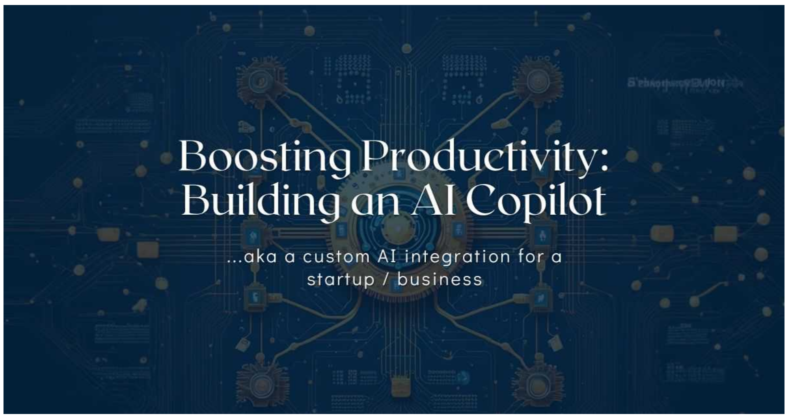
SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Search

Book a Consultation 🚀


Boosting Productivity: Building an AI Copilot
...aka a custom AI integration for a startup / business

Boosting Productivity: Building an AI Copilot

PROFESSIONAL ARTICLE

JUNE 28, 2023

1. Introduction and Motivation

During our daily work, we often get tangled up in little tasks that keep eating our time. Think about all those documents we need to open for a single answer or all the time we spend on Confluence or Notion hunting for the info we need. These tasks are like mosquitoes on a summer night - small but distracting, and they keep us from focusing on the big picture.

ChatGPT has been pretty good at handling some of these issues, much like it has for countless other teams. But we wanted to go a bit further and push the envelope. Here's what was on our wish list:

1. **Collaborative chat**: We wanted to have a real group chat with ChatGPT, not a series of disjointed conversations. ChatGPT does offer share-links, but those aren't real collaborations, as each person continues a separate thread. We wanted a group-based collaboration, and having AI in the group-chat as a part of the team.
2. **Knowledge-base access**: Our partners send us tons of brochures and marketing materials every day. We wanted our Agent

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

(Bot) to know about these resources and use them to give us the right answers when we need them.

3. **Tool integration**: Doing Internet search, going through some pages on our own website or using web tools (navigate to, scrape information from the page, etc) - would be important for us to automate some things, and would increase our productivity.
4. **Cost-effectiveness**: We're all using personal ChatGPT Plus at ~$20/month. But we thought, if we make a separate GPT3.5 & GPT4 agent (mixing these), it could end up being cheaper to use the models via API instead.
5. **Future-proofing**: Lastly, we wanted to make sure we could switch to another Language Model if we needed to. It's like having a backup plan if we decide we're not comfortable sharing sensitive data with OpenAI (i.e. using AWS's Falcon).

TLDR; - skip everything and go straight to the demo.

2. Constructing the AI Copilot with LangChain

In this section, we're going to walk you through the tools that helped us bring this project to life.

- **A. Discord - Our Collaboration HQ**

Although there are a bunch of projects out there, building a separate Front-end would have been overkill for us. We wanted a free, easy-to-use platform where we could collaborate freely - so instead of starting a Slack or Teams workspace, we opted for **Discord** (it also works on all Operating Systems, and has a mobile app) - which checks all the boxes.

Like I mentioned, there are MANY UI projects out there:

CONTACT ME

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

- Quivr offers both Front-end and Back-end.
- Text-generation-webui works with a variety of LLMs, including OpenAI.
- Openplayground similar to OpenAI playground, but compatible with numerous LLMs.
- Langflow offers a full UI to build final agents which can then be exported for production use.

We picked Discord, but the choice is yours. You could integrate with anything you fancy 🙂

- **B. Langchain and its Components**

To be honest it was an obvious choice - I already had some experience with LangChain after building our Home AI Assistant as a module of MagicMirror2 - so the general idea that this project was possible was thanks to LangChain.

What is LangChain - it's a framework around LLMs, it offers building blocks to build various chatbots and other applications around LLMs. It has various abstraction layers, concrete examples and a ton of community contributions, here are a few:

1. 🔗 **Chains** - are sequences of component calls that allow the creation of complex, coherent applications by combining multiple components or chains together in a modular and manageable manner. Think of these as the basic building block, which gets an input, uses LLM as processor and then presents the output.
2. 🕵 **Agents** - are interfaces that provide dynamic and flexible responses to user input. They utilize a set of tools and determine which ones to use based on the input they receive.
3. ⚒ **Tools** and **Toolkits** - are interfaces that an agent can use to interact with the world. Think of: Web Search, Calculator, Python Interpreter, etc.

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

4. 🧠 **Memory** - refers to the capability of retaining information from previous interactions to enhance the user's experience. By default, Chains and Agents in LLMs are stateless, treating each query independently. However, some applications, like chatbots, require remembering past interactions both in the short and long term. This is where the Memory class comes in, capturing, transforming, and extracting knowledge from a sequence of chat messages.
5. 🧮 **VectorStores** - think of these as databases. They convert unstructured data into numerical form (vectors) for efficient storage and similarity-based retrieval.
6. 📃 **Document Loaders** - are tools used to fetch and convert data from various sources into 'Documents', which comprise text and related metadata. They can load data from text files, web pages, or video transcripts, among others, with options for immediate or lazy loading.

I won't go through these in details, instead I'll post here a bunch of links so if you're interested you can go exploring these:

1. Official Python and JS documentations.
2. Collection if examples and tutorials - https://github.com/gkamradt/langchain-tutorials
3. Video Explainer of LangChain + Examples: https://www.youtube.com/watch?v=aywZrzNaKjs
4. A gentle introduction - https://towardsdatascience.com/a-gentle-intro-to-chaining-llms-agents-and-utils-via-langchain-16cd385fca81
5. A ton of videos on various simple and advanced subjects: https://www.youtube.com/playlist?list=PLqZXAkvF1bPNQER9mLmDbntNfSpzdDIU5

CONTACT ME

2023 © Sergiu Nagailic

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

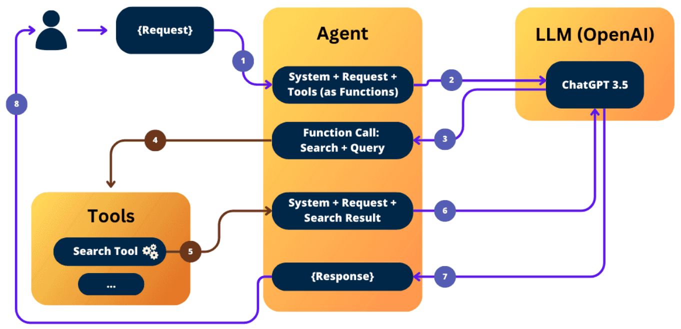
Book a Consultation 🚀

3. Implementation and Tools

Before we get our hands dirty with the implementation, let's understand how an Agent works. We're going to use the BaseSingleActionAgent as an example here.

An Agent is given a set of tools, some prompts (like "You're an XYZ Assistant, your name is ABC..."), and a Language Learning Model (LLM), or a Chain that contains an LLM.

The Agent then might decide to respond to your question using one of the tools (make a *Function Call*), or it might answer without needing any tools at all. Initially, instructing the AI to use a tool required the output to be in a certain format, like specific strings or even pure JSON. You'd then have to parse this output to realize the AI wanted to use a particular tool. Thankfully, with OpenAI's latest updates, things have become a bit simpler. You can read more about that here.


{Request}
Agent
LLM (OpenAI)
1
System + Request + Tools (as Functions)
2
ChatGPT 3.5
8
Function Call: Search + Query
3
4
Tools
System + Request + Search Result
6
Search Tool
5
...
{Response}
7

Let's break-down what's happening here:

1. The user submits a request.
2. The Agent pulls up prompt templates (System Message), adds the user's message, and includes all available tools as Function parameters, prompting the LLM.
3. The LLM decides to use the Search tool and specifies the query.
4. The Search Tool gets activated.
5. Results (or 'observations') are gathered.
6. The Agent collates everything and prompts the LLM once more.

CONTACT ME

2023 © Sergiu Nagailic

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

CONTACT ME

2023 © Sergiu Nagailic

7. The LLM generates the final response.
8. The user receives the final response.

That's a lot of steps, right? But don't worry, all these steps are completed pretty quickly, although the timing does depend on how long it takes the tool to make the observation.

Now that we know how basically an agent functions, let's get to work 🚀 .

Function Calling - creating a custom agent

Starting with a solid base is always a good idea, so I used this template to create my own agent. The initial version was pretty much a carbon copy of the template, but I gradually made tweaks to fit my needs.

Here's what I did:

1. **Added Memory**: I integrated a memory component for retaining information from previous interactions.

```
def plan(
            self,
            intermedia
            callbacks:
            **kwargs:
    ) -> Union[AgentAc
        """Given input
        Args:
            intermedia
            **kwargs:
        Returns:
            Action spe
            :param int
            :param cal
        """
        user_input = k
        agent_scratchp
        prompt = self.
            input=user
            chat_histo
            agent_scra
        )
        messages = pro

        if callbacks i
```

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

```
            callbacks

        predicted_mess
            messages,
        )

        agent_decision
        return agent_d

    @classmethod
    def create_prompt(
        messages = [
            SystemMess

            MessagesPl
            HumanMessa
            MessagesPl
        ]
        input_variable
        return ChatPro
```

2. **Implemented Normalizations:** This step helps take care of situations where functions might get invoked incorrectly, such as with wrongly formatted arguments. Some tools don't always provide explicit guidelines on how their arguments should be used, since they used to rely on custom JSON parsing or other methods.
3. **Inserted Debugging Print Statements:** While tracing is a helpful debugging tool, it doesn't always give you the full picture. That's why I also added a bunch of **pprint** statements to help me keep track of what's going on at any given moment.

Moreover, a new type of agent has been added recently that allows the Language Learning Model to use multiple tools simultaneously (with Function Calls). It can gather all observations and then make a decision. In my implementation, the LLM can only use one tool at a time, in sequence (i.e., it might use Tool 1 twice, then switch to Tool 2, then Tool 3, and finally generate a response). You can check out this new type of agent here.

Remember, the implementation you choose would depend on the specific needs and constraints of

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

your project. It's always a good idea to understand the tools and libraries you're using and adapt them to your use-case.

Creating Custom Tools - Google Search and MTR Search (as examples)

Defining a tool is pretty simple, you have to define the following:

- **Name**: Give your tool a descriptive name.
- **Description**: This helps the AI understand how and when to use the tool.
- **Arguments**: Explain the arguments that the AI should pass to your tool.
- **Run Function**: Define the code that should be executed when the tool is used.

It's as simple as that! Tools can range from other agents to different chains. Let's delve into some examples.

First off, Google Search. There are numerous tools in LangChain that offer Search Engine Result Page (SERP) capabilities, but most of them (excluding DuckDuckGo) need an API key. And, for some reason, they're quite pricey. However, with a bit of time and the use of Playwright (a headless browser), you can get relevant results from Google for free.

Here's my implementation:

```
def google_search(browser,
    print(f'Querying google
    page = get_current_page
    page.goto('https://www.

    page.wait_for_timeout(r
    reject_all_button = pag
    if reject_all_button:
        page.wait_for_timeo
        reject_all_button.c

    page.fill('textarea[nam
    page.query_selector('bo
    button = page.get_by_ro
```

CONTACT ME

2023 © Sergiu Nagailic

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

```
    button.click(position={
    page.wait_for_timeout(r

    page.wait_for_selector(
    results = page.query_se
    print(f'Found {len(resu
    search_results = []
    for result in results:
        title = result.query
        link = result.query_
        description = resul

        # Format the data i
        search_result = "{{
            title.inner_tex
            link.get_attrib
            description.inn
        )
        search_results.appe

    return search_results
```

It’s not as fast, as I make random pauses here and there, to make sure all the elements load properly, but I’m okay waiting 3-5 seconds, if it’s free.

Using a similar approach, I also created a tool that fetches results from our own platform. It does so by executing specific searches and applying certain facets (search filters).

UpdateDB Tool - Document Loading and Google Drive Syncing

I also defined a tool, that once invoked, will:

1. ☁️ Sync Google Drive to a local folder, using: https://rclone.org/drive/
2. 📁 Go through the folder, use various loaders (because I really wanted to use a separate loader for PDFs - PyPDF) extract texts from those files, if needed, break it down into smaller chunks.
3. 🔁 Loop through those and figure out if we really want to vectorize the document or not (so we don’t burn tokens). There might be cases when the document is already in the vectorstore, so you don’t need to re-add it

CONTACT ME

2023 © Sergiu Nagailic

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

CONTACT ME

2023 © Sergiu Nagailic

(unless something was changed and the hash of the document doesn't match)

4. 📝 Add missing snippets
5. ℹ️ Inform back that the update was successful

It's incredibly handy to have this tool. Our non-technical colleagues can just drop 5, 10, or even 50 files into the Drive folder, ask the Agent to update the database, and it's done!

Knowledge-Base (Vectorstore) - as a tool

Creating the Knowledge-Base tool was quite challenging, and I spent considerable time experimenting with it. While I'm not sure if I've devised the optimal approach, it's currently functional and produces good results.

Initially, I just used the Vector Store Toolkit (it comes with 2 Tools): VectorStoreQATool and VectorStoreQAWithSourcesTool. However, I noticed that the default VectorStoreQATool uses RetrievalQA (which is a Chain), and passes the question as the argument to it. However, RetrievalQA uses that question to do 2 things: find the documents matching that question and then use that question to generate the answer. This was suboptimal, as many times I would get a "I don't know" reply.

Let's illustrate with an example: If the question is "What clinics offer dental treatment?" The default behavior is:

❌ **What happens:** It finds no or very few matching documents, resulting in an unhelpful response - "I don't know".

✅ **What I expect:** It should locate any documents referencing "dental treatment", retrieve 5-10 relevant snippets, and then formulate the question: "What clinics offer dental treatment?" This revised question would then be passed to the main chain.

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

For this, I had to alter RetrievalQA, so it gets 2 parameters: filter & question. Filter - to filter all the documents from the database, and then, Question - to actually ask the question against all those snippets.

In my case, I just extend the default RetrievalQA:

```
class RetrievalQASearch(Ret
    """Chain for question-a
    doc_filter: str = None

    def set_filter(self, do
        self.doc_filter = d

    def _call(
            self,
            inputs: Dict[st
            run_manager: Op
    ) -> Dict[str, Any]:
        _run_manager = run_
        question = inputs[s

        search = question i

        docs = self._get_do
        answer = self.combi
            input_documents
        )

        if self.return_sour
            return {self.ou
        else:
            return {self.ou
```

And then I use this as a tool:

```
class KnowledgeBaseInput(Ba
    """Input for KnowledgeB
    doc_filter: str = Field
    question: str = Field(.


class KnowledgeBaseTool(Bas
    name = "knowledge_base"
    description = "Use this
                  " - use O
    args_schema: Type[Knowl
    vectorstore: VectorStor
```

CONTACT ME

2023 © Sergiu Nagailic

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

```
    llm: BaseLanguageModel

    def _run(self, doc_filt
        """Use the tool."""
        retrieval_chain = R
            self.llm,
            retriever=self.
            chain_type_kwar
                'document_p
                    input_v
                    templat

                ),
                'document_s
            }

        )
        retrieval_chain.set_
        return [retrieval_c
```

With this change, I now receive two arguments. I didn't want to override the run method or call method because they're used in various ways and expect the question / query to be a string. Hence, I created a setter that assigns the filter right before running our query.

As the vectorstore and llm are properties, I need to pass them when defining my tool: KnowledgeBaseTool(vectorstore=self.db, llm=llm).

The results with these adjustments were significantly better, with fewer "I don't know" responses. You can inspect the process and see if the correct filter was used. If needed, you can also explicitly tell the AI what filter to use.

4. Testing and Debugging the AI Copilot

LangChain offers multiple ways to debug, and I've found the following to be the most effective for my needs:

1. **Getting the stats from OpenAI**: Understanding how many tokens were used, how many calls were made, etc., is important for maintaining the efficiency of your code and budgeting for token use.

CONTACT ME

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

2. **Tracing**: Observing what tool results the LLM received, among other things, can help in identifying potential issues or points of improvement.

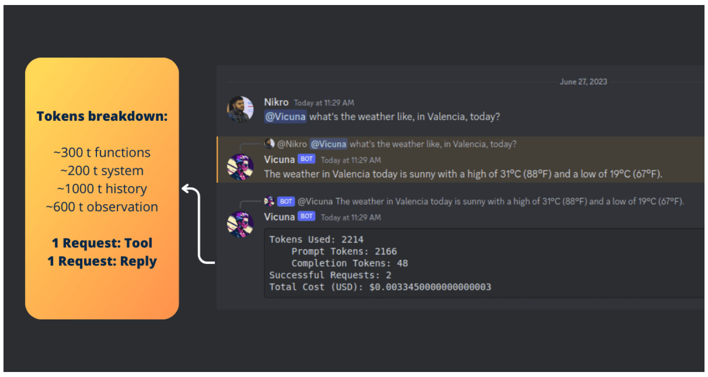
```
openai_counter = OpenAICallI
stdout_handler = ConsoleCal
ev_loop = asyncio.get_event_

future = ev_loop.run_in_exe
response = await future
```

A point to note is that I'm using Discord and its Python library, **discord.py**, which uses event loops. You can disregard the ev_loop part. Essentially, I pass two callback handlers to the chain.from langchain.callbacks.tracers.stdout import ConsoleCallbackHandler

This line is particularly crucial. There are many callback handlers available, including Stdout, but the ConsoleCallbackHandler is particularly useful as it goes deeper into sub-calls, providing a detailed view of what's happening inside.

Here's an illustrative image:



Nikro Today at 11:29 AM
@Vicuna what's the weather like, in Valencia, today?

@Nikro @Vicuna what's the weather like, in Valencia, today?
Vicuna BOT Today at 11:29 AM
The weather in Valencia today is sunny with a high of 31°C (88°F) and a low of 19°C (67°F).

BOT @Vicuna The weather in Valencia today is sunny with a high of 31°C (88°F) and a low of 19°C (67°F).
Vicuna BOT Today at 11:29 AM

```
Tokens Used: 2214
    Prompt Tokens: 2166
    Completion Tokens: 48
Successful Requests: 2
Total Cost (USD): $0.0033450000000000003
```

In addition, the **openai_counter** gradually compiles the statistics of all the calls made. In Discord, I typically reply to my own message with these statistics:

```
# This prints-out the stats
```

CONTACT ME

```
await sent_message.reply(f'
```

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

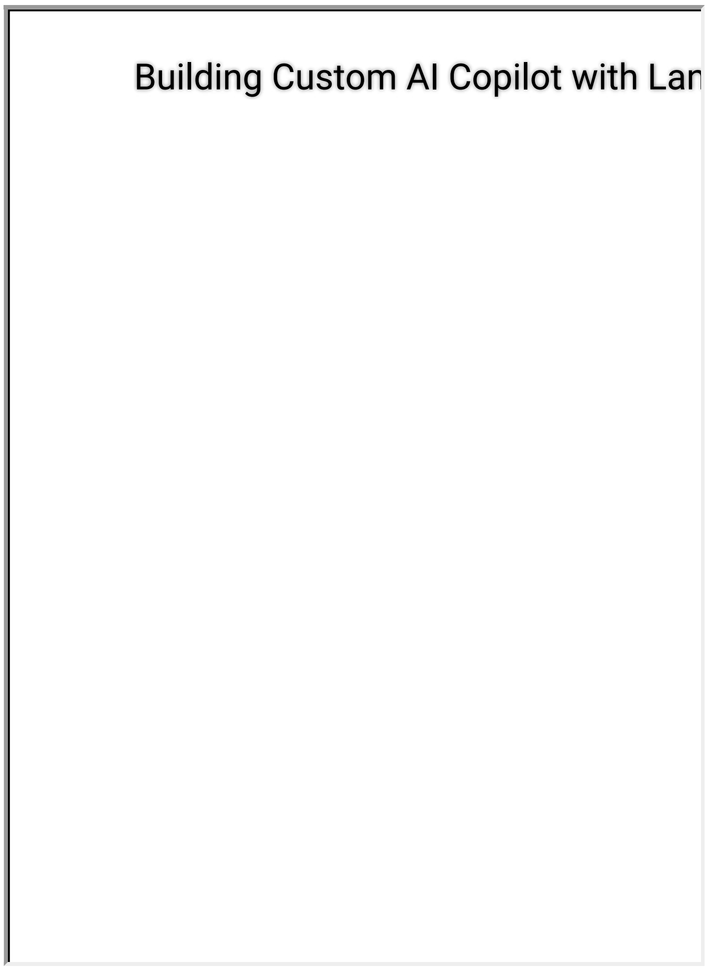
Doing this provides a handy summary of your AI's operations and helps track and optimize resource usage.

5. AI Copilot in Action

Wow, you made it that far (unless you skipped everything 😀) - great! Let's see our Agent in action.

- **Google Search**

Well, you already saw an example above, but here's a little video example as well:



Building Custom AI Copilot with Lan

- **Using MTR Search**

Let's look up some information results from our own platform:

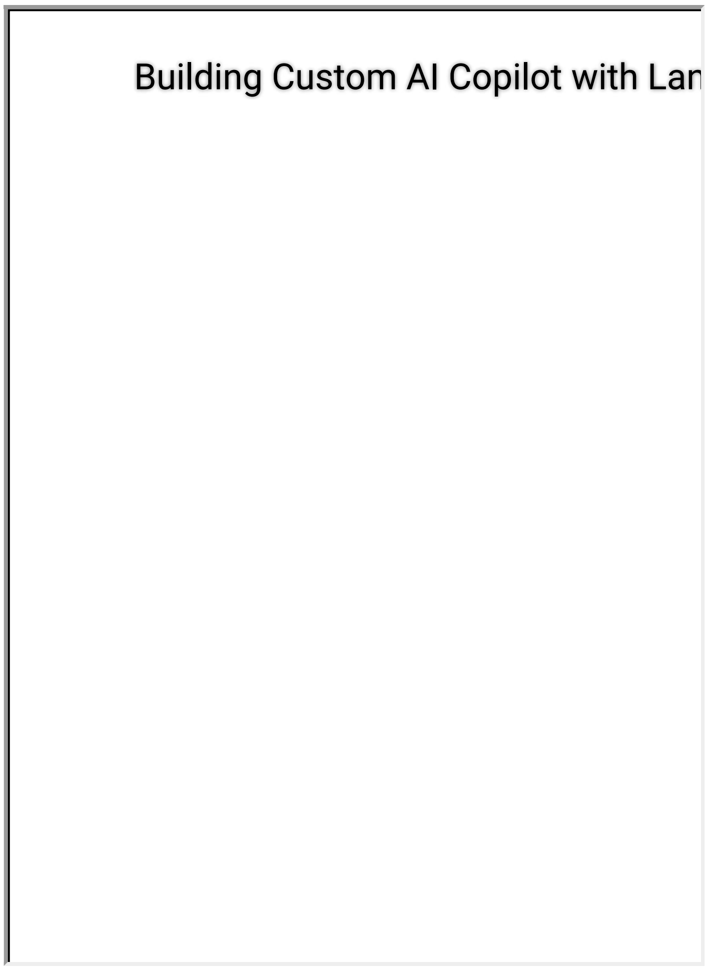
CONTACT ME

2023 © Sergiu Nagailic

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀



- **Adding files to the Knowledge-Base and Using the Knowledge-Base**

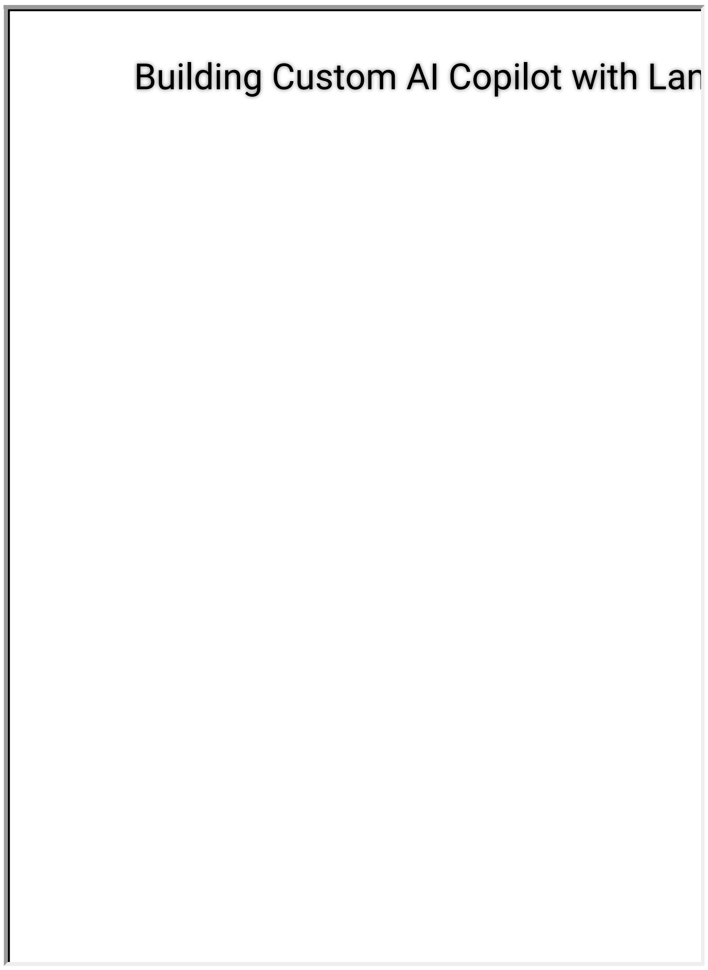
Let's check how a simple user might add a file to the knowledge-base, update it and then ask bot a question:

CONTACT ME

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀



So, there you have it, folks!

If you've been following along, you've got everything you need to build your **own AI Copilot** that can supercharge your startup or business. But hey, if this all sounds like an exciting journey, yet you're unsure about taking the plunge solo, don't worry! That's where I come in. I can help you navigate this uncharted territory and tailor the AI to suit your needs. Together, we can turn your business into a productivity powerhouse. Interested? You can connect with me directly on LinkedIn or schedule a chat on Calendly. I can't wait to **collaborate** and help take your business to the next level!

Tags:

CONTACT ME

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

#LANGCHAIN #COPILOT

#CHATGPT

Categories:

#AI

Comments:

Feel free to ask any question / or share any suggestion!

CONTACT ME

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

What do you think?

1 Response

Upvote Funny Surprised Sad

0 Comments 1 Login ▼

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

♡ Share

Best Newest Oldest

Be the first to comment.

Subscribe Privacy

Do Not Sell My Data

CONTACT ME

SERGIU NAGAILIC

Senior Drupal Developer - CTO & Co-founder at MedicalTourism.Review

Book a Consultation 🚀

ALSO ON **NAGAILIC SERGIU**

Crafting Our AI Assistant

Gatsby.JS Live Pr
with Drupal

CONTACT ME