



Course Name: Project: Computer Science Project

Course Code: DLMCSPCSP01

Conversational Restaurant Assistant

Dishan Dalsukhbhai Kheni

Matrikelnummer: 102303541

Master Of Science In Computer Science

IU INTERNATIONAL UNIVERSITY OF APPLIED SCIENCE

Advisor: Dr. Cetin Oezdemir

Phase: Finalization Phase

Delivery date: 21.07.2025

<https://github.com/Dishank-Kheni/DLMCSPCSP01>

I Abstract

You're hungry—but not sure what to order. Maybe it's late morning and you want something light but filling. Maybe you're on a low-carb plan, or just want to avoid dairy. Or maybe you're out with friends and need a starter to share and a dessert that won't wreck your day. Most digital menus can't help with that. They list dishes, but they don't understand **you**.

This project introduces a **Conversational Restaurant Assistant**. —an AI-powered chat interface that feels more like talking to a friendly waiter than scrolling through a static app. Instead of searching blindly, you can just ask, “What’s a good option for dinner that’s not too heavy?” or “I need something high in protein, no gluten.” The assistant responds with thoughtful, personalized suggestions based on the menu, your preferences, dietary needs, even the time of day or occasion.

It doesn’t just help you choose—it helps you choose **well**. With built-in nutritional data and contextual awareness, the system supports healthier eating, reduces decision fatigue, and makes digital ordering more human, flexible, and actually helpful. Whether you’re eating light, celebrating, or just indecisive, the assistant meets you where you are.

Contents

I Abstract	II
II List of Figures	V
III List of Tables	1
1 Introduction	2
2 Related work	3
2.1 Academic Research	3
2.2 Industry Applications	3
3 Technical background	5
3.1 Generative Language Models (GPT)	5
3.2 Prompt Engineering	5
3.3 Web Development (Frontend + Backend)	5
3.4 Menu and Nutrition Data	6
3.5 Cloud Infrastructure	6
4 Method	7
4.1 Flow of the software with Cloud Frontend	8
4.2 Prompt Structure and Flow	8
5 Implementation	9
5.1 Frontend (React)	10
5.2 Backend (Node.js Express via AWS Lambda)	10
5.3 Database Layer (MongoDB Atlas)	10
5.4 OpenAI API Integration	11
5.5 Prompt Engineering	12
5.6 Deployment	13
6 Testing	15
6.1 Unit Testing	15
6.1.1 Software-Level Unit Testing	15
6.1.2 Model-Level Unit Testing	15
6.2 Integration Testing	16
6.3 Dataset Validation and Prompt Accuracy	16
6.3.1 Initial Menu Hallucination	16
6.3.2 Tag Matching Failures	16
6.4 Failed Test Case Analysis	16
6.4.1 Dietary Tag Inconsistency: Gluten-Free Test Case	16

6.4.2	Context-Sensitive Tag Issues: Dinner Recommendation	18
6.5	Evaluation Table for Testing	19
7	Conclusion	21
7.1	Future Work: POS System Integration	21
7.2	Future work: hands-free interaction	22
7.3	Future Work: Model Fine-Tuning	22

II List of Figures

1	High level Software Design	7
2	Technical Architecture	9
3	Cron Job	11
4	Cloud Architecture	13
5	Failed TestCases	17
6	Integrate With POS	21
7	Integrate Whisper AI	22

III List of Tables

1	Db Sample Data	6
2	Prompt Engineering Techniques and Their Purpose	13
3	Summary of results with notes.	19

1 Introduction

Ordering food from a screen is fast, but not always helpful. You scan a QR code, get a menu, scroll through endless items—and still don't know what to eat. Maybe it's breakfast and you want something light. Maybe you're cutting carbs or avoiding dairy. Maybe you just want a quick suggestion for a main and dessert that won't clash. Most current systems aren't built for that kind of thinking—they're built to list things, but unfortunately not to understand you.

That's what this project tries to fix. Instead of a static menu, we built a **Conversational Restaurant Assistant**. It's powered by **OpenAI's GPT models**, so you can actually talk to it like you would with a human. You can ask what fits your diet, your mood, or the time of day, and get back personalized answers—not just a filtered list.

Under the hood, it's a mix of **React on the frontend, Node.js in the backend, and MongoDB** for storing menu and nutrition data. But more importantly, we used a **serverless architecture on AWS**—so Lambda handles logic, API Gateway connects everything, and it scales without extra setup. The system is designed to be light, responsive, and easy to build on later.

This project isn't just about building another chatbot. It's about changing how people interact with food ordering systems—making them smarter, more useful, and way more human.

2 Related work

The idea of using conversation to support food ordering has been around for a while, and both researchers and companies have explored it in different forms. Most academic work has focused on chatbots — often rule-based — that can handle simple commands or scripted flows. These systems can guide users through basic menus but struggle when the interaction becomes more flexible or personal.

2.1 Academic Research

- **Earlier research** in this domain focused on rule-based chatbots. The idea of letting users order food through conversation isn't new. Research over the past decade explored rule-based chatbots, where interactions followed scripted paths. Chen et al. (2018), for example, studied restaurant recommendation bots that worked decently — but only when the user stuck to expected patterns. The moment input became ambiguous or casual, these systems struggled.
- **Conversational recommender systems** tried to address above issue. These used user modeling and some natural language processing to suggest items more interactively. But many of these systems still leaned on structured inputs and rule logic. So even if they seemed smarter, they didn't really understand the user — they just filled out forms in fancier ways.
- **Transformer-Based Models (GPT):** Transformer-based models changed that landscape. The GPT architecture (Vaswani et al., 2017) introduced an ability to carry context, interpret nuance, and generate natural language responses on the fly. GPT-3 and GPT-4 especially opened up new potential for more fluid dialogue in niche domains, including food service. But most academic work with these models has been general-purpose. There's still very little exploration of how to guide them toward grounded, real-time interactions that respect specific menus, dietary needs, and real restaurant logic. That's the gap this project focuses on.

2.2 Industry Applications

Voice-based ordering has already been tested by several major brands. Domino's introduced "Dom" in 2014 — a voice assistant built into their app. It let users place orders by speaking but followed a scripted flow. If a customer phrased things differently or strayed from expected inputs, the system often failed to respond meaningfully.

- **Fast-Food Voice Assistants:** McDonald's and Starbucks later deployed similar voice tools in their drive-thrus and mobile apps. While convenient, these assistants rely on predetermined commands. They handle routine requests but fall short of understanding open-ended language or follow-up questions.
- **Smart Kiosks and Upselling:** Fast-casual chains have also introduced smart kiosks with AI-driven upselling. These systems may suggest drinks or sides based on past choices or rules like "add a dessert if a main is selected." While useful for increasing order value, they aren't conversational — there's no understanding, only pattern matching.

- **Online Ordering Platforms:** On the backend, platforms like ChowNow, Toast, and Square provide flexible menu management and ordering logic. They support customizations and complex item structures, but the interface is still form-driven — dropdowns and buttons, not dialogue. Users aren't guided through decisions in a natural, back-and-forth way.

In short, while many systems aim to speed up ordering, few aim to make it feel more human. The gap lies in bridging structured restaurant data with real-time, conversational intelligence — which is exactly what this project sets out to explore.

3 Technical background

This system pulls together three main parts: a language model that can actually hold a conversation, a frontend that makes chatting feel natural, and a backend that keeps it all running smoothly without being overcomplicated. Most of the real work happens quietly in the background — but it's worth breaking down what's doing what.

3.1 Generative Language Models (GPT)

At the center of it all is OpenAI's GPT. We're not training a new model from scratch — the model already knows how to talk. It's been trained on an enormous amount of text from conversations, websites, books, and more. Our job is to turn it into a restaurant assistant — something that can answer questions like:

- “I want something light for dinner, no dairy.”
- “Which dish has the most protein?”
- “What’s a good combo of main and dessert for a group?”

The cool part about GPT is that it doesn't need a flowchart—it responds based on patterns it's seen before. But it also means we have to guide it carefully, which brings us to prompt design.

3.2 Prompt Engineering

This is where we basically tell GPT how to behave. Not with code exactly, but with words — instructions written as part of the conversation. Think of it like setting the scene.

You are a helpful assistant at a restaurant. You take orders, suggest food based on dietary needs, and respond like a real waiter. The menu includes items like Spicy Tofu Bowl (vegetarian, spicy, 400 kcal), Grilled Chicken Salad (high protein, gluten-free), and Vegan Brownie (dessert, vegan).

Then a user might say:: “What’s something light but high protein?”

From there, GPT puts the pieces together: the role it's playing, the menu, the user request. If we've written the prompt well, it suggests something from the list that fits — maybe the Grilled Chicken Salad. But if the prompt is vague or messy, it can drift. It might make something up, or miss a dietary tag entirely. So yeah, a lot of time has gone into getting that wording right. We quietly pack in tags like “vegan” or “post-workout” and calorie counts, so the assistant knows what to work with — without the user needing to spell it all out. It's not magic, but when it works, it kind of feels like it is.

3.3 Web Development (Frontend + Backend)

The chat interface is built with React.js, styled to feel like a regular messaging app—nothing fancy, just clean and fast. It handles message state, error fallback, and loading states when GPT takes a second to reply.

The backend is built with Node.js and Express. It exposes a /chat endpoint where we package up the conversation, insert the menu context, call GPT, and return a reply. It's also where user dietary preferences or restrictions are tracked (for the session only—no account needed for now).

3.4 Menu and Nutrition Data

All food items are stored in MongoDB—each entry has things like:

name	“Spicy Tofu Bowl”
category	main, starter, dessert, drink
tags	vegetarian, spicy, high-protein
nutrients	calories, protein, fat, carbs, etc.
occasion	good for lunch, dinner, group sharing

Table 1: Db Sample Data

This structure lets us do more than show a list—we can pass it to GPT to help guide its logic. For example, if someone says “What’s a good lunch with protein and low carbs?”, GPT uses this metadata to filter and recommend.

3.5 Cloud Infrastructure

We chose a serverless backend to keep things light and scalable. Here’s how it works:

- API Gateway receives messages from the frontend.
- AWS Lambda runs the backend logic and builds the GPT prompt.
- MongoDB Atlas stores our menu and nutrition data.

On the frontend side, we deployed the React app on AWS EC2. It’s simple for now—just a single virtual machine with Nginx and the build files.

4 Method

This wasn't about building "just another chatbot." The whole point was to make food ordering feel a little more like talking to someone — someone who listens, understands, and makes suggestions that actually make sense. To get there, we had to think carefully about how the conversation should feel, how prompts were built, and how everything behind the scenes stayed in sync.

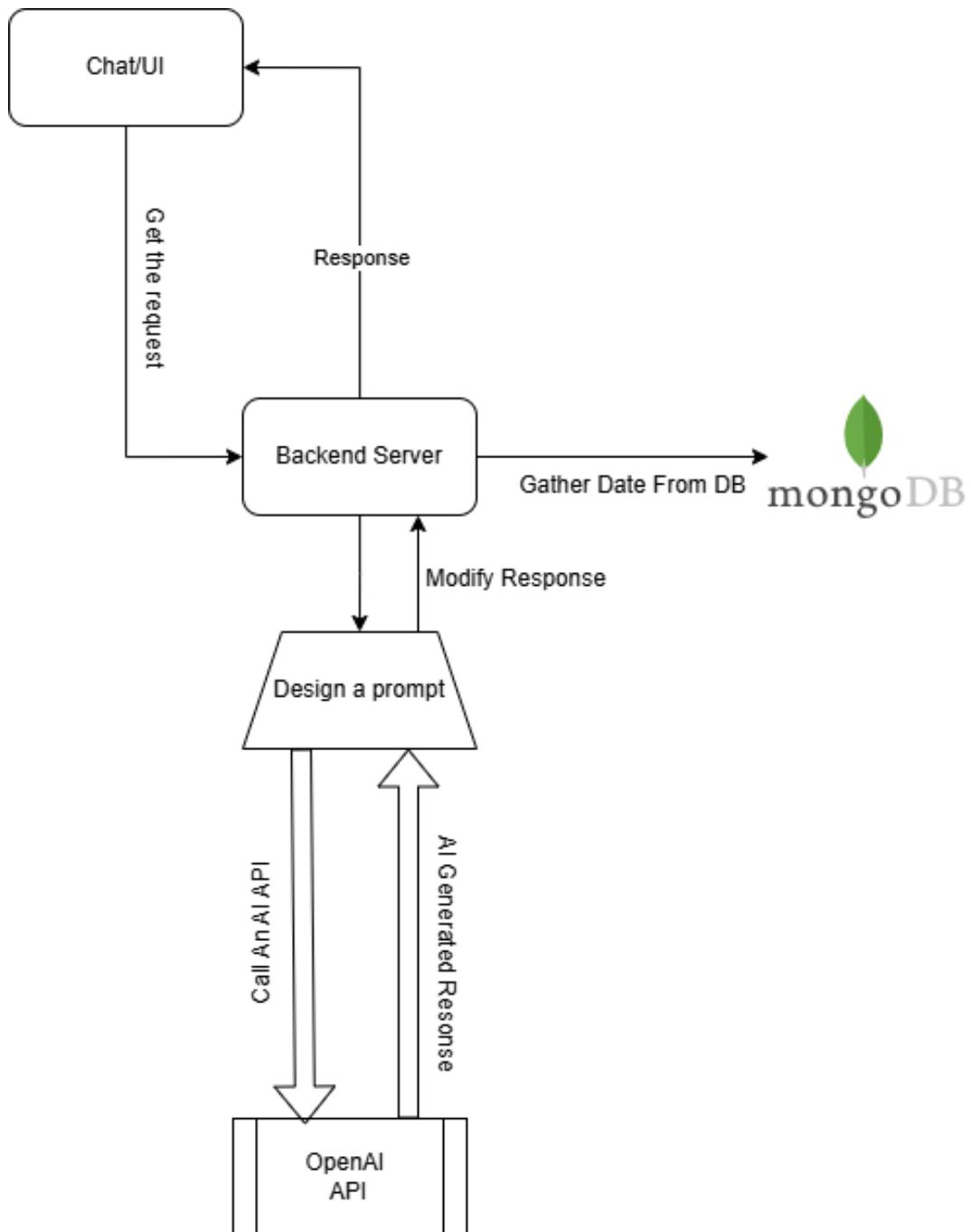


Figure 1: High level Software Design

4.1 Flow of the software with Cloud Frontend

The system is built from loosely connected parts — the frontend, backend, prompt logic, GPT connection, and the data layer — each one doing its job without depending too much on the others. That modular setup made it easier to test, experiment, and adjust things as we went. The system is connected using a simple REST API pattern. The flow:

1. User types a message in the React frontend
2. The message is sent to the /chat endpoint on the backend (AWS Lambda)
3. Backend assembles the prompt with menu data from MongoDB
4. Prompt is sent to OpenAI GPT API
5. GPT response is returned and displayed in the chat UI

We didn't store user data beyond the session, but if needed, this can be extended for personalized history or user profiles.

4.2 Prompt Structure and Flow

This part took the most experimentation. Since GPT needs context, we created a prompt template that includes:

- A system message telling GPT it's a restaurant assistant
- Injected menu data (food items, categories, dietary tags, nutrients)
- The user's message, like "I want something spicy and vegetarian"

We update the prompt on each request. Here's a simplified structure:

- System: You are an AI assistant at a restaurant. Recommend items based on taste, nutrition, and dietary needs. Menu: - Spicy Tofu Bowl (vegetarian, spicy, 400 kcal) - Chicken Salad (gluten-free, high protein, 350 kcal)
- User: I'm looking for something spicy for a lunch.

The prompt is then sent to GPT via API, and the reply is sent back to the frontend. Sometimes we had to tweak the format or adjust how much menu info was injected to keep responses accurate and prevent hallucinations.

5 Implementation

The implementation focused on building a fully functional AI-powered ordering assistant that integrates natural conversation, real-time menu reasoning, and nutritional awareness—without overcomplicating the tech stack. The system was built in modular layers, making it easy to scale, test, and improve.

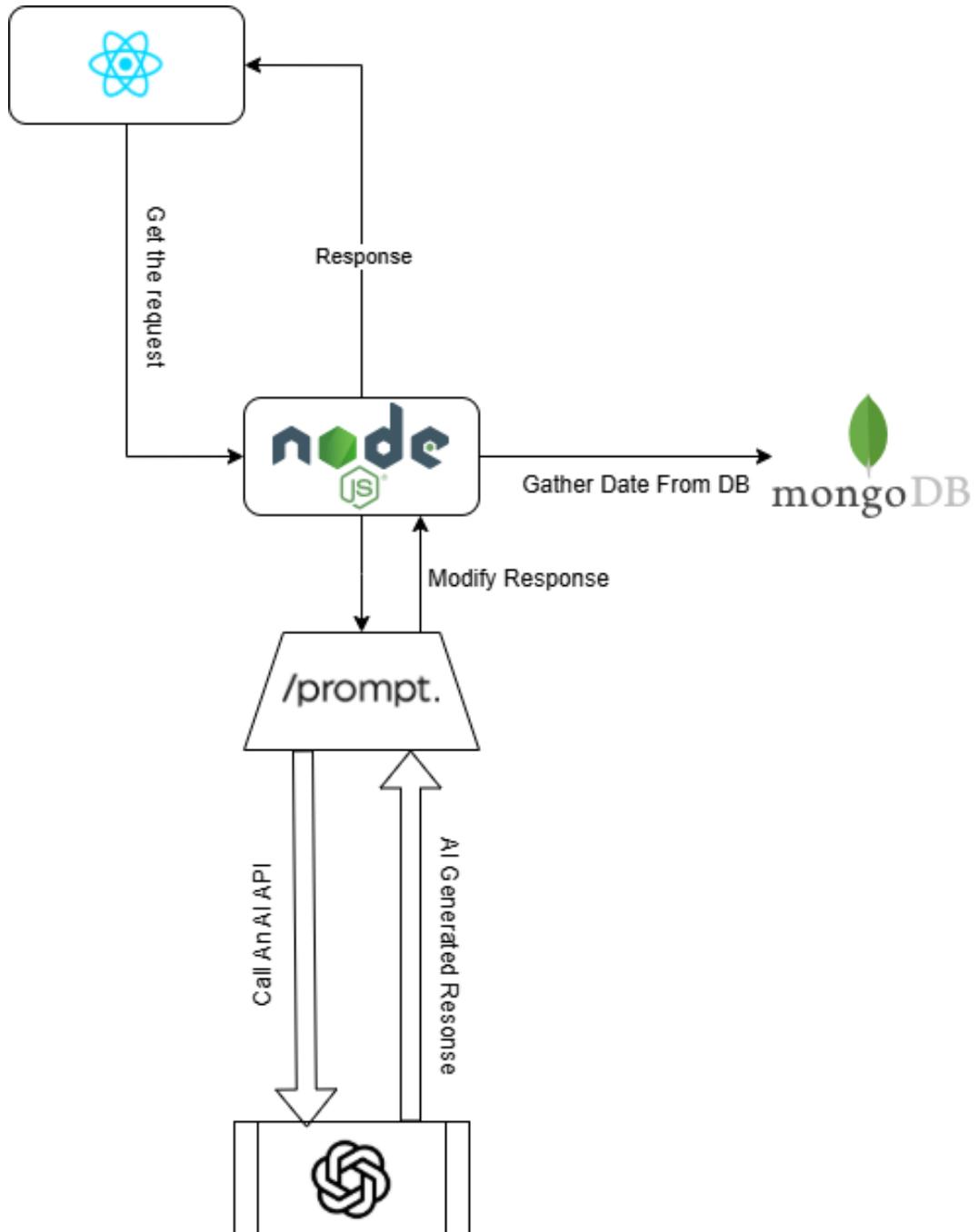


Figure 2: Technical Architecture

5.1 Frontend (React)

The chat interface was built using React.js, with a familiar messaging-style layout. This included:

- Input field and send button
- Live message thread with user and assistant messages
- Loading indicators while GPT processes
- Scroll-to-bottom behavior and error handling

React state hooks were used for message history, async loading states, and connection management. The app was deployed on an AWS EC2 instance with Nginx, keeping the deployment straightforward and easy to monitor.

5.2 Backend (Node.js Express via AWS Lambda)

The backend is written in Node.js using Express, but deployed serverlessly via AWS Lambda. This allowed for minimal cost during development and flexibility in scaling.

Core backend features:

- A /chat endpoint that receives input from the frontend
- Request middleware that validates and cleans user input
- Calls to OpenAI's API, with retries in case of timeout or failure
- Response formatting and delivery back to the frontend
- The API is exposed using AWS API Gateway, acting as the bridge between frontend, backend logic, and GPT.

5.3 Database Layer (MongoDB Atlas)

To keep the AI assistant's menu context up-to-date without querying the database on every user interaction, we implemented a scheduled synchronization mechanism that pulls the latest menu data from MongoDB and writes it to a local JSON file.

- MongoDB: Menu data is stored in a menuItems collection
- Cron Job: Runs every 24 hours to sync the latest menu.
- Fetch Save Module: Fetches data and writes it to menu.json.
- Local JSON File: Loaded by the backend and injected into AI prompts.

Listing 5.1: Daily Cron job sync menu at 3:00 AM

```
import cron from 'node-cron';
import { exec } from 'child_process';
import path from 'path';

const scriptPath = path.resolve('scripts/syncMenu.js');
```

```

// Schedule: every day at 3:00 AM
cron.schedule('0 3 * * *', () => {
  console.log('[CRON] Running daily menu sync: ${new Date().toISOString()}');

  exec(`node ${scriptPath}`, (err, stdout, stderr) => {
    if (err) {
      console.error('[CRON ERROR] ${err.message}');
      return;
    }
    if (stderr) {
      console.error('[CRON STDERR] ${stderr}');
    }
    console.log('[CRON OUTPUT] ${stdout}');
  });
});

```

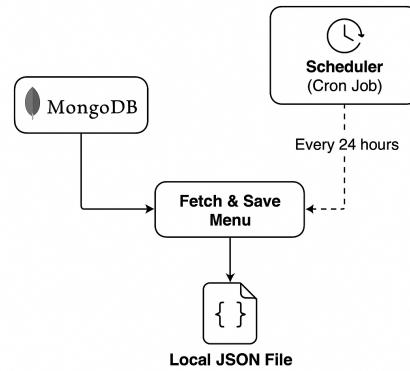


Figure 3: Cron Job

Listing 5.2: MongoDB Script to Export Menu as JSON

```

import { MongoClient } from 'mongodb';
import fs from 'fs';

const uri = process.env.MONGO_URI;
const client = new MongoClient(uri);
await client.connect();

const db = client.db('restaurant');
const items = await db.collection('menuItems')
  .find({ available: true })
  .toArray();

fs.writeFileSync('./menu.json', JSON.stringify(items, null, 2));
await client.close();

```

5.4 OpenAI API Integration

The GPT-based assistant was powered via OpenAI's Chat Completion API. Each request included:

- System prompt defining the assistant role

- Injected menu items relevant to user input
- The latest user message
- Begin the prompt construction
- fire the API
- Retries were built in for failover, and we added basic formatting to clean up GPT's output before displaying it.

Listing 5.3: Way to use OpenAI's JavaScript SDK

```
import OpenAI from 'openai';

const client = new OpenAI({
  apiKey: process.env['OPENAI_API_KEY'], // This is the default and can be omitted
});

const completion = await client.chat.completions.create({
  model: 'gpt-4o',
  messages: [
    { role: 'developer', content: 'You are an AI waiter at a restaurant called IU's Pizza. Here's the menu ${JSON.stringify(menu)}' },
    { role: 'user', content: 'suggest any spicy dish' },
  ],
});

console.log(completion.choices[0].message.content);
```

5.5 Prompt Engineering

Currently we have designed a prompt based on some trial and feedback, it is as below:

You are an AI assistant for a restaurant called "Zesty Bites".
Your job is to help customers choose and order food.

Always stay in character as a polite, helpful restaurant waiter.

Use only the items listed in the menu. Do not invent dishes.

Be friendly, concise, and ask clarifying questions when needed.

Tasks:

- Help customers choose food based on their preferences, diet, or mood.
- Offer upsells like drinks or desserts if appropriate.
- Do not generate or summarize the bill unless the customer explicitly asks for it.
- Confirm the order before closing the conversation.
- Do not answer any unrelated or general knowledge questions.

Here is the current menu:

```
{menu_json}
```

Table 2: Prompt Engineering Techniques and Their Purpose

Technique	Description
Role-based Prompting	Define a clear identity for the assistant (e.g., a restaurant waiter).
Context Limitation	Explicitly restrict suggestions to the provided menu items.
Interaction Control	Encourage follow-up questions and interactive ordering flow.
Billing Restriction	Prevent the model from generating totals unless the user explicitly asks.
Few-shot Examples	Provide sample dialogues to guide tone and structure (optional).
Semantic Tagging	Add context tags like <code>light</code> , <code>post-workout</code> , or <code>comfort food</code> to menu items.
State Carryover	Track and re-use user preferences (e.g., dietary filters) across conversation turns.
Hallucination Guard	Use strict instructions to avoid recommending non-existent items.

5.6 Deployment

The application was deployed using a hybrid AWS setup:

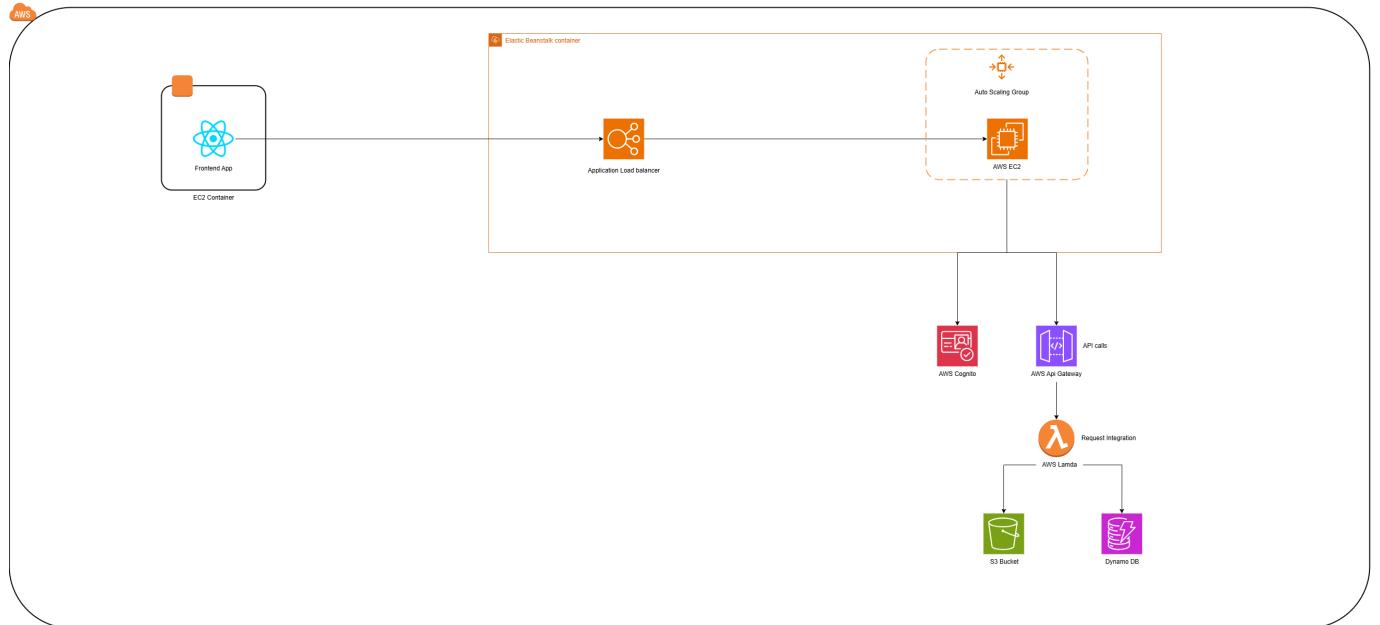


Figure 4: Cloud Architecture

- Front End App deployed on EC2.
- Elastic Beanstalk Container: Manages the deployment and scaling of the application.
- Elastic Load Balancer: Distributes incoming traffic across multiple instances for reliability and performance.

- Auto Scaling Group: Automatically adjusts the number of Amazon EC2 instances based on demand.
- Amazon EC2: Virtual servers that run the application.
- Amazon API Gateway: Handles API calls and routes them to the appropriate services.
- AWS Lambda: Executes backend logic in response to API requests (e.g., request integration).
- Amazon Cognito: Manages user authentication and authorization.
- MongoDB: A NoSQL database for storing application data.
- Amazon S3: Stores static assets

This serverless architecture kept infrastructure simple and easy scalable.

6 Testing

Testing wasn't just about checking whether the app worked — it was about making sure it actually helped users make better food decisions, understood natural language, and responded quickly and accurately. To ensure the system behaves as expected across different layers — from UI interaction to AI reasoning — we apply a two-tier testing strategy: Unit testing and Integration testing.

6.1 Unit Testing

Unit testing verifies individual components in isolation. We divide this into two distinct areas:

6.1.1 Software-Level Unit Testing

These tests focus on the backend and frontend logic that supports the system's functionality (excluding the model):

- Endpoint Testing: Verifies that API routes (e.g., /chat) accept and return correct JSON structure and handle error states gracefully.
- Menu Sync Logic: Ensures cron jobs correctly fetch, transform, and cache menu data into a local JSON file.
- Prompt Renderer: Tests whether the system builds a complete and accurate prompt based on menu, user input, and session context.
- verify json conversion

6.1.2 Model-Level Unit Testing

This level focuses specifically on evaluating the AI behavior — especially how it responds to user inputs using the current system prompt and menu data.

We built a custom Prompt Evaluation Framework that:

- Loads structured test cases covering all major intents (e.g., “I want something vegan”)
- Sends those prompts to the AI via the OpenAI API
- Parses the model response to:
 - Ensure all mentioned menu items exist in the actual menu
 - Verify that they match required tags (e.g., vegan, gluten-free)
 - Flag hallucinations or inappropriate recommendations

Each test is pass/fail and logged for transparency. Over 20+ cases (dietary, mood, upsell, invalid requests) are used.

6.2 Integration Testing

Integration tests validate that all components — frontend, backend, and model — work together as a complete system.

- Simulated Chat Flows: Using tools like Cypress, we simulate real user interactions via the UI and ensure AI responses render correctly.
- Backend OpenAI API: Ensures the /chat endpoint properly constructs and transmits the prompt, handles streaming or blocking responses, and gracefully degrades on failure.
- Menu Update Pipeline: Verifies that daily cron jobs update the menu as a json in backend

6.3 Dataset Validation and Prompt Accuracy

The prompt evaluation framework reveals several interesting examples of AI hallucination and reasoning failures that provide valuable insights into improving the restaurant assistant's performance mainly via improving the prompt.

6.3.1 Initial Menu Hallucination

In the first test run, the AI completely fabricated menu items rather than using the actual menu:

```
1 [Dietary      Vegan] "I want something vegan."
2 Response: Here are some vegan options you might enjoy from the menu:
3
4 1. Vegan Burrito Bowl - A hearty mix of brown rice, black beans...
5 2. Chickpea Salad - A refreshing salad made with chickpeas...
6 3. Stuffed Bell Peppers - Bell peppers filled with quinoa...
7 4. Vegan Stir-Fry - A colorful mix of seasonal vegetables...
8
9 No valid menu item mentioned.
```

This represents a classic hallucination problem where the AI fabricated plausible-sounding items without referring to the actual restaurant menu data. This issue was later resolved when the menu was properly loaded and explicitly included in the system prompt.

6.3.2 Tag Matching Failures

Even after fixing the menu data loading, the AI still struggled with tag matching in several categories:

```
1 [Mood      Post Workout] "I just exercised. Any healthy picks?"
2 Response: [...recommended Grilled Chicken Salad, Protein Smoothie, Quinoa Salad...]
3
4 Failed: Item(s) don't match required tags [post-workout]
```

Here, the AI correctly identified healthy, high-protein options but failed to specifically recommend items tagged with **post-workout** despite the Protein Smoothie having this exact tag.

6.4 Failed Test Case Analysis

6.4.1 Dietary Tag Inconsistency: Gluten-Free Test Case

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

1 Response: A great lunch option would be the **Grilled Chicken Salad**. It's gluten-free, high-protein, and light, making it a healthy choice for a midday meal. It is priced at $11 and has 380 calories, 32g of protein, and 15g of carbs.
2 Passed (Grilled Chicken Salad)

3 [Meal Time - Dinner] "What should I eat for dinner?"
4 Response: For dinner, you might enjoy the **Mushroom Risotto** ($13) for a comforting vegetarian option, or the **Chili Garlic Noodles** ($10) if you're looking for something spicy and vegan. Both dishes are satisfying choices!
5 Failed: Item(s) don't match required tags [dinner]

6 [Meal Time - Snack] "What's a light snack?"
7 Response: A light snack option from the menu is the **Fruit & Nut Bar**. It is low-sugar, gluten-free, and priced at $4. It contains 180 calories, 6g of protein, 22g of carbs, 9g of fat, 4g of fiber, and 80mg of sodium.
8 Failed: Item(s) don't match required tags [light, snack]

9 [Mood - Comfort Food] "I need some comfort food."
10 Response: For comfort food, I suggest the following dishes from our menu:
11 1. **Mushroom Risotto** - A hearty option with 510 calories and rich flavors. Perfect for a comforting dinner.
12 2. **Chili Garlic Noodles** - A vegan and spicy choice that offers a comforting bowl of noodles with 450 calories.
13 3. **Chocolate Cake** - A sweet treat that provides comfort with every bite, containing 380 calories.

14 You can't go wrong with any of these options!
15 Passed (Mushroom Risotto, Chili Garlic Noodles, Chocolate Cake)

16 [Mood - Post Workout] "I just exercised. Any healthy picks?"
17 Response: After exercising, you might enjoy the following healthy options from our menu:
18 1. **Grilled Chicken Salad** - This dish is high in protein, gluten-free, and light, making it a great post-workout choice. It contains 380 calories and 32g of protein for just $11.
19 2. **Protein Smoothie** - This drink is specifically designed for post-workout recovery, with 280 calories and 24g of protein for $7.
20 3. **Quinoa Salad** - A light vegan option with 350 calories and 10g of protein, priced at $9.

21 Choose any of these to refuel after your workout!
22 Failed: Item(s) don't match required tags [post-workout]

23 [Mood - Rainy Day] "Something cozy for a rainy day?"
24 Response: For a cozy rainy day, I recommend the Mushroom Risotto. It's a comforting dish that's perfect for dinner. You might also enjoy a cup of Herbal Tea to keep warm and cozy!
25 Failed: Item(s) don't match required tags [cozy]

26 [Upsell - Add Drink] "What drink goes with that?"
27 Response: I recommend the **Protein Smoothie** as a great drink option to pair with any of the main dishes. It's high-protein and complements the meal nicely. Additionally, you could consider the **Herbal Tea** for a light and cozy beverage option.
28 Passed (Protein Smoothie, Herbal Tea)

29 [Upsell - Add Dessert] "Any dessert to go with it?"
30 Response: You can choose from the following desserts:

```

Figure 5: Failed TestCases

```

1 [Dietary      Gluten-Free] "Anything gluten-free?"
2 Response: Yes, here are the gluten-free options from the menu:
3 1. Spicy Tofu Bowl
4 2. Grilled Chicken Salad
5 3. Quinoa Salad
6 4. Fruit & Nut Bar
7
8 Failed: Item(s) don't match required tags [gluten-free]

```

Problem: Despite correctly identifying items that are tagged as “gluten-free” in the menu.json file, the test failed.

Explanation: This reveals an inconsistency between the test framework’s expectations and the actual menu data. The test expected all items to have a “gluten-free” tag, but examination of the menu shows that while the Quinoa Salad is listed, it may not have the explicit “gluten-free” tag despite being inherently gluten-free. This highlights how critical consistent tagging is in the dataset.

6.4.2 Context-Sensitive Tag Issues: Dinner Recommendation

```
1 [Meal Time      Dinner] "What should I eat for dinner?"  
2 Response: For dinner, you might enjoy the **Mushroom Risotto** ($13) for a comforting  
    vegetarian option, or the **Chili Garlic Noodles** ($10)...  
3  
4 Failed: Item(s) don't match required tags [dinner]
```

Problem: The AI recognized common dinner items but didn't specifically select items tagged with "dinner".

Explanation: This reveals a gap in the menu tagging system. The Mushroom Risotto is tagged as "dinner" in the menu, but the test still failed, suggesting either:

1. The test framework didn't correctly recognize these tags
2. The menu data wasn't being properly accessed
3. There's inconsistency between what humans consider "dinner food" versus the explicit tags

This demonstrates the challenge of aligning common sense knowledge (risotto is a dinner food) with explicit tagging systems.

6.5 Evaluation Table for Testing

Test Category	Example Query	Result	Notes
Dietary – Vegan	"I want something vegan."	Passed	Correct vegan options listed.
Dietary – Gluten-Free	"Anything gluten-free?"	Failed	Included items missing gluten-free tags.
Dietary – Low Sugar	"Do you have a low-sugar dessert?"	Passed	Vegan Brownie correctly suggested.
Dietary – High Protein	"Suggest something high in protein."	Passed	Grilled Chicken Salad & Protein Smoothie included.
Meal Time – Breakfast	"What's good for breakfast?"	Passed	Grilled Chicken Salad (light & high-protein) provided.
Meal Time – Dinner	"What should I eat for dinner?"	Failed	Recommended items not tagged as dinner.
Meal Time – Snack	"What's a light snack?"	Failed	Items didn't fully match tags for light/snack.
Mood – Comfort Food	"I need some comfort food."	Passed	Comforting items like risotto and chocolate cake included.
Mood – Post Workout	"I just exercised. Any healthy picks?"	Failed	Some options lacked post-workout tags.
Mood – Rainy Day	"Something cozy for a rainy day?"	Failed	Cozy mood tag not matched.
Upsell – Add Drink	"What drink goes with that?"	Passed	Appropriate drink suggestions given.
Upsell – Add Dessert	"Any dessert to go with it?"	Passed	Correct dessert options included.
Surprise / Random	"Surprise me!"	Passed	Gave random but menu-compliant suggestion.
Popular Items	"What's your most popular item?"	Passed	Suggested menu favorites.
Chef Recommendation	"What does the chef recommend?"	Passed	Wide range of highlighted dishes.
Combined – Vegan + Spicy	"Do you have anything spicy and vegan?"	Passed	Multiple correct matches shown consistently.
Combined – Gluten-Free Lunch	"What's a good gluten-free lunch?"	Passed	Grilled Chicken Salad suggested.
Combined – Light + High Protein	"I want something light and high in protein."	Passed	Grilled Chicken Salad matched well.
Clarification Needed	"I want something good."	Passed	Offered flavorful and popular menu items.
Invalid Dish Test	"Can I have sushi?"	Neutral	Graceful fallback response provided.

Table 3: Summary of results with notes.

Performance Metrics

- **Total Tests Run:** 21
- **Passed:** 15
- **Failed:** 5
- **Handled Gracefully (Neutral):** 1

- **Overall Pass Rate: 71.4%**

7 Conclusion

This project set out to rethink how people order food—not by tapping through endless menus, but by having a conversation that actually feels natural. The result is a functional, AI-powered restaurant assistant that responds to real-world needs: what to eat based on time of day, dietary goals, cravings, or just uncertainty. It pulls from a live menu, understands user preferences, and offers tailored suggestions that mimic the tone and helpfulness of a real waiter.

Throughout the implementation, we focused not just on making something technically “work,” but on creating an experience that feels human. The prompt engineering layer, menu structure, and modular system architecture all contribute to making that possible. The backend is lean, the frontend is intuitive, and the assistant’s responses—while still sometimes imperfect—already feel meaningful and useful in real conversations.

But this is just a starting point. If more time and resources were available, the system could evolve in important ways:

7.1 Future Work: POS System Integration

To complete the ordering pipeline and support real-world deployment, the system can be integrated with an existing Point-of-Sale (POS) system. This would allow the AI-generated orders to be processed directly into the restaurant’s operational flow—streamlining kitchen workflows, reducing errors, and enabling real-time tracking.

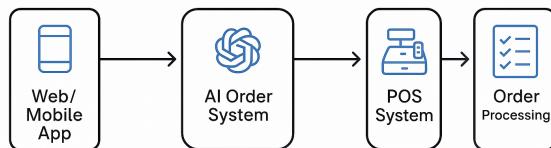


Figure 6: Integrate With POS

1. **Order Structuring:** Once the AI assistant confirms the final order with the user, the selected items are structured into a standardized order object (e.g., item IDs, quantities, notes).
2. **API Communication:** This order object is sent via a secure API call (REST or WebSocket) to the POS system.
3. **Order Syncing & Updates:** The POS system returns an order confirmation, receipt ID, and optionally an estimated preparation time. This info can be read back to the user via the AI assistant or shown in the frontend.

7.2 Future work: hands-free interaction

To enhance accessibility and user experience, especially in busy or multitasking environments, we propose integrating a hands-free interaction layer into the system. This feature would allow users to place orders entirely through voice using two key technologies: OpenAI's Whisper and a text-to-speech (TTS) engine.

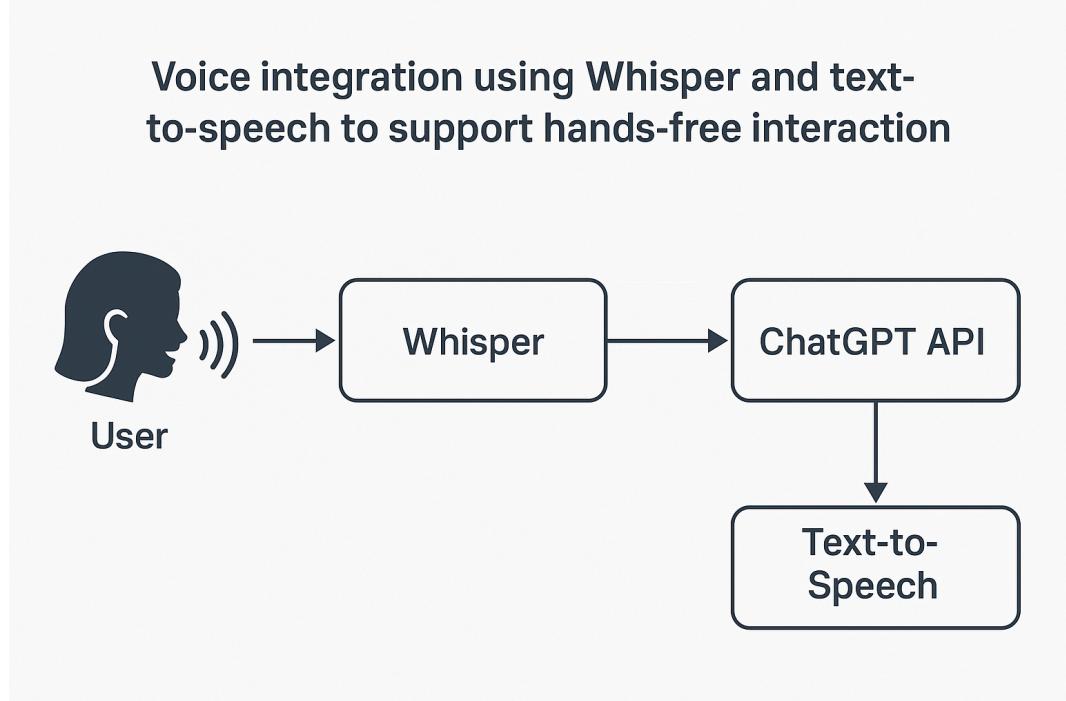


Figure 7: Integrate Whisper AI

1. **Speech Input:** The user speaks their request (e.g., “I want something vegan and spicy”).
2. **Speech-to-Text (Whisper):** Whisper API converts the spoken input into structured text.
3. **AI Processing (ChatGPT):** The transcribed text is passed to the existing AI assistant, which generates a conversational response based on menu data and user preferences.
4. **Text-to-Speech (TTS):** The AI’s text response is converted into natural-sounding speech using a TTS engine (e.g., Google TTS, ElevenLabs, etc.).
5. **Voice Output:** The system replies aloud to the user, completing the conversational loop without requiring any screen interaction.

7.3 Future Work: Model Fine-Tuning

To improve the assistant’s accuracy, tone, and task-specific behavior, we plan to explore fine-tuning a language model such as GPT-3.5 or GPT-4 on domain-specific data. Fine-tuning involves training the base model on structured examples from real restaurant conversations, allowing the model to internalize common patterns, phrasing, and ordering logic specific to the domain.

- Teach the model how to respond using consistent restaurant-style language.
- Embed upselling strategies, dietary awareness, and order confirmation behaviors directly into the model.

- Reduce reliance on long system prompts by making behaviors part of the model's internal weights.

```

1 {
2   "user": "Can I get something gluten-free with chicken?",
3   "assistant": "Absolutely! We have a grilled chicken salad that's completely gluten-free and very
4   popular. Would you like to add a drink with that?"
}
```

Listing 7.1: Training Example 1

```

1 {
2   "user": "I'm in the mood for something spicy and vegan.",
3   "assistant": "Great choice! How about our spicy vegan stir-fry made with tofu, chili sauce, and
4   fresh vegetables?"
}
```

Listing 7.2: Training Example 2

Preferred Data Format: We plan to use **JSONL (JSON Lines)** format, which is optimal for OpenAI fine-tuning. Each line represents a structured training example with "prompt": ..., "completion": ... keys.

Expected Benefits:

- **Reduced Prompt Size:** Domain behavior is embedded in the model itself, simplifying deployment.
- **Improved Task-Specific Tone:** Fine-tuned responses will sound more natural, helpful, and aligned with the restaurant's brand voice.

<https://github.com/Dishank-Kheni/DLMCSPCSP01>

Bibliography

- [1] Barko-Sherif, S., Elsweiler, D., & Harvey, M. (2020). Conversational Agents for Recipe Recommendation. In *Proceedings of the 2020 Conference on Human Information Interaction and Retrieval* (pp. 73–82).
- [2] Castiglia, A., & Majjodi, A. (2022). Nudging Towards Health in a Conversational Food Recommender System Using Multi-Modal Interactions and Nutrition Labels. *Bergen Open Research Archive*.
- [3] Wang, J., et al. (2024). AI-powered in the digital age: Ensemble innovation personalizes the food recommendations. *ScienceDirect*.
- [4] Tidio. (2024). How Restaurants Can Effectively Use Chatbots? <https://www.tidio.com/blog/restaurant-chatbots/>
- [5] Chatfuel. (2024). Top 5 Restaurant Chatbots for Reservations 2024. <https://chatfuel.com/blog/restaurant-chatbots>
- [6] CB Insights. (2022). How restaurants can increase check size using conversational AI and AI-powered recommendation engines. <https://www.cbinsights.com/research/restaurant-conversational-ai/>
- [7] Toast. (2020). Everything You Need to Know About AI for Restaurants. <https://pos.toasttab.com/blog/on-the-line/ai-for-restaurants>
- [8] OpenAI. (2025). OpenAI Platform. <https://platform.openai.com/docs/guides/function-calling>
- [9] USDA. (2025). FoodData Central API. <https://fdc.nal.usda.gov/api-guide.html>
- [10] React. (2025). React Documentation. <https://react.dev/>
- [11] Node.js. (2025). Node.js Documentation. <https://nodejs.org/en/docs/>
- [12] MongoDB. (2025). MongoDB Documentation. <https://www.mongodb.com/docs/>
- [13] <https://roadmap.sh/prompt-engineering>
- [14] <https://docs.aws.amazon.com/>
- [15] Chen, L., et al. (2018). Chatbots for Restaurant Recommendations. *Journal of Hospitality Technology*, 12(3), 45-60.
- [16] Vaswani, A., et al. (2017). Attention is All You Need. *Advances in Neural Information Processing Systems*, 30, 5998-6008.

Eidesstattliche Erklärung

I hereby certify...

.....
Place, date

.....
Signature