



universität  
**uulm**

**Fakultät für  
Ingenieurwissenschaften,  
Informatik und  
Psychologie**  
Datenbanken und  
Informationssysteme

# Enhancing BPMNGen with Prompting Strategies for Automated BPMN 2.0 Process Model Generation

Abschlussarbeit an der Universität Ulm

**Vorgelegt von:**

Philipp Letschka  
philipp.letschka@uni-ulm.de  
1050994

**Gutachter:**

Prof. Dr. Manfred Reichert

**Betreuer:**

Luca F. Hörner

2025

Fassung 15. November 2025

© 2025 Philipp Letschka

Satz: PDF- $\text{\LaTeX}$  2 <sub>$\epsilon$</sub>

# Danksagung

Das ist der Text der Danksagung

# **Zusammenfassung**

Das ist der Text der Zusammenfassung

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Problemstellung und Zielsetzung . . . . .	1
1.3	Struktur der Arbeit . . . . .	1
<b>2</b>	<b>Theoretische Grundlagen</b>	<b>2</b>
<b>3</b>	<b>Umstrukturierung und Innovation</b>	<b>3</b>
3.1	Generelle Umstrukturierungen . . . . .	3
3.1.1	Objektorientierter Ansatz . . . . .	3
3.1.2	Von Assistants zu Responses . . . . .	5
3.1.3	Verbesserung der instructions . . . . .	5
3.2	Formatauswahl . . . . .	6
3.3	Dateien . . . . .	8
3.4	Chain of Thought . . . . .	9
3.4.1	Implementierung eines neuen Modus . . . . .	10
3.4.2	Konversationskontext . . . . .	11
3.4.3	Konversationen . . . . .	14
3.5	Weitere Anbieter . . . . .	18
3.5.1	Grok . . . . .	18
3.5.2	Gemini . . . . .	18
3.5.3	Claude . . . . .	18
3.6	Streaming . . . . .	18
3.7	Schema-Constraining . . . . .	18
3.8	Evtl. Sampling . . . . .	18
3.9	Evtl. Reflective Prompting . . . . .	18

<b>4 Performanzanalyse</b>	<b>19</b>
4.1 Qualität . . . . .	19
4.2 Geschwindigkeit . . . . .	19
4.3 Kosten . . . . .	19
<b>5 Verwandte Arbeiten</b>	<b>20</b>
<b>6 Fazit</b>	<b>21</b>
6.1 Zusammenfassung . . . . .	21
6.2 Ausblick . . . . .	21
<b>A Quelltexte</b>	<b>30</b>
<b>Literatur</b>	<b>31</b>

# **1 Einleitung**

## **1.1 Motivation**

## **1.2 Problemstellung und Zielsetzung**

## **1.3 Struktur der Arbeit**

## **2 Theoretische Grundlagen**



## 3 Umstrukturierung und Innovation

Als erstes gilt es herauszufinden welcher Teil des Code, der zuständig für das prompting ist, wie verbessert werden kann. Das Projekt von TBA benutzt zur Erstellung von BPMN Diagrammen die OpenAI API und verwendet hier die bereitgestellte Technologie der Assistants.

### 3.1 Generelle Umstrukturierungen

Während der Code gut für seinen (bisherigen) speziellen Anwendungsfall ist, können hier einige Verbesserungen gemacht werden.

#### 3.1.1 Objektorientierter Ansatz

Das Ziel ist es, den Code einfach erweiterbar und wartbar zu machen. Hierfür ist es wichtig, den code möglichst schnell an Änderungen der OpenAI Api anpassen zu können. Um das zu erreichen wird ein Objektorientierter Ansatz gewählt. Die objektorientierte Programmierung bietet für den Aufbau des Prompting-Codes viele Vorteile und macht die Entwicklung langfristig übersichtlicher und wartbarer. Wir erstellen eine abstrakte Klasse 'ai.ts' welche die gesamte Logik des Prompting beinhaltet und eine Klasse 'openai.ts' welche von der AI Klasse erbt. Die OpenAi Klasse muss nun nur noch Methoden implementieren, welche konkret auf die aktuelle version der API angepasst sind. Durch die Verwendung von abstrakten Methoden wie generateContent(), createTitle() oder processResponse() wird sichergestellt, dass jede konkrete Implementierung dieselbe Schnittstelle einhält, aber ihre eigenen internen Abläufe definieren kann. Dies erleichtert den Austausch und die Erweiterung von Modellen, ohne den restlichen Code verändern zu müssen. Darüber hinaus

werden wiederkehrende Prozesse, etwa das Speichern von Verläufen, das Verarbeiten von Antworten oder die Konvertierung zwischen Formaten, zentral in der Basisklasse gekapselt. Falls sich die API ändert, kann dies nun einfach in der Erbenden Klasse angepasst werden, ohne die dahinterliegenden Logik verändern zu müssen.

So sieht nun in abgespeckter Variante die Klasse für die OpenAI API aus. Es gibt eine Methode `mapPromptInput()`; um den Prompt in das richtige Format der API zu bringen, `generateContent()`; um den eigentlichen API aufruf durchzuführen und `processResponse()`; um die Antwort der API auszulesen.

```
1 export class ChatGPT extends Ai {
2   openai = new OpenAI({
3     apiKey: OPENAI_API_KEY,
4   });
5   assist = await openai.beta.assistants.retrieve("asst_...");
6
7   protected mapPromptInput(input) {
8     return {
9       input: input.prompt,
10      model: this.model,
11    };
12  }
13
14  protected async generateContent(input) {
15    return await openai.beta.threads.runs.createAndPoll(
16      thread_id,
17      { assistant_id: assist.id },
18      { role: "user", content: input }
19    );
20  }
21
22  protected processResponse(response) {
23    return response.output_text.toString();
24  }
25 }
```

#### 3.1.2 Von Assistants zu Responses

Die Änderungen welche im vorherigen Kapitel beschrieben wurden, zeigen gleich ihren Effekt, da OpenAI ankündigt ihre Assistants API einstellen zu wollen. Sie empfehlen einen Umzug zu ihrer neuen Responses API. Dies ist nun recht einfach umzusetzen, da wir nur die elementaren Methoden der API ändern müssen. So sieht nun die neue Methode `generateContent()` ; aus:

```
1 protected async generateContent(input) {  
2     return this.openai.responses.create(input);  
3 }
```

Einer der zentralen Unterschiede zwischen der Assistants- und der Responses-API besteht darin, dass die System-Instructions bei der Verwendung der Responses-API manuell übergeben werden müssen. Dadurch ist es notwendig, die entsprechenden Anweisungen bei jeder Anfrage erneut mitzusenden. Dieser Umstand bringt jedoch nicht nur zusätzlichen Aufwand mit sich, sondern eröffnet auch neue Möglichkeiten: Die Instructions können flexibel und situationsabhängig angepasst werden, wodurch sich das Verhalten des Modells dynamisch steuern lässt. Im folgenden Abschnitt wird gezeigt, wie dieser Ansatz weiter verbessert und effizienter gestaltet werden kann.

#### 3.1.3 Verbesserung der instructions

Da die Instructions nun manuell mit jeder Anfrage übergeben werden, bietet sich die Möglichkeit, deren Aufbau gezielt zu optimieren. Ziel dieser Optimierung ist es, die Anzahl der benötigten Input-Tokens zu reduzieren, ohne dabei Qualität einzubüßen. Im besten Fall wird die Ausgabequalität sogar verbessert. Der Assistant erhält als Grundlage zwei PDF-Dateien, die BPMN-Diagramme im Detail beschreiben, sowie zwei Textdateien: eine mit der Definition des verwendeten JSON-Formats und eine mit allgemeinen Regeln zum Aufbau der Diagramme. Die beiden PDF-Dokumente umfassen zusammen mehr als 10 MB und über 100 Seiten Text. Da ChatGPT bereits ein solides Grundverständnis von BPMN-Diagrammen besitzt, werden diese umfangreichen Dateien aus den Instructions entfernt. Mehrere Tests zeigen, dass

sich diese Reduktion nicht negativ auf die Ergebnisqualität auswirkt. Dadurch lassen sich eine große Zahl an Tokens sowie Rechenzeit und Kosten einsparen.

Die beiden verbliebenen Textdokumente werden anschließend zusammengeführt, überarbeitet und in ein einheitliches, strukturiertes Format gebracht. Alle Regeln sind in einer geordneten Liste zusammengefasst und durch sogenanntes structured prompting klarer und maschinenlesbarer gestaltet. Ergänzend werden den Instructions zwei illustrative Beispiele hinzugefügt: Zum einen ein minimales Beispiel, das den grundsätzlichen Aufbau des JSON-Formats verdeutlicht und die obligatorischen Elemente zeigt. Zum anderen ein umfangreicheres, praxisnahes Beispiel, das ein vollständiges BPMN-Diagramm mit allen relevanten Komponenten abbildet. Diese Kombination sorgt dafür, dass der Assistant sowohl einfache als auch komplexe Diagramme präzise interpretieren und reproduzieren kann.

## 3.2 Formatauswahl

Bisher wird die KI angewiesen, das Diagramm in einem eigens definierten JSON-Format zu erzeugen. Dieses Format wurde jedoch speziell für diesen Anwendungsfall entworfen und existiert in dieser Form nicht offiziell. Entsprechend konnte das Modell während des Trainings kein Vorwissen darüber erwerben, sondern muss das Format ausschließlich auf Grundlage der bereitgestellten Instructions erlernen. Dadurch besteht die Möglichkeit, dass Fehler auftreten, etwa dann, wenn die Anweisungen unvollständig sind oder dem Modell bestimmte Kontextinformationen fehlen.

Um dieses Problem zu vermeiden, wird künftig die Option ergänzt, dass die KI ihre Ausgabe auch direkt im offiziellen Standardformat erzeugen kann. Das weltweit am häufigsten verwendete Austauschformat für BPMN-Diagramme ist XML, zu dem umfangreiche Dokumentation und etablierte Werkzeuge existieren. Dennoch bietet das eigens entwickelte JSON-Format einen entscheidenden Vorteil: Es besitzt eine deutlich höhere Informationsdichte und lässt sich dadurch kompakter und effizienter verarbeiten. Beide Formate haben somit ihre jeweiligen Stärken und Einsatzgebiete.

Kriterium	JSON-Format	XML-Format
<b>Vorteile</b>	hohe Informationsdichte, geringer Tokenverbrauch, schnelle Generierung.	Standardisiert, gut dokumentiert, weit verbreitet, einfachere Instructions
<b>Nachteile</b>	Kein Standard, Lernaufwand, komplizierter Konvertierungsalgorithmus.	hoher Tokenverbrauch, umfangreiche Syntax, unübersichtlicher, mehr Kosten, längere Generierung.

Tabelle 3.1: Vergleich der Vor- und Nachteile der unterstützten Ausgabeformate

Für die Weiterentwicklung ist ein flexibles System das Ziel, das eine dynamische Auswahl des Ausgabeformats besitzt. Dadurch kann der Nutzer selbst entscheiden, welches Format im jeweiligen Anwendungsfall die besseren Ergebnisse liefert. Da sich die Formatwahl ähnlich wie die Wahl des verwendeten Modells direkt auf die Qualität der Ergebnisse auswirkt, wird die Auswahlmöglichkeit direkt in die Modellkonfiguration integriert. So kann beispielsweise zwischen Varianten wie 'gpt-4.1-mini (xml)' und 'gpt-4.1-mini (json)' gewählt werden. Wird kein Format angegeben, erfolgt die Ausgabe standardmäßig im XML-Format.

Eine Anfrage sieht damit z.B. folgendermaßen aus:

```

1 // POST /threads
2 {
3   "inputString": "Bitte generiere mir ein BPMN Diagramm,
4     welches den Ablauf in einem Restaurant zeigt",
5   "model": "gpt-5 (xml)",
6 }

```

Bei einer Anfrage kann dann die jeweilige AI über eine Map

```
const availableGPTs: Map<string, Ai>
```

zugeordnet werden, welche die Anfrage bearbeitet.

```
// TODO: update neuerungen noch machen
```

## 3.3 Dateien

Um die Qualität des Promptings weiter zu verbessern, soll eine Funktionalität implementiert werden, die es ermöglicht, auch Dateien direkt an die KI zu übermitteln. Dabei steht im Vordergrund, dass das Verfahren sowohl im Frontend als auch im Backend möglichst unkompliziert umgesetzt werden kann. Es soll keine aufwendigen oder zeitraubenden Konvertierungen erfordern und eine breite Auswahl an Dateitypen unterstützen, um die Nutzung so flexibel wie möglich zu gestalten. Nach einer Analyse der OpenAI-Dokumentation unter <https://platform.openai.com/docs/guides/images-vision?api-mode=responses&format=base64-encoding-analyze-images> zeigt sich, dass die einfachste und zugleich effizienteste Methode zur Dateiübertragung die Verwendung einer Base64 Data URL ist. Diese Variante bietet eine einfache Möglichkeit, Binärdaten wie Bilder oder Dokumente direkt in Textform zu kodieren und zu übermitteln, ohne zusätzliche Infrastruktur oder spezielle Upload-Mechanismen zu benötigen.

Eine Base64 Data URL ist im eine Textdarstellung einer Datei, die direkt in eine URL eingebettet wird. Dabei werden die ursprünglichen Binärdaten in ein spezielles Textformat namens Base64 umgewandelt. Diese kodierten Daten beginnen typischerweise mit einer Kennzeichnung wie `data:image/png;base64,...` und enthalten danach die eigentlichen kodierten Inhalte. Der große Vorteil liegt darin, dass der Browser oder die API diesen Text automatisch wieder in die ursprüngliche Datei zurückwandeln kann. Auf diese Weise lassen sich Dateien direkt in JSON API-Anfragen integrieren, ohne dass zusätzliche Dateipfade, Server oder externe Speicherorte erforderlich sind. Dieses Verfahren ist daher besonders gut geeignet, um eine einfache, schnelle und universell kompatible Dateiübertragung zu ermöglichen. Dadurch, dass die Datei nun einfach als String übergeben werden kann, können wir die Datei dem Body der Anfrage hinzufügen.

Eine Anfrage sieht damit z.B. folgendermaßen aus:

```
1 // POST /threads
2 {
3   "inputString": "Bitte generiere mir ein BPMN Diagram,
4     welches den Ablauf in einem Restaurant zeigt",
5   "model": "gpt-5 (xml)",
6   "file": "data:image/png;base64,A35ekZ...",
7 }
```

Der string wird auf Validität geprüft

```
1 private checkBase64DataUrl(dataUrl: string): boolean {
2   regex = /^data:([\w.+]+\[/([\w.+]+)?;base64, [\w+\/]+=*$/;
3   return regex.test(dataUrl);
4 }
```

und dann im richtigen Format an die OpenAI API gesendet:

```
1 const imageInstructions = {
2   role: "user",
3   content: [
4     {
5       type: "input_file",
6       file_url: input.getFileDataUrl() as string,
7     } as ResponseInputFile,
8   ],
9 } as ResponseInputItem
```

## 3.4 Chain of Thought

Um die Erzeugung und Interaktion rund um BPMN-Diagramme weiter zu verbessern, wird eine Technik implementiert, die als 'Chain of Thought' bezeichnet wird. Dieses Verfahren ermöglicht es dem Modell, komplexere Denkprozesse intern nachzuvollziehen und schrittweise zu argumentieren, bevor eine Antwort erzeugt wird.

Dadurch kann der Dialog natürlich und strukturiert verlaufen, da der Chatbot in der Lage ist, Zusammenhänge besser zu verstehen und über mehrere Gesprächsschritte hinweg Ergebnisse zu liefern.

Diese Erweiterung erlaubt es dem Nutzer, auf vielfältige Weise mit dem Chatbot zu interagieren: Es können Fragen gestellt, Ideen entwickelt, bestehende Diagramme erläutert oder Verbesserungsvorschläge erfragt werden. Der BPMN-Bot soll damit nicht nur ein Werkzeug für Diagrammerstellung bleiben, sondern sich zu einem vollwertigen, kontextbewussten Assistenten weiterentwickeln, der beim gesamten Modellierungsprozess unterstützt.

Darüber hinaus ist vorgesehen, dass der Chatbot selbstständig Rückfragen stellt, wenn bestimmte Angaben unvollständig, mehrdeutig oder widersprüchlich sind. So kann ein interaktiver Dialog entstehen, in dem beide Seiten aktiv zum Verständnis und zur Qualität der Diagrammerstellung beitragen. Um dies zu ermöglichen, benötigt der Chatbot Zugriff auf den bisherigen Gesprächsverlauf sowie auf den aktuellen Zustand des jeweiligen Diagramms. Nur durch diese Kontextkenntnis kann die KI präzise, nachvollziehbare und qualitativ hochwertige Antworten generieren.

#### 3.4.1 Implementierung eines neuen Modus

Um die Erstellung von Diagrammen für den Nutzer nicht unnötig zu verkomplizieren, bleibt die direkte Generierung eines BPMN-Diagramms weiterhin bestehen. Gleichzeitig soll jedoch mehr Flexibilität bei der Art der Interaktion geboten werden. Zu diesem Zweck wird der Anfrage ein zusätzlicher Parameter hinzugefügt, der das Antwortverhalten der KI steuert.

Über den Parameter `mode` kann festgelegt werden, in welchem Modus die KI reagieren soll. Der bisherige Modus, bei dem ausschließlich das Diagramm erzeugt wird, trägt nun die Bezeichnung `quick`. In diesem Modus erfolgt die Ausgabe direkt und ohne weiteren Dialog.

Der neu eingeführte Modus `detail` aktiviert den sogenannten ‘Chain of Thought’-Ansatz. In diesem Modus verhält sich die KI dialogorientiert. Sie kann Rückfragen stellen, Überlegungen anstellen oder alternative Vorschläge anbieten, bevor das endgültige Diagramm erstellt wird. Dadurch entsteht eine interaktive Konversation,



die vor allem bei komplexeren Prozessen oder unvollständigen Eingaben von Vorteil ist.

Eine Anfrage, die eine solche erweiterte Unterhaltung ermöglicht, könnte beispielsweise folgendermaßen aussehen:

```
1 // POST /threads
2 {
3   "inputString": "Bitte schlag drei Prozessbeschreibungen
4     vor, aus denen dann eine ausgewählt wird um ein Diagramm
5     zu erstellen.",
6   "model": "gpt-5 (xml)",
7   "mode": "detail",
8 }
```

#### 3.4.2 Konversationskontext

Da es nun darum geht ein Chat zu implementieren, bei dem die KI möglichst gut auf Nachrichten reagieren kann, ist es wichtig, dass die KI zugriff auf vorherige Nachrichten sowie auf das Diagramm hat. Wäre das nicht der Fall, könnte die KI keine Fragen über das Diagramm beantworten und auch keine richtige Unterhaltung führen, da immer nur die aktuelle Nachricht bereitgestellt wird. Bei LLM Anbietern wie OpenAI ist es notwendig, dass die Nachrichten historie in der Anfrage mitgeschickt wird.

Hierfür müssen alle Nachrichten eines Theads aus der Datenbank geladen werden. Die Nachrichten werden dann auf wesentliche gefiltert und auf ein kompaktes Forumat gebracht. Die OpenAI Klasse muss dann nur noch die Nachrichten auf das gefoderte Format bringen und in der Anfrage mitschicken. Da die Anfragen an die KI immer komplexer werden, wird nun eine Klasse *PromptInput* angelegt. Diese Klasse beinhaltet alle Informationen welche der KI mitgesendet werden sollen. Bei einer erstellung diese Objekt können alle Rohdaten wie Instructions, Dateien oder der Chatverlauf übergeben werden, welche die Klasse dann automatisch formatiert. Ein entscheidender Schritt hierbei ist es die Teile der Nachrichten zu entfernen, in denen die KI mit einem Diagramm geantwortet hat. Das ist wichtig, da die Diagram-

me viele Tokens beinhalten und eigentlich nur das aktuellste Diagramm wichtig ist. Hier bei kann es aber auch sein, dass das Diagramm noch vom Nutzer bearbeitet wurde. Um dies zu berücksichtigen sind die Diagramme nicht Teil der Nachrichten, welche mitgesendet werden. Die Implementierung der KI-Schnittstelle kann anschließend ein Objekt der Klasse `PromptInput` entgegennehmen. Dieses Objekt dient als zentrale Datenstruktur, über die alle für die Anfrage relevanten Informationen an das Modell übergeben werden. Mithilfe vordefinierter Hilfsmethoden lässt sich der Inhalt komfortabel in das gewünschte Eingabeformat konvertieren, sodass keine manuelle Aufbereitung mehr erforderlich ist.

Die Klasse `PromptInput` verfügt über folgende Attribute:

```
1 export class PromptInput {  
2   instructions: string[];  
3   history: {role: "user" | "assistant", content: string}[];  
4   prompt: string;  
5   file?: ;  
6 }
```

Damit ist es möglich, bestehende Nachrichtenverläufe aus der Datenbank abzurufen und automatisch in das benötigte Format zu überführen. Die Klasse übernimmt hierbei die vollständige Strukturierung der Daten, sodass diese für die Kommunikation mit der KI genutzt werden können:

```
1 const instructions = [formatInstructions, modeInstructions];  
2 const chatsFromDB = getAllChatsFromDB(threadID);  
3 new PromptInput(instructions, prompt, chatsFromDB, file);
```

Das so erzeugte `PromptInput`-Objekt kann anschließend durch interne Methoden in das finale Format umgewandelt werden, das von der KI-Schnittstelle erwartet wird. Dadurch entsteht ein einheitlicher, wiederverwendbarer Datenfluss zwischen Anwendung, Datenbank und Modell, der die Wartung sowie zukünftige Erweiterungen vereinfacht.

```
1 protected mapPromptInput(input: PromptInput) {  
2     [...]  
3     const historyInstructions = input.history.map((item) => {  
4         return {role: item.role, content: item.content};  
5     });  
6     return {  
7         input: [systemInstructions, historyInstructions,  
8             userInstructions, fileInstructions],  
9         model: this.model,  
10    };  
11 }
```

Darüber hinaus soll künftig auch die jeweils aktuellste Version des Diagramms an die Anfrage angehängt werden. Auf diese Weise erhält die KI den vollständigen Kontext und kann das bestehende Diagramm nicht nur bearbeiten, sondern auch inhaltliche Fragen dazu beantworten.

Da das aktuelle Diagramm nicht Bestandteil des eigentlichen Nachrichtenverlaufs ist, wird es in den sogenannten 'System Instructions' hinterlegt. Diesen Instruction Block wird nun als 'Update Instructions' bezeichnet und enthält stets die zuletzt gespeicherte Version des Diagramms. Die entsprechenden Daten werden automatisch aus der Datenbank geladen und vor dem Absenden der Anfrage in die System Instructions eingefügt.

Durch dieses Verfahren ist sichergestellt, dass die KI bei jeder Interaktion auf dem neuesten Stand bleibt und Änderungen im Diagramm jederzeit konsistent nachvollziehen kann.

```
1 protected updateInstructions(threadID: string, format: format){  
2     const diagram = getLatestDiagramFromDB(threadID);  
3     return [`The The following diagram has already been created:  
4         ${format == "xml" ? diagram?.xml : diagram?.json}`]  
5 }
```

Damit hat die KI nun alle Informationen die sie benötigt um eine Konversation führen zu können.

#### 3.4.3 Konversationen

Für eine vollständige Konversation benötigt die KI nun aber noch Anweisungen. Dafür werden der KI noch die sogenannten *modeInstructions* bereitgestellt. Diese sehen zunächst so aus:

```
1 [Output Language depends on prompt language]
2
3 Alternative to creating a bpmn diagram you may ask any
4 questions you feel like need answering to create the best bpmn
5 diagram.
6 If you do not have any questions any more, please write a
7 quick summary of the agreed diagram before creating the bpmn
8 xml.
9 If the initial prompt is really basic provide the user with a
10 creative scenario summary.
11
12 Once the user has agreed on a summary, Do no longer answer in
13 plain text but only in bpmn xml.
14 If you want to provide questions, summary or any other type of
15 text, that is not a diagram. Do not use JSON or XML.
16 Only plain Text!
```

Diese Instructions werden nur mitgesendet, falls der *detail* modus genutzt wird.

Es ist nun eine Konversation möglich.

Wenn man den Bot nun fragt: "Bitte schlag mir ein Diagramm vor" antwortet dieser mit:

Gerne, ich schlage Ihnen ein klassisches und leicht verständliches Szenario vor: einen **Online-Bestellprozess**. Dieses Diagramm würde die Interaktion zwischen einem Kunden und einem Online-Shop darstellen und eignet sich gut, um grundlegende BPMN-Konzepte zu zeigen. **Zusammenfassung des vorgeschlagenen Diagramms:**

- **Beteiligte (Pools):**

1. Kunde
2. Online-Shop

- **Ablauf beim Kunden:**

1. Der Prozess beginnt damit, dass der Kunde ein Produkt auswählt.
2. Er gibt eine Bestellung auf, was eine Nachricht an den Online-Shop sendet.
3. Danach wartet er auf die Bestellbestätigung.
4. Anschließend wartet er auf die Versandbestätigung.
5. Der Prozess endet, wenn der Kunde die Ware erhalten hat.

- **Ablauf im Online-Shop:**

1. Der Prozess startet, sobald die Bestellung des Kunden eingeht.
2. Die Bestellung wird geprüft.
3. Eine Entscheidung (Gateway): Ist der Artikel verfügbar?
  - **Ja:** Die Bestellung wird bestätigt (Nachricht an den Kunden), die Ware wird verpackt und versendet. Eine Versandbestätigung wird an den Kunden gesendet.
  - **Nein:** Die Bestellung wird abgelehnt (Nachricht an den Kunden) und der Prozess endet.
4. Nach dem Versand der Ware ist der Prozess für den Shop abgeschlossen.

Dieses Szenario zeigt die Zusammenarbeit zwischen zwei Teilnehmern, den Austausch von Nachrichten und eine einfache Geschäftsentscheidung. Sind Sie mit diesem Vorschlag einverstanden? Falls ja, erstelle ich das entsprechende BPMN 2.0 XML für Sie.

Man kann darauf antworten mit "Ja, bitte generier mir das Diagramm genau so wie beschrieben". Der Bot versteht, dass er nun nicht antworten, sondern ein Diagramm erstellen soll. Daher sendet er nun kein Klartext sondern direkt ein BPMN XML. Dieses sieht folgendermaßen aus:

### 3 Umstrukturierung und Innovation

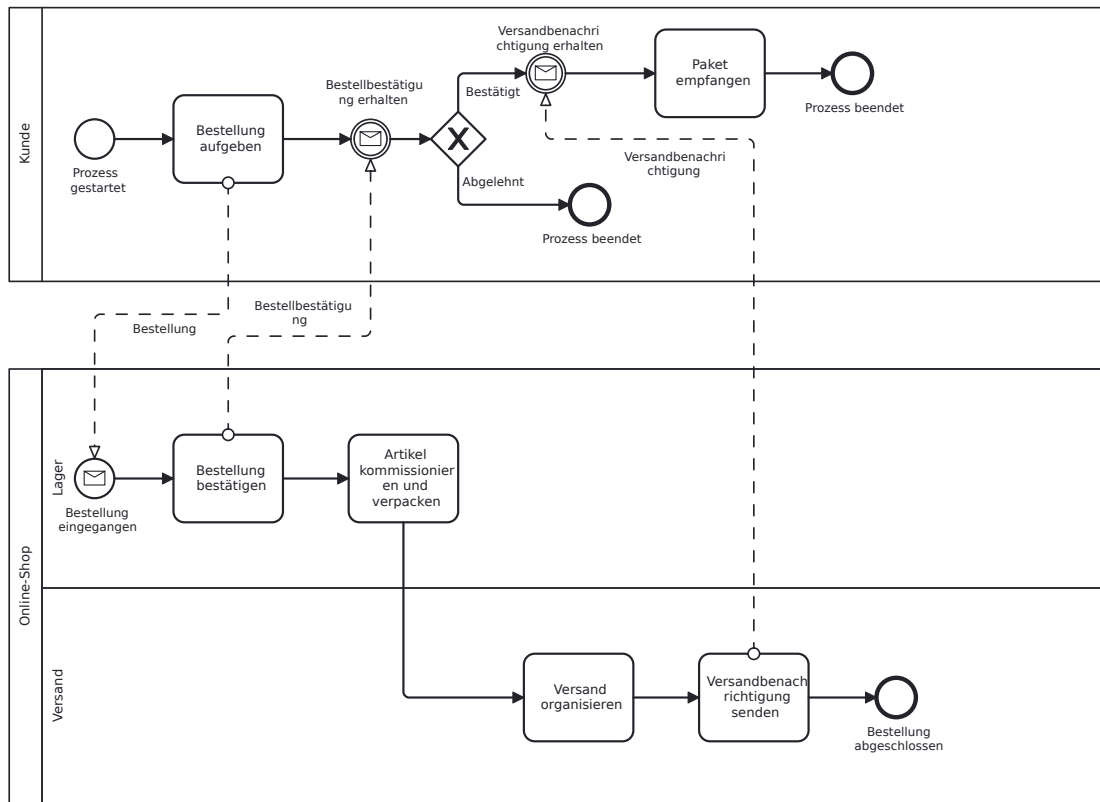


Abbildung 3.1: Beschreibung/Beschriftung des Bilds

Es fällt auf, dass der Bot das Gateway 'Ist der Artikel verfügbar?' vergessen hat. Dies ist allerdings kein Problem da das Diagramm nun weiter durch Prompts verbessern kann. "Bitte bearbeite das Diagramm indem du das vergessene Gateway 'Ist der Artikel verfügbar?' hinzufügst." Der Bot erkennt nun wieder, dass keine Textantwort gewünscht ist und beginnt mit der Übersendung des neuen Diagramms.

### 3 Umstrukturierung und Innovation

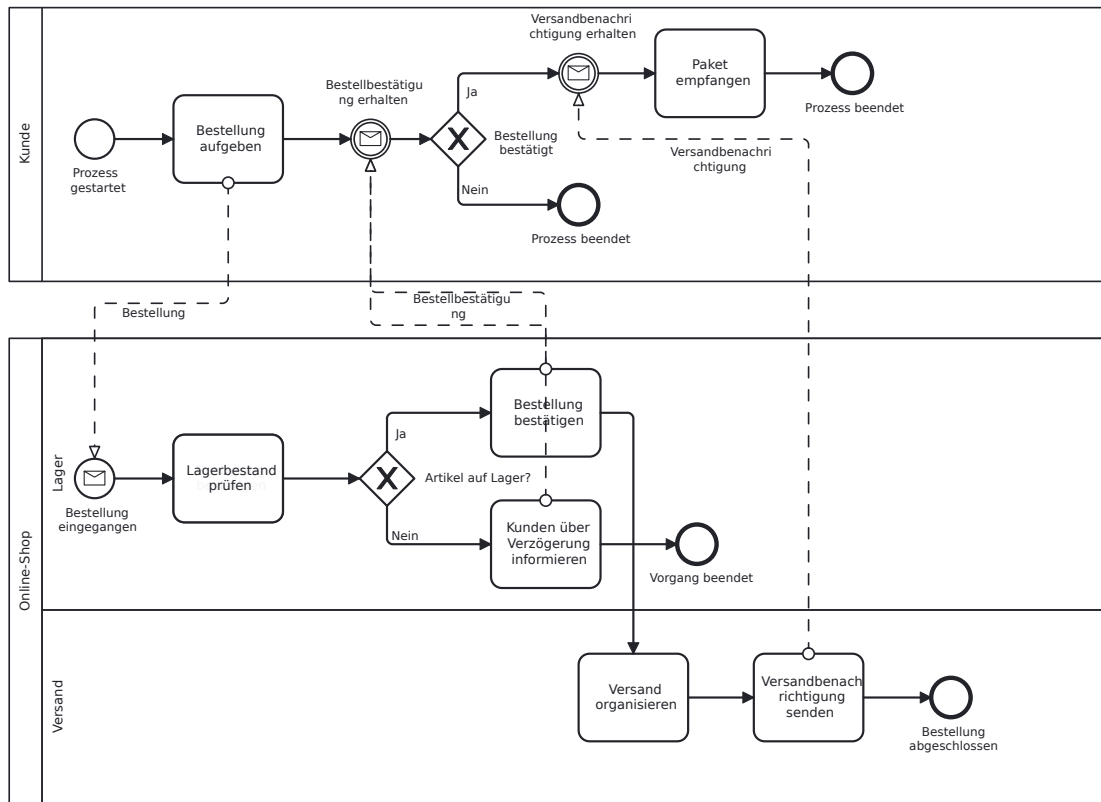


Abbildung 3.2: Beschreibung/Beschriftung des Bilds

## **3.5 Weitere Anbieter**

### **3.5.1 Grok**

### **3.5.2 Gemini**

### **3.5.3 Claude**

## **3.6 Streaming**

## **3.7 Schema-Constraining**

## **3.8 Evtl. Sampling**

## **3.9 Evtl. Reflective Prompting**



## **4 Performanzanalyse**

### **4.1 Qualität**

### **4.2 Geschwindigkeit**

### **4.3 Kosten**

## **5 Verwandte Arbeiten**

Hier sind verwandte Arbeiten

# **6 Fazit**

## **6.1 Zusammenfassung**

## **6.2 Ausblick**

# Notizen

## Gefundene Paper

### **A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT**

- **Autoren:** Jules White et al. (2023)
- **Inhalt:** Katalogisierung von Prompting-Mustern zur systematischen Wiederverwendung.
- **Link:** [arxiv.org/abs/2302.11382](https://arxiv.org/abs/2302.11382)

### **Chain-of-Thought Prompting Elicits Reasoning in Large Language Models**

- **Autoren:** Jason Wei et al. (2022)
- **Inhalt:** Einführung der „Chain-of-Thought“-Technik zur Verbesserung logischer Schlussfolgerungen.
- **Link:** [arxiv.org/abs/2201.11903](https://arxiv.org/abs/2201.11903)

### **The Prompt Report: A Systematic Survey of Prompting Techniques**

- **Autoren:** Schulhoff et al. (2024)
- **Inhalt:** Systematische Übersicht mit 58 Techniken, Kategorisierung und Beispielen.
- **Link:** [arxiv.org/abs/2406.06608](https://arxiv.org/abs/2406.06608)

## **A Systematic Survey of Prompt Engineering in Large Language Models**

- **Autoren:** Sahoo et al. (2024)
- **Inhalt:** Überblick über Methoden, Anwendungen, Modelle und Herausforderungen.
- **Link:** [arxiv.org/abs/2402.07927](https://arxiv.org/abs/2402.07927)

## **Training Language Models to Follow Instructions with Human Feedback (InstructGPT)**

- **Autoren:** Ouyang et al. (OpenAI, 2022)
- **Inhalt:** Einsatz von RLHF zur Verbesserung der Instruktionsbefolgung durch Sprachmodelle.
- **Link:** [arxiv.org/abs/2203.02155](https://arxiv.org/abs/2203.02155)

# Vorlagenbeispiele

Diese kleine Einleitung soll dem Nutzer helfen selbst die eigene Arbeit mit  $\text{\LaTeX}$  zu schreiben. Sie enthält Beispiele zu den wichtigsten Themen .

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

## Dokumentgliederung

Für diese Arbeit verwendet man folgende LaTeX-Kommandos zur Strukturierung:

```
\chapter{Einleitung}
\section{Dokumentgliederung}
\subsection{}
\subsubsection{}
```

Allerdings sollte man sich überlegen, ob man wirklich bis zur Stufe `subsubsection` Überschriften benötigt.

## Illustrationen

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

## Bilder und Abbildungen

Auch in einer wissenschaftlichen Arbeit können Bilder und Abbildungen zur Veranschaulichung und zur Illustration sachlicher Inhalte integriert und eingefügt werden. Für Fotografien und Bilder unterstützt PDF- $\LaTeX$  direkt `jpg` und `png`. Ansonsten empfiehlt es sich, Vektorgrafiken zu verwenden und diese als `pdf` zu speichern. Sollte ein Bild einmal von zu viel weißem Raum umgeben sein, kann man mit dem Werkzeug `pdfcrop` das Bild automatisch zuschneiden.

Mit Hilfe eines Labels `\label{fig:bild1}` kann man sich dann im fortlaufenden Text mittels eines Querverweises auf diese Grafik beziehen: `\ref{fig:bild1}`.

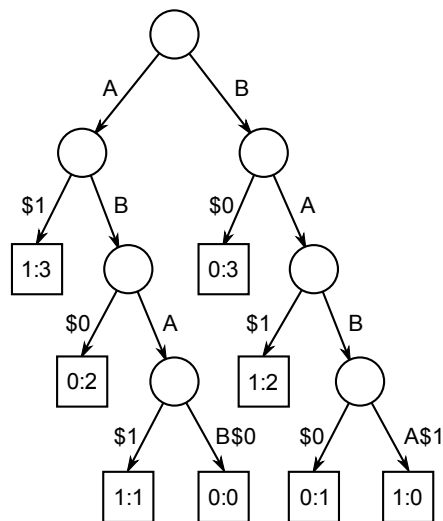


Abbildung 6.1: Beschreibung/Beschriftung des Bilds

An der Stelle des ref-Kommandos platziert LaTeX die Nummer der Abbildung: ‚siehe Abbildung 6.1‘.

## Tabellen

Seite 26, Abschnitt 6.2, enthält Beispieltabelle 6.1. In vielen  $\text{\LaTeX}$ -Büchern finden sich gute Anleitungen zum Erstellen von Tabellen. Komplexere Tabellen können sinnvollerweise in Excel oder einer anderen Tabellenkalulation vorgefertigt und mit einem Umwandlungsprogramm oder -werkzeug in LaTeX-Quellcode konvertiert werden.

A	B	C
x	x	x
x	x	x

Tabelle 6.1: Eine kleine Beispieltabelle



## Formeln

Mathematische Formeln lassen sich in der Umgebung `math` erzeugen. Die Kurz-Schreibweise lautet `\( a^2+b^2=c^2 \)`; hierbei steht die Formel dann im laufenden Text:  $a^2 + b^2 = c^2$ . Die kürzeste Form ist mit zwei `$` um die Formel, z.B. so: Wasser ist H<sub>2</sub>O. `H$_2$O`

Mit der Schreibweise `\[ y=x^2 \]` wird die Formel mittig in einer eigenen Zeile gesetzt, z.B.

$$y = x^2$$

Formeln in der Umgebung `equation` werden mittig in einer eigenen Zeile gesetzt und fortlaufend nummeriert:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (6.1)$$

Wenn wir z.B. über die beliebte Mitternachtsformel (Gleichung 6.1) Details im umliegenden Text schreiben wollen, lässt sich diese wie ein Bild oder eine Tabelle referenzieren, sofern man ihr ein Label zugewiesen hat..

## Programmier-Code

Mehrzeiliger Programmier- und Quellcode kann mit `verbatim` in einer Umgebung gesetzt werden:

```
Dieser Text steht in einer verbatim-Umgebung und wird daher  
in Schreibmaschinenschrift geschrieben.
```

```
LaTeX-Kommandos, z.B. \includegraphics[width=.6\textwidth]{bild.jpg}  
werden nicht interpretiert, sondern "verbatim" ausgegeben.
```

Schöner und professioneller lässt sich Programmier-Code mit dem `listings`-Paket, eingeben, formatieren und ausgeben. Dazu kann man in der Präambel die Sprache angeben, in der die Quellcodes geschrieben sind.

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         System.out.println("Hello World");  
4     }  
5 }
```

Innerhalb einer Zeile gibt man Wörter am Besten als `\verb##` an, dabei erwartet  $\text{\LaTeX}$  zweimal das gleiche Zeichen als Begrenzer. Im Beispiel ist dies die Raute #, man kann aber auch jedes andere Zeichen nehmen, z.B. das Plus +.

## Text

Textteile können bei Bedarf mit dem Befehl `\emph{}` *hervorgehoben* werden. Falls in einem Satz ein Punkt vorkommt, macht man danach kein Leerzeichen sondern eine Tilde (z. ~B. ~so!), denn dann fügt  $\text{\LaTeX}$  den korrekten Abstand ein, z. B. so!

In der Präambel der vorliegenden tex-Datei gibt es den Befehl `hyphenation`, der zur Silbentrennung da ist.  $\text{\LaTeX}$  verfügt zwar über eine eingebaute Silbentrennung, die jedoch bei manchen Wörtern falsch trennt. Damit diese Wörter korrekt getrennt werden, gibt man sie dann mit dem Befehl in der Präambel an<sup>1</sup>.

Fußnoten werden mit dem Befehl `footnote` mitten in den fortlaufenden Text eingefügt.<sup>2</sup>

In wissenschaftlichen Arbeiten muss man des öfteren andere Arbeiten zitieren. Dazu nutzt man die Stiloptionen und Zitierbefehle des Pakets `biblatex`, z. B. `numeric` (=Standard-Stil) oder `verbose` resp. `\cite{name}` oder `\autocite{name}`. In eckigen Klammern kann man noch die Seitenzahl angeben, falls notwendig. Der Name ist ein Schlüssel aus der Datei `bibliography.bib`. Falls einmal ein Werk nur indirekt zu einem Teil der Arbeit beigetragen hat, kann man es auch mit `nocite` angeben, dann landet es in der Literaturliste, ohne dass es im Text ausdrücklich zitiert wird.

---

<sup>1</sup>Das Wort *Silbentrennung* ist hier das Beispiel

<sup>2</sup>Wie man schon im vorherigen Absatz sehen konnte.

## Weiterführendes

Zum Schluss sei auf die Vielzahl an Büchern zu  $\text{\LaTeX}$  verwiesen. In jeder Bibliothek wird sich eine Einführung finden, in der dann weitere Themen wie mathematische Formeln, Aufbau von Briefen und viele nützliche Erweiterungen besprochen werden.

# A Quelltexte

In diesem Anhang sind einige wichtige Quelltexte aufgeführt.

```
1 function greet(name: string): void {  
2     console.log(`Hello, ${name}!`);  
3 }  
4 greet("World");
```

# Literatur

- [1] Jörg Knappen. *Schnell ans Ziel mit LATEX 2e*. 3., überarb. Aufl. München: Oldenbourg, 2009.
- [2] Frank Mittelbach, Michel Goossens und Johannes Braams. *Der Latex-Begleiter*. 2., überarb. und erw. Aufl. ST - Scientific tools. München [u.a.]: Pearson Studium, 2005.
- [3] Joachim Schlosser. *Wissenschaftliche Arbeiten schreiben mit LATEX : Leitfaden für Einsteiger*. 5., überarb. Aufl. Frechen: mitp, 2014.
- [4] Thomas F. Sturm. *LATEX : Einführung in das Textsatzsystem*. 9., unveränd. Aufl. RRZN-Handbuch. Hannover [u.a.]: Regionales Rechenzentrum für Niedersachsen, RRZN, 2012.
- [5] Herbert Voß. *LaTeX Referenz*. 2., überarb. u. erw. Aufl. Berlin: Lehmanns Media, 2010.

Name: Philipp Letschka

Matrikelnummer: 1050994

### **Erklärung**

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Philipp Letschka