



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 6. November 2025

Danksagung

Das ist der Text der Danksagung

Zusammenfassung

Das ist der Text der Zusammenfassung

Inhaltsverzeichnis

1 Einleitung	1
1.1 Motivation	1
1.2 Problemstellung und Zielsetzung	1
1.3 Struktur der Arbeit	1
2 Umstrukturierung und Innovation	2
2.1 Generelle Umstrukturierungen	2
2.1.1 Objektorientierter Ansatz	2
2.1.2 Von Assistants zu Responses	4
2.1.3 Verbesserung der instructions	4
2.2 Formatauswahl	5
2.3 Dateien	7
2.3.1	7
2.4 Chain of Thought	7
2.4.1	7
2.5 Weitere Anbieter	7
2.5.1 Grok	7
2.5.2 Gemini	7
2.5.3 Claude	7
2.6 Streaming	7
2.7 Schema-Constraining	7
2.8 Evtl. Sampling	7
2.9 Evtl. Reflective Prompting	7
3 Performanzanalyse	8
3.1 Qualität	8
3.2 Geschwindigkeit	8

Inhaltsverzeichnis

3.3 Kosten	8
4 Verwandte Arbeiten	9
5 Fazit	10
5.1 Zusammenfassung	10
5.2 Ausblick	10
A Quelltexte	19
Literatur	20

1 Einleitung

1.1 Motivation

1.2 Problemstellung und Zielsetzung

1.3 Struktur der Arbeit

2 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.

2.1 Generelle Umstrukturierungen

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

2.1.1 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 }
```

2.1.2 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.

2.1.3 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.

2.2 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
Vorteile	hohe Informationsdichte, geringer Tokenverbrauch, schnelle Generierung.	Standardisiert, gut dokumentiert, weit verbreitet, einfachere Instructions
Nachteile	Kein Standard, Lernaufwand, komplizierter Konvertierungsalgorithmus.	hoher Tokenverbrauch, umfangreiche Syntax, unübersichtlicher, mehr Kosten, längere Generierung.

Tabelle 2.1: 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.

2.3 Dateien

2.3.1 ...

2.4 Chain of Thought

2.4.1 ...

2.5 Weitere Anbieter

2.5.1 Grok

2.5.2 Gemini

2.5.3 Claude

2.6 Streaming

2.7 Schema-Constraining

2.8 Evtl. Sampling

2.9 Evtl. Reflective Prompting

3 Performanzanalyse

3.1 Qualität

3.2 Geschwindigkeit

3.3 Kosten

4 Verwandte Arbeiten

Hier sind verwandte Arbeiten

5 Fazit

5.1 Zusammenfassung

5.2 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

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

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

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

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

Vorlagenbeispiele

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

 Lorem ipsum dolor sit amet, consectetuer 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, consectetuer 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-Kommados 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-L^AT_EX 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}.

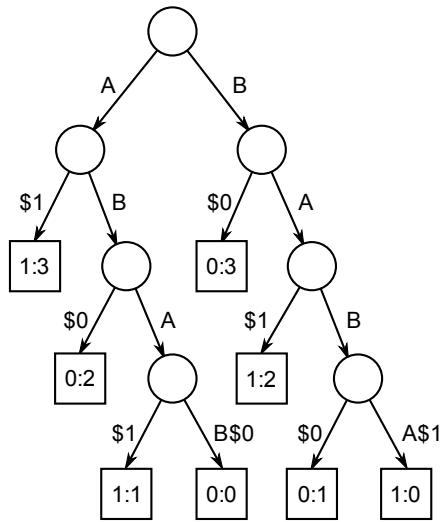


Abbildung 5.1: Beschreibung/Beschriftung des Bilds

An der Stelle des ref-Kommandos platziert LaTeX die Nummer der Abbildung: ,siehe Abbildung 5.1'.

Tabellen

Seite 15, Abschnitt 5.2, enthält Beispieltabelle 5.1. In vielen \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 5.1: 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 H₂O. `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:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (5.1)$$

Wenn wir z.B. über die beliebte Mitternachtsformel (Gleichung 5.1) 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 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 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. 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 an¹.

Fußnoten werden mit dem Befehl footnote mitten in den fortlaufenden Text eingefügt.²

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.

¹Das Wort Silbentrennung ist hier das Beispiel

²Wie man schon im vorherigen Absatz sehen konnte.

Weiterführendes

Zum Schluss sei auf die Vielzahl an Büchern zu \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 : Leitfäden 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 selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den

Philipp Letschka