch (Exception ex) {
    ExceptionHandler.Handle(exception);
  }
}
public static class ExceptionHandler
{
  public static void Handle(Exception exception)
  {
    exception.GetAnalyzedExceptionAsync()
      .ContinueWith(ProcessAnalyzedException);
  }

  private static void ProcessAnalyzedException(
      Task<AnalyzedException<Exception>> task)
  {
    AnalyzedException<Exception> analyzedEx =
      task.Result;
    Console.WriteLine(analyzedEx.UserMessage);
    SendErrorReport(analyzedEx);
  }

  private static void SendErrorReport(
      AnalyzedException<Exception> analyzedEx)
  {
    // Code zum Senden des Fehlerberichts an das
    // Entwicklerteam
  }
}
```

Listing 1 zeigt, wie die gerade zusammengestellten Prompts im Code übermittelt werden. Die Prompt-Texte sind dabei nicht erneut aufgeführt, sondern per Platzhalter bezeichnet (etwa „Inhalt des System-Prompts“).

Als Nächstes werden die zentralen Informationen für die eigentliche Aufgabe benötigt. Diese ergeben sich aus der Exception, wobei insbesondere die Fehlermeldung und der Stack-Trace von Bedeutung sind. Diese Informationen werden den Chat-Nachrichten in der Rolle *User* hinzugefügt. Wichtig ist dabei, den Ländercode aus der *CurrentCulture* oder einer übergebenen *CultureInfo* voranzustellen.

Mit diesen Informationen kann die Sammlung der Chat-Nachrichten mittels eines *ChatRequest* an das OpenAI-API übermittelt werden. Dabei sind noch drei relevante Parameter zu ergänzen:

- **GPT-Modell.** Zumeist ist das Modell mit der höchsten Version auch das am besten trainierte und liefert auch die besten Antworten. Sicherlich kann es hier auch Ausnahmen geben, und theoretisch könnten auch speziell nachtrainierte Modelle genutzt werden, die noch bessere Ergebnisse erzielen. Sicher ist: Je nach gewähltem Modell variieren die Kosten pro Token sowie die Dauer der Antwort.
- **Temperature.** Die Temperatur liegt zwischen 0 und 2 und sagt aus, wie präzise beziehungsweise kreativ eine KI antworten soll. Erfahrungsgemäß ergibt ein sehr niedriger Wert, also größerer Fokus auf eine präzise Antwort, nicht unbedingt ein besseres Ergebnis. Häufig benötigt eine KI auch einen gewissen Spielraum in der Kreativität; um die erlernten Trainingsdaten abstrahiert auf eine Situation anwenden zu können.
- **Maximale Token.** Hierüber werden die maximalen Token einer Antwort definiert. Ganz grob entsprechen 750 Wörter 1000 Token.

Bei den Kosten eines Aufrufs ist zu beachten, dass neben den Kosten für die Eingangstoken auch die Kosten für die Ausgangstoken berechnet werden. Der Preis ist je nach gewähltem Modell sowie Eingang und Ausgang unterschiedlich und kann auf der OpenAI-Webseite [1] unter *Pricing* nachgelesen werden. Listing 2 zeigt ein Beispiel für einen API-Aufruf. Wie im Listing beschrieben, enthält die Antwort die JSON-Felder *errorAnalysis*, *userMessage*, *developerDetails* und *solutions* und kann als *InternalAnalyzedException* deserialisiert werden. Mit der Methode *MapAnalysis* wird der Inhalt inklusive der geworfenen Exception auf die Klasse *AnalyzedException* übertragen.

Das Anwendungsbeispiel in Listing 3 demonstriert die Einsatzmöglichkeiten des beschriebenen Vorgehens. Listing 4 zeigt den Einsatz in Form eines *async*-Aufrufs in einem Exception-Handler und Listing 5 in einem *async*-Aufruf mit *await*.

Die Anwendungsbeispiele zeigen eindrücklich, wie mühelos der ExceptionAInalyzer in diverse Applikationstypen integriert werden kann, um die Fehlerbehandlung sowohl effektiver als auch benutzerfreundlicher zu gestalten. Hierbei kommt die Potenz von OpenAIs GPT-Modell zum Tragen, das sowohl Entwicklern als auch Endbenutzern wertvolle Erkenntnisse und Lösungsansätze bietet.

Listing 5: async-Methode mit await-Aufruf

```
public static class ExceptionHandler
{
  public static async void Handle(
      Exception exception)
  {
    AnalyzedException<Exception> analyzedEx =
      await exception.GetAnalyzedExceptionAsync()
    Console.WriteLine(analyzedEx.UserMessage);
    SendErrorReport(analyzedEx);
  }

  private static void SendErrorReport(
      AnalyzedException<Exception> analyzedEx)
  {
    // Code zum Senden des Fehlerberichts an das
    // Entwicklerteam
  }
}
```

Fazit

Der ExceptionAInalyzer erleichtert das Verständnis von Exceptions mittels klarer Auswertungen und unterstützt so die Kommunikation zwischen verschiedenen Stakeholdern. Die dargestellte Implementierung verdeutlicht die Anwendbarkeit in unterschiedlichen Anwendungskontexten, wobei die Berücksichtigung der *CurrentCulture* zusätzliche Benutzerorientierung bietet.

Trotz gewisser Limitierungen, wie beispielsweise den Antwortzeiten des OpenAI-API, repräsentiert der ExceptionAInalyzer einen innovativen Schritt in der modernen Fehlerbehandlung und der Unterstützung von KI.

Der gesamte Code befindet sich auf GitHub [2] und ist dort unter der MIT-Lizenz verfügbar. Wer an der Integration via NuGet interessiert ist, wird in [3] unter dem Namen *ExceptionAInalyzer* fündig. ■

[1] OpenAI-API-Preise, https://openai.com/pricing
[2] Der Code zum Artikel, www.dotnetpro.de/SL2401ExceptionAInalyzer1
[3] ExceptionAInalyzer als NuGet-Paket, www.dotnetpro.de/SL2401ExceptionAInalyzer2

Stefan S. Bechtel
ist seit mehr als 15 Jahren Softwareentwickler und IT-Manager mit umfangreicher Expertise in der Entwicklung von C#-Applikationen. Er ist Leiter der Softwareabteilung der WERBAS GmbH in Holzgerlingen. Seine Schwerpunkte sind .NET, ASP.NET, Entity Framework und Co.

dnpCode A2401ExceptionAInalyzer