|  |  |
| --- | --- |
|  |  |
| GraPro ai chatbot |  |
|  |  |
|  | Graduaatsproef Tiago Belmonte |
|  |  |

Associate Degree Afstudeerwerk

Ai Chatbot via Python

Door

Tiago Belmonte

Afbeelding met Lettertype, Graphics, logo, wit

Door AI gegenereerde inhoud is mogelijk onjuist.

Graduaatsproef

Graduaat Programmeren

Hogeschool Gent

Onder begeleiding van

Lector graduaatsproef

Luc Vervoort

Inhoudsopgave

[GraPro ai chatbot 1](#_Toc199516821)

[Graduaatsproef Tiago Belmonte 1](#_Toc199516822)

[Samenvatting 4](#_Toc199516823)

[Summary 5](#_Toc199516824)

[Toelichting van mijn keuze 6](#_Toc199516825)

[Waarom heb ik gekozen voor een ai assistent? 6](#_Toc199516826)

[Waarom Python 7](#_Toc199516827)

[Hoe ben ik te werk gegaan? 8](#_Toc199516828)

[Start van project 8](#_Toc199516829)

[Toevoegen chatbot via ollama 8](#_Toc199516830)

[Api integratie 9](#_Toc199516831)

[Weer integratie 9](#_Toc199516832)

[Nieuws integratie 11](#_Toc199516833)

[Google kalender integratie 12](#_Toc199516834)

[Spraakherkenning en natuurlijke interactie 13](#_Toc199516835)

[Alles samenvoegen 14](#_Toc199516836)

[Front-end met Tkinter 15](#_Toc199516837)

[Eindresultaat 15](#_Toc199516838)

[Mijn bevindingen 16](#_Toc199516839)

[Moeilijkheden 16](#_Toc199516840)

[Conclusie: wat heb ik geleerd? 16](#_Toc199516841)

[Bronnenlijst 17](#_Toc199516842)

# Samenvatting

De opdracht voor onze graduaatsproef bestond erin om een softwaretoepassing te ontwikkelen in een programmeertaal of omgeving die we tijdens de opleiding nog niet geleerd hadden. Dit moest ons stimuleren om onze eigen grenzen te verleggen en zelfstandig nieuwe kennis op te bouwen.

Voor deze uitdaging koos ik ervoor om een persoonlijke AI-assistent te bouwen in Python, een taal die tot dan toe volledig nieuw voor mij was.

Deze assistent is een desktopapplicatie die via spraakcommando’s met de gebruiker communiceert. De applicatie werkt zoals een chatbot, hij kan al je vragen beantwoorden maar hij kan ook onder andere het weerbericht, nieuwsberichten en agenda-afspraken (via integratie met Google Calendar) tonen. Waardoor het een echte persoonlijke assistent is met ai geïntegreerd

De integratie van ai heb ik mogelijk gemaakt via Ollama, een lokale AI-chatbot die draait op een taalmodel (LLM) en we in de lessen Programmeren Gevorderd 3 van de heer Vervoort kort aan bod kwam. Deze maakt het mogelijk om vragen van de gebruiker te beantwoorden zonder externe API’s of internetconnectie, wat zorgt voor snellere en veiligere interacties. Het is ook bovendien gratis waardoor dat iedereen dat kan raadplegen.

Voor de visuele interface maakte ik gebruik van Tkinter, terwijl speech recognition en text-to-speech zorgen voor een natuurlijke interactie.

Doorheen dit project heb ik mezelf niet alleen een nieuwe taal aangeleerd, maar ook mijn vaardigheden in API-integratie, object georiënteerd programmeren en UI-ontwikkeling sterk verbeterd.

Daarnaast kreeg ik dankzij dit project meer inzicht in de werking van AI-modellen, en groeide dan ook mijn interesse in de wereld van Ai.

GitHub repository: https://github.com/TiagoBelmonte/Graduaat-Programmeren/tree/main/Jaar2/Graduaatsproef

# Summary

The assignment for our graduation project was to develop a software application using a programming language or environment that we had not yet learned during our studies. The goal was to push our boundaries and independently acquire new knowledge.

For this challenge, I chose to build a personal AI assistant in Python, a language that was completely new to me at the time. This assistant is a desktop application that communicates with the user through voice commands. The application can display weather information, news updates, and calendar events (via integration with Google Calendar).

A notable part of the project is the integration of Ollama, a local AI chatbot running on a language model (LLM). This allows the assistant to answer user questions without relying on external APIs or an internet connection, resulting in faster and more secure interactions.

I used Tkinter for the visual interface, while speech recognition and text-to-speech provide a natural user experience.

Throughout this project, I not only learned a new programming language, but also significantly improved my skills in API integration, object-oriented programming, and UI development.

Additionally, I gained deeper insight into how AI models work, and my interest in voice-driven and autonomous assistants grew. This project has laid a solid foundation for me to continue exploring these technologies in the future.

GitHub repository: https://github.com/TiagoBelmonte/Graduaat-Programmeren/tree/main/Jaar2/Graduaatsproef

# Toelichting van mijn keuze

## Waarom heb ik gekozen voor een ai assistent?

De keuze om voor mijn graduaatsproef een AI-assistent te bouwen, kwam voort uit mijn grote interesse in chatbots sinds de opkomst van ChatGPT. Ook in de lessen, toen we kennismaakten met Ollama, werd mijn nieuwsgierigheid verder aangewakkerd.

Daarnaast zag ik dit project als een uitgelezen kans om mijn kennis en vaardigheden rond artificiële intelligentie uit te breiden, aangezien AI tegenwoordig een steeds belangrijkere rol speelt in diverse sectoren.

Reeds bij het indienen van mijn conceptaanvraag vroeg ik expliciet toestemming om een chatbot te ontwikkelen met Python. Na het opzetten van een basischatbot via Ollama – wat technisch gezien voornamelijk neerkwam op het integreren van een package – had ik echter het gevoel dat ik mezelf nog niet voldoende had uitgedaagd.

Daarom besloot ik om verder te gaan dan enkel een klassieke chatbot. Ik breidde het project uit met meerdere API-koppelingen zodat de assistent ook kon antwoorden op praktische, dagelijkse vragen zoals het weeroverzicht en het laatste nieuws – iets wat een gewone chatbot vaak niet standaard kan.

Om de assistent echt persoonlijk te maken, voegde ik bovendien een integratie toe met mijn Google Agenda. Zo werd het niet zomaar een chatbot, maar een volwaardige digitale assistent die mijn afspraken beheert en inspeelt op mijn dagelijkse noden.

## Waarom Python

Na rond te zoeken op het internet en te vragen aan mijn stagementor in welk programmeertaal ik het best een ai chatbot kon maken koos ik voor Python en dit om verschillende redenen.

Allereerst staat Python bekend om zijn gebruiksvriendelijkheid en leesbaarheid. De syntaxis is eenvoudig en duidelijk, wat het gemakkelijk maakt om snel aan de slag te gaan, zelfs zonder voorkennis.

Dit maakte het voor mij een interessante keuze want ik leer graag dingen bij al doende, en met python kon ik direct aan de slag gaan.

Daarnaast biedt Python een uitgebreid ecosysteem aan krachtige bibliotheken, specifiek gericht op AI-ontwikkeling. Denk hierbij aan TensorFlow en PyTorch voor machine learning, NLTK en spaCy voor natuurlijke taalverwerking, en SpeechRecognition voor spraakherkenning.

Voor de grafische interface gebruikte ik Tkinter, dat eenvoudig integreert met de rest van de Python-code. Deze tools hielpen mij om snel werkende oplossingen te bouwen zonder alles van nul te moeten schrijven.

Een ander belangrijk voordeel is de grote en actieve community rond Python. Hierdoor vond ik altijd wel documentatie, tutorials of oplossingen voor problemen waar ik tegenaan liep.

Bovendien is Python uitermate veelzijdig en integreert het eenvoudig met andere technologieën en API’s. In mijn project heb ik onder andere koppelingen gebouwd met API’s voor het weer, het nieuws en mijn Google Agenda. Ook de integratie met Ollama, een lokale AI-chatbot die draait op een taalmodel, verliep vlot dankzij de flexibiliteit van Python.

# Hoe ben ik te werk gegaan?

## Start van project

Ik begon aan dit project door twee korte tutorials te volgen die ik op Youtube vond. Die gaven mij een goede eerste houvast: ik leerde hoe ik een Python-project volledig vanaf nul kon opzetten, hoe ik de nodige libraries moest installeren en integreren, en ik kreeg inzicht in de basisstructuur en syntax van de taal.

Deze eerste stap zorgde ervoor dat ik met vertrouwen aan mijn eigen AI-assistent kon beginnen.

## Toevoegen chatbot via ollama

Het eerste wat ik aan mijn programma heb toegevoegd, was de chatbotfunctie. In eerste instantie probeerde ik een verbinding op te zetten met de API van OpenAI, het bedrijf achter ChatGPT, om hun chatbot te gebruiken. De verbinding opzetten lukte, maar al snel merkte ik dat dit betalend is. Daarom ben ik op zoek gegaan naar een gratis alternatief, en zo kwam ik terecht bij Ollama, een tool die we ook in de les leerden kennen.

Ollama maakt het mogelijk om lokaal een taalmodel, zoals LLaMA, te draaien. Hierdoor vermijd je het gebruik van externe API’s en, nog belangrijker, de bijbehorende kosten. Dit maakt het een ideale oplossing voor wie lokaal wil testen of werken met AI-modellen.

De integratie verliep verrassend vlot. Nadat ik de Ollama-omgeving lokaal had geïnstalleerd en een model zoals llama3 had opgestart, kon ik via HTTP-verzoeken communiceren met de chatbot. Hiervoor gebruikte ik de requests-bibliotheek in Python om de vragen van de gebruiker door te sturen naar het model.

Ik schreef een aparte functie, ask\_ollama(), die een prompt ontvangt van de gebruiker en deze verzendt via een POST-verzoek naar http://localhost:11434/api/chat. Binnen dit verzoek geef ik aan welk model gebruikt wordt en voeg ik het gebruikersbericht toe. Het model antwoordt met een gestructureerde JSON-respons, waaruit ik vervolgens het antwoord van de chatbot filter en teruggeef aan de gebruiker. Afbeelding met tekst, schermopname, software, Multimediasoftware

Door AI gegenereerde inhoud is mogelijk onjuist.

# Api integratie

### Weer integratie

Na de chatbotfunctionaliteit wilde ik mijn AI-assistent uitbreiden met praktische, dagdagelijkse informatie.

Een van de eerste uitbreidingen was het weeroverzicht, zodat gebruikers eenvoudig via spraak of tekst het weer in een bepaalde stad kunnen opvragen.

Hiervoor maakte ik gebruik van de publieke API van OpenWeather, een betrouwbare dienst die actuele weersinformatie aanbiedt. Om dit correct en veilig te integreren, werkte ik met een .env-bestand waarin ik mijn persoonlijke API-sleutel opsloeg. Met behulp van de dotenv-bibliotheek kon ik deze sleutel dynamisch inladen in mijn Python-code, zonder deze hardcoded in het script te zetten – een goede praktijk voor veiligheid en herbruikbaarheid.

De eigenlijke functionaliteit schreef ik in een functie genaamd get\_weather(). Die ontvangt een stadsnaam als invoer en bouwt op basis daarvan een GET-verzoek naar de juiste URL van de OpenWeather API. Ik specificeerde in de parameters dat de temperatuur in Celsius moest worden weergegeven en gebruikte Nederlands als taal voor de omschrijving van het weer.

In de JSON-respons die de API terugstuurt, haalde ik de temperatuur en beschrijving van het weer eruit en presenteerde die in een begrijpelijke zin, bijvoorbeeld: "In Gent is het 18°C met lichte regen." Eventuele fouten, zoals een onbestaande stad of verbindingsproblemen, worden opgevangen via try-except, zodat de gebruiker altijd een duidelijke foutmelding krijgt.

Daarnaast voorzag ik een raw=True parameter om – indien gewenst – de volledige API-respons terug te geven. Dit maakte het makkelijk om te testen of later meer gegevens toe te voegen, zoals wind, luchtvochtigheid of gevoelstemperatuur.

Op deze manier kreeg mijn AI-assistent een nuttige, concrete functionaliteit erbij die vlot integreerde in het bestaande spraaksysteem.

Afbeelding met tekst, schermopname, software

Door AI gegenereerde inhoud is mogelijk onjuist.

### Nieuws integratie

Na het implementeren van het weeroverzicht wilde ik mijn AI-assistent verder uitbreiden met actuele informatie die voor de gebruiker relevant is. Een logische toevoeging was het nieuwsoverzicht, zodat de assistent bijvoorbeeld vragen kan beantwoorden zoals “Wat is het laatste sportnieuws?” of “Geef me het politieke nieuws van vandaag.”

Voor deze functionaliteit koos ik voor de newsdata.io API, een toegankelijke en betrouwbare dienst die wereldwijd nieuws aanbiedt over verschillende onderwerpen. Mijn persoonlijke API-sleutel bewaarde ik, zoals gewoonlijk, in een .env-bestand onder de naam NEWS\_API\_KEY, zodat deze veilig blijft en niet zichtbaar is in de broncode.

Met behulp van de dotenv- en requests-bibliotheken in Python bouwde ik een functie zoals get\_news(topic), waarin je als parameter het gewenste onderwerp meegeeft (bijvoorbeeld "sport" of "politics"). De functie maakt een GET-verzoek naar de endpoint van de newsdata.io API, met parameters voor het land (country=be voor België), de taal (language=nl) en het opgegeven onderwerp (category=sport, politics, …).

Na een succesvolle respons filter ik de eerste paar artikels uit de JSON-data. Ik selecteer bijvoorbeeld de titels en korte beschrijvingen van de meest recente nieuwsberichten en geef deze op een duidelijke manier terug aan de gebruiker. Een voorbeeldantwoord van de assistent zou kunnen zijn:  
"Hier is het laatste sportnieuws: 1. Club Brugge wint met 3-0. 2. Rode Duivels bereiden zich voor op EK. 3. Transfers in de Jupiler Pro League nemen toe."

Net zoals bij de andere API-koppelingen zorg ik voor foutafhandeling via try-except, zodat de gebruiker altijd een duidelijke melding krijgt bij verbindingsproblemen of ongeldige invoer.

Deze toevoeging maakt de assistent nog veelzijdiger: de gebruiker kan nu vragen stellen zoals “Wat is het laatste technieuws?”, “Wat gebeurt er in de politiek?” of gewoon “Geef me het nieuws van vandaag.”, en krijgt telkens relevante, actuele informatie terug.

### Google kalender integratie

Een van de belangrijkste onderdelen om van mijn project een echte persoonlijke assistent te maken, was de koppeling met mijn Google Agenda. Dankzij deze integratie kon de assistent niet alleen algemene vragen beantwoorden, maar ook persoonlijke informatie teruggeven, zoals mijn afspraken voor vandaag, morgen of de komende week.

Voor deze integratie gebruikte ik de Google Calendar API in combinatie met de officiële Python-clientbibliotheek van Google. De toegang tot de agenda wordt beveiligd via het OAuth 2.0-protocol, wat ervoor zorgt dat de gebruiker expliciet toestemming moet geven voordat gegevens opgehaald mogen worden.

De eerste stap was het aanmaken van een credentials.json-bestand via het Google Cloud Platform, waarin de toegangsgegevens van mijn applicatie stonden. Daarna schreef ik de functie get\_calendar\_service(), die zorgt voor de authenticatie. Wanneer de gebruiker het programma voor de eerste keer opstart, wordt er automatisch een browser geopend waarin je toestemming geeft om de kalender te raadplegen. Die toestemming wordt lokaal opgeslagen in een token.json, zodat je dat niet telkens opnieuw hoeft te doen.

Vervolgens schreef ik de functie get\_events\_for\_day(). Deze functie accepteert een parameter ("today", "tomorrow" of "week") en berekent op basis daarvan het juiste tijdsinterval. Daarna stuurt ze een aanvraag naar de primaire kalender van de gebruiker en haalt ze alle afspraken op binnen dat interval.

De gegevens die terugkomen uit de API bevatten onder andere de titel en het tijdstip van elke afspraak.

Door deze koppeling kreeg mijn AI-assistent niet alleen toegang tot algemene informatie zoals weer of nieuws, maar werd hij echt persoonlijk en functioneel inzetbaar, bijvoorbeeld als ochtendhulp om de dagplanning voor te lezen. Het gaf een extra dimensie aan de assistent, en het voelde als een echte integratie van AI in mijn dagelijks leven.

## Spraakherkenning en natuurlijke interactie

Een belangrijk kenmerk van een persoonlijke AI-assistent is dat je ermee kan communiceren op een natuurlijke manier, net zoals met een echte persoon. Daarom wilde ik niet enkel tekstuele input toelaten, maar ook spraakherkenning en spraakuitvoer voorzien. Dankzij deze toevoeging voelt de interactie menselijker aan en kan de gebruiker eenvoudigweg een vraag inspreken in plaats van deze te typen.

Voor de spraakherkenning gebruikte ik de populaire Python-bibliotheek speech\_recognition. Hiermee kon ik audio-opnames van de microfoon omzetten naar tekst. In mijn functie get\_voice\_input() start ik een opname met de microfoon zodra de gebruiker begint te spreken. Deze audio wordt dan doorgestuurd naar de Google Speech Recognition API, die het Nederlands ondersteunt. Het resultaat is een string die verder gebruikt wordt in het programma, alsof de gebruiker het getypt had.

Wanneer de spraak niet verstaan wordt, vang ik dit op met duidelijke foutmeldingen zodat de gebruiker weet dat hij of zij opnieuw moet proberen. Dit zorgt voor een gebruiksvriendelijke ervaring, ook wanneer iets niet meteen goed gaat.

Naast het kunnen verstaan van de gebruiker, vond ik het ook belangrijk dat de assistent zelf kon terugspreken. Hiervoor gebruikte ik de bibliotheek edge-tts, een wrapper rond Microsoft’s text-to-speech-service. Ik koos voor de stem nl-NL-MaartenNeural, die natuurlijk en vlot klinkt in het Nederlands.

De functie speak(text) verwerkt elk antwoord van de assistent, genereert een audiofragment, speelt dit af via pygame, en verwijdert het tijdelijke audiobestand nadien automatisch. Ook hier zorgde ik voor een korte wachttijd zodat alles synchroon en zonder onderbrekingen verloopt.

Dankzij deze twee functies – get\_voice\_input() en speak() – kreeg mijn AI-assistent een natuurlijke dialoogstructuur. De gebruiker spreekt, de assistent luistert, verwerkt de vraag, zoekt het antwoord op en leest het vervolgens luidop voor. Hierdoor voelt de interactie niet enkel functioneel, maar ook menselijker aan – wat precies was wat ik met deze persoonlijke assistent wilde bereiken.

## Alles samenvoegen

Nadat alle functies van mijn AI-assistent waren uitgewerkt, heb ik alles samengebracht in één centrale functie: behandel\_vraag(vraag). Deze functie bepaalt op basis van trefwoorden in de vraag wat de gebruiker bedoelt en roept de juiste logica aan.

Als er sprake is van afspraken of agenda, wordt get\_events\_for\_day() aangeroepen om de agenda van vandaag, morgen of de week op te halen. Bij vragen over het weer detecteert detect\_weather\_question() de stad, waarna get\_weather() en maak\_weer\_zin() het antwoord genereren.

Voor alle andere vragen fungeert Ollama als chatbot via ask\_ollama().

Zo zorgt deze centrale afhandeling ervoor dat alle onderdelen samenwerken en de assistent op een natuurlijke manier kan reageren, afhankelijk van de inhoud van de vraag.

Afbeelding met tekst, schermopname, software

Door AI gegenereerde inhoud is mogelijk onjuist.

### Front-end met Tkinter

De frontend van mijn AI-assistent is op dit moment nog in ontwikkeling. Voor de visuele interface zal ik gebruikmaken van Tkinter, de standaard GUI-bibliotheek in Python. Tkinter is lichtgewicht, eenvoudig te gebruiken en perfect geschikt voor het bouwen van een overzichtelijk venster waarin gebruikers kunnen typen, resultaten kunnen lezen en visueel feedback krijgen van de assistent. Het doel is om een gebruiksvriendelijke en functionele interface te creëren die alle kernfunctionaliteiten op een toegankelijke manier samenbrengt.

# Eindresultaat

Het eindresultaat van mijn AI-assistent zal een eenvoudige maar gebruiksvriendelijke desktopapplicatie zijn, opgebouwd met Tkinter. Bij het opstarten verschijnt er een overzichtelijk scherm met het logo van de assistent centraal in beeld. Onder het logo bevindt zich een duidelijke knop waarmee de gebruiker een gesprek kan starten. Door op deze knop te klikken, begint de applicatie te luisteren naar de gebruiker via spraakherkenning.

Zodra de gebruiker een vraag stelt, verwerkt de assistent deze en geeft hij het antwoord mondeling terug via text-to-speech. Hierdoor ontstaat een natuurlijke, vloeiende interactie tussen mens en machine. Het doel is een vlotte en intuïtieve ervaring te creëren, zonder afleiding of technische barrières voor de gebruiker.

# Mijn bevindingen

## Moeilijkheden

Tijdens het ontwikkelen van mijn AI-assistent kwam ik verschillende uitdagingen tegen. Eén van de grootste moeilijkheden was het werken met externe API’s en het correct afhandelen van de bijhorende JSON-responsen. Ook het beheren van API-sleutels en het correct instellen van .env-bestanden vergde enige opzoeking en testen. De integratie met Google Calendar was eveneens niet vanzelfsprekend, vooral het instellen van OAuth 2.0 en het begrijpen van de Google API-structuur vroeg geduld en nauwkeurigheid.

Daarnaast was het werken met spraakherkenning en text-to-speech niet altijd stabiel: sommige bibliotheken waren gevoelig aan omgevingsgeluid of gaven foutmeldingen bij het afspelen van audio. Tot slot was het niet evident om alles goed te structureren in één centrale functie, zodat de assistent logisch reageerde op verschillende soorten vragen.

## Conclusie: wat heb ik geleerd?

Door aan dit project te werken heb ik ontzettend veel bijgeleerd, vooral omdat ik met Python werkte – een taal die ik voor deze opdracht nog niet beheerst had. Ik heb geleerd hoe ik op een gestructureerde manier kan omgaan met API-integraties, hoe ik spraakherkenning en spraaksynthese praktisch inzet, en hoe ik werk met asynchrone functies in combinatie met gebruikersinput.

Daarnaast heb ik beter leren nadenken over gebruiksvriendelijkheid, en hoe belangrijk een duidelijke structuur en foutafhandeling zijn in een applicatie die met echte gebruikers werkt. Ook leerde ik hoe waardevol het is om problemen stap voor stap aan te pakken, documentatie te raadplegen, en mijn project uit te breiden op een manier die technisch uitdagend én persoonlijk relevant is.

# Bronnenlijst

**LENOVO.** (z.d.). AI Programming with Python. Van https://www.lenovo.com/nl/nl/glossary/ai-programming-with-python/  
**TAILOR IT.** (z.d.). 10 redenen waarom Python zo populair is in 2021. Van https://www.tailorit.nl/certificeringen/10-redenen-waarom-python-zo-populair-is-in-2021/  
**ARXIV.ORG.** (z.d.). The Python ecosystem for scientific computing. Van <https://arxiv.org/abs/1309.3425>  
**WIKIPEDIA.** (z.d.). Python (programming language). Van <https://nl.wikipedia.org/wiki/Python_(programmeertaal)>  
**NEWDATA.IO.** (z.d.). Free News API. Van <https://newsdata.io/>  
**GOOGLE DEVELOPERS.** (z.d.). Google Calendar API Overview. Van https://developers.google.com/calendar/api  
**PYPI.** (z.d.). SpeechRecognition 3.8.1. Van <https://pypi.org/project/SpeechRecognition/>  
**MICROSOFT.** (z.d.). edge-tts – Text to Speech met Azure. Van <https://learn.microsoft.com/en-us/azure/cognitive-services/speech-service/overview>