# Building HAL 9000 (And It Runs Completely on My Mac)

HAL 9000 is a fictional artificial intelligence (AI) character in Arthur C. Clarke's novel "2001: A Space Odyssey" and its film adaptation directed by Stanley Kubrick. It had a conversational interface—humans could just talk to it like humans talk with each other. It was super intelligent. The original idea came about in 1964 when Kubrick and Clarke started working on the project.

The year 1964 was when the Ford Mustang was introduced. Rotary phones were how you made phone calls, and you wore thick black polycarbonate glasses. Microwave ovens were a new thing, as were color TVs. As the world danced to “Pretty Woman” by Roy Orbison, or “Twist and Shout” by the Beatles, society was looking forward to the seemingly impossible goal of putting man on the moon by the end of the decade.

It’s in that enchanting time that HAL 9000 was imagined, a super intelligent computer program that could control the functions of a spaceship, that was self-aware, and people could interact with in natural language.

Fast forward to 2024. Although I don’t quite yet have my personal flying spaceship, HAL 9000 is a pretty close reality. Nerd that I am, I set out to build HAL 9000 for myself.

In my last article in CODE Magazine, I talked about running AI locally. The goal of this article is different but straightforward. I want to build a HAL 9000 that I can speak to, in any language, about any topic. And it should give me answers about whatever I ask for. Additionally, I want to do it all on my off-the-shelf commercially available MacBook Pro. Finally, I want to be able to build it so that it runs completely offline, so that in the rare case I manage to get a spaceship, I don’t have to rely on an internet connection to run it.

As I build it, I’ll share all of the code and explain it as I go. In the end, I’ll put together a fully functioning application, HAL 9000.

## What You’re Going to Need

To follow this article, you’ll need a beefy machine. You’re not going to rely on the cloud to build the model for you. You’ll need a powerful local compute capability. This means either a higher-end Windows/Linux laptop or one of the newer Macs. And yes, you’ll need a GPU. AI involves a lot of calculations and to speed things up, a lot of them are offloaded to the GPU. The difference between doing everything on the CPU vs. GPU is astronomical. For my purposes, I’ll use my rusty trusted M1 Max MacBook Pro. It’s a few years old, but it has enough oomph to work on thousands of pages of text, which is good enough for my needs. Hopefully, you have a similarly equipped machine, or, to follow along you could just use a smaller input dataset.

Also, you’ll download and use standard libraries, packages, and large language models that other companies and people have built. But when you’re done with it, no data will be sent to the cloud, and your application will have the ability to be able to run completely offline. To get started, though, you’ll need an internet connection.

Also, I will use Python, so ensure that you have Python 3x installed.

## The Main Components

Let’s think about the problem and break it down into smaller parts.

I’ll need the ability to listen and convert my spoken text into ASCII text. When I speak into my mic, saying “Let’s talk about New York City,” my program should be able to transcribe the text I say on the fly.

I’ll need a large language model (LLM) that takes my spoken text, transcribed to plain text, as inputs, and returns a sensible response.

And finally, I’ll need the ability to take the LLM’s response and convert it to audio, which I can then hear through my speakers.

All this put together should give me a super intelligent sentient being.

## Enter Hugging Face

Hugging Face ([http://huggingface.co](http://huggingface.co/)) is a popular open-source AI community and platform focused on natural language processing (NLP) and transformer-based models. It has a pretty impressive transformers library, a number of pre-trained models, a model hub where you can find or contribute to models, a large number of datasets for your own experimentation, and so much more. I figured that this would be a great place for me to start exploring what I can build.

For my purposes, models are what I care about. I went to <https://huggingface.co/models>, started looking at the various models, and found that there are a number of impressive models available. As I started playing with them, I started discovering superpowers. For instance, a long time ago, I saw that Microsoft Cognitive services (now known as Microsoft AI services) had a REST API that you could just show a picture to and it would detect what it saw in the picture. Or draw bounding boxes around things it saw in the picture. There’s a whole section of models dedicated to computer vision. I tried a few capabilities there, like I showed it a picture of my dogs, and it immediately recognized the dogs in the picture, even the breeds; or I could have a conversation about the image, like “Tell me more about a Doberman” etc. Theoretically speaking, I could just show my AI bot a picture from my webcam, and start talking about it. Or I could run it on my phone and show it a weed, have it recognize the plant, and then tell me how to get rid of it. Or how about a bunch of scanned receipts and ask something like, “How much did I spend on Burritos last year?” I’ll leave computer vision for another day. For now, let’s refocus on audio.

There are a bunch of models available under the Audio section of the models page in Hugging Face. There are text-to-speech models, text-to-audio, automatic speech recognition, audio-to-audio, audio classification, and voice activity detection.

For my needs, I already know that I will find text-to-speech and automatic speech recognition useful.

## Automatic Speech Recognition

This is the first problem I need to solve. I need to be able to speak to the computer, hopefully in any language, and it should be able to transcribe the texts with decent accuracy. I noticed that one of the models available was openai/whisper. So I decided to play with it.

Whisper (<https://github.com/openai/whisper>) is a general-purpose speech recognition model. It’s trained on a large dataset of diverse audio and is also a multitasking model that can perform multilingual speech recognition, speech translation, and language identification. In order to use Whisper on my Mac, I needed to install FFmpeg first. Well, that’s easy. I was able to install that using the command below.

brew install ffmpeg

With the above in place, I started writing my first simple Whisper-based program. The first step is to define a Python .venv, which I’ll skip the details of since I assume you’re already familiar with the basics of Python.

With that in place, let’s define the requirements.txt, which is shown below

soundfile

pyaudio

SpeechRecognition

git+https://github.com/openai/whisper.git

I defined a launch.json in my .vscode folder that allowed for debugging, which can be seen in **Listing 1**. And now my canvas was set up to start playing with Whisper. I went ahead and wrote the following simple program:

import whisper

model = whisper.load\_model("base")

result = model.transcribe("audio.mp3")

print(result["text"])

To my shock, whatever text was spoken in audio.mp3, Whisper transcribed accurately into text. I do notice that these open-source models show a lot of errors and warnings. Those errors and warnings are worth paying heed to. For the purposes of this article, I want to keep my outputs clean, so I suppressed them, as follows.

import logging

import warnings

warnings.filterwarnings('ignore')

for name in

logging.Logger.manager.loggerDict.keys():

logging.getLogger(name).setLevel(

logging.CRITICAL)

Okay, I’m excited. My simple audio transcription is working. I also want to be able to talk into a microphone and have Whisper transcribe it on the fly. I played around a bit with Whisper and was able to write code, as shown in **Listing 2**. This code allows me to speak into the microphone, and Whisper can detect it. A curious thing you see in **Listing 2** is “device\_index=7”. What is that magic number 7? Well, 7 isn’t just my lucky number; it’s the index of the microphone I wish to listen to. To list all microphones on your computer, just use the below code snippet:

for index, name in

enumerate(

sr.Microphone.list\_microphone\_names()):

print("Microphone with name \"{1}\"

found for `Microphone(device\_index={0})`".

format(index, name))

Now let’s run my code and see how it works. To run the code in VSCode, I simply hit F5, and once the code prints “Say something,” I just said whatever I wished. The output can be seen in **Figure 1**.

A screen shot of a computer

Description automatically generated

**Figure 1:** Speech recognition seems to work.

As exciting as this is, I couldn’t help but wonder: What if I wanted to make this smarter? As in, be able to speak in any language, detect the language, and use that detected language to both chat with my LLM and use it for audio transcription. This can be done using the code shown in **Listing 3**.

This is incredible, but I’m going to keep things simple by limiting this article to English. You can imagine how easy it would be to modify this bot to work in any language by just detecting the language being spoken and passing that as an input to the recognized language to converse in.

As impressive as this is, I want my audio transcribing to work continuously. In other words, until I say a catch phrase like “Goodbye,” I want my program to keep doing audio transcriptions. After all, as I’m writing the chat bot, I’m going to have a conversation with it. I’ll say things like, “Let’s talk about New York City,” and it’ll tell me some general information about New York City, and then I might ask further questions based on the context of the answer.

The code to listen for audio continuously until I say “Goodbye” can be seen in **Listing 4**. This code running in my VSCode’s debug output can be seen in **Figure 2**.

A computer screen shot of a black screen

Description automatically generated

**Figure 2:** Output of continuously recognizing spoken text

One thing I’ll say about the output you see in **Figure 2**, is that although the transcription is shockingly accurate, and works across multiple languages, I did notice that any background noise can easily confuse it. So if you’re following me along in actual code, try and do this in a quiet environment. That said, there are tweaks you can make to tune out background noise.

All right, I think we have the first ingredient of the bot all done. Let’s make it smarter by connecting my spoken text to a large language model.

## Connecting My Text to a large LLM

A large language model (LLM) is a type of artificial intelligence (AI) designed to process and understand human language, typically using deep learning techniques. There are many large language models, specialized for various needs. You could pick from any number of large language models available on Hugging Face. All the big popular names like Llama and Gemma and Phi, etc., are available.

Now, large tech companies have spent billions of dollars building these models, so they don’t just give it away. For most models, you’ll have to fill out a form and acknowledge terms of use, and, for some models like Llama3.1, you must wait for approval. In certain jurisdictions, they may not allow you to use the model at all.

You could take any of these models, and fine tune them also. Fine-tuning a Large Language Model (LLM) involves adjusting the model's weights and parameters to better perform a specific task or adapt to a particular domain. By fine tuning, you can improve performance for a specific task, you could adopt to your specific domain, you can reduce bias, or you might improve model generalizability, as need be. I’ll leave fine tuning for a future article. For now, I feel that a generic LLM will suffice.

I started playing around with a few models, and decided I’ll use Gemma for this article. Gemma (Generative Expert Memory Model Architecture) is an AI model developed by Google. Google has put in all the hard work already. They’ve trained it on 45 terabytes of data, and it’s incredible and knowledgeable in so many fields. I found Gemma to be great for conversational AI, and with minimal prompt engineering, I was able to get it to give me coherent answers that were fun and useful. I’m not saying Llama is bad; to be honest, all of these models are quite comparable to each other.

Let’s start building the app with Gemma. Because I’m running everything locally, I went with the 2-billion parameter version of Gemma. The more parameters, the better your accuracy, but the more beefy computer you’re going to need to run this. How about using LangChain to do something smart? When you have access to puny hardware, run the 2-billion parameter model, and when you have a server, call out to the server running the 45B parameter. When you’re online, call out to OpenAI and some model there.

To use this model, I had to visit <https://huggingface.co/google/gemma-2-2b-it> and on the right hand top, click on the “Use this model” button and select “Using Transformers”. Right there it showed me some example code. Well, this is too easy.

I must say that you’ll have to frequently fill out an acknowledgement form, and the team building Llama must allow you to use a model. This is as simple as going to the page for Llama on Hugging Face, filling out a simple form, going to your Hugging Face account settings, and creating an access token for yourself at <https://huggingface.co/settings/tokens>. With all that in place, just include the below snippet in your code and you’re good to go.

from huggingface\_hub import login

login("yourtoken")

Now back to Gemma, where you don’t need to have an authentication token, let’s start writing code for the conversational bot. The first step is to create a pipeline object, as can be seen below.

import torch

from transformers import pipeline

pipe = pipeline(

"text-generation",

model="google/gemma-2-2b-it",

model\_kwargs={"torch\_dtype": torch.bfloat16},

device="mps",

)

The “mps” is because I’m running on a Mac. If you’re on a PC with an NVIDIA card, just replace that with “cuda”.

A transformer pipeline in AI refers to a sequence of processing stages that use transformer architectures to perform specific tasks. At the bare minimum, a pipeline takes a model to make predictions from inputs, and a tokenizer for mapping raw text inputs to a token. A tokenizer is simply a component that splits text into individual words, phrases, or sub words, called tokens. For instance, you may have something like, “take some text and do sentiment analysis on it,” and you’d need a model to perform this task. Using that model, you’d create a pipeline. Although not the focus of this article, I wanted to show you how simple it is to build sentiment analysis. You can see the code for sentiment analysis using a popular model in **Listing 5**. As you can see, it’s a matter of having a model and a tokenizer, building a pipeline and using it.

It would be cool to pair this sentiment analysis with the simple old bot and make it smart enough that if the conversation is becoming too sad, it’ll throw a joke in there for fun. I’ll leave that as an experiment for you to do.

But let’s refocus on my intelligent bot and get back to building it. With my pipeline set up, I can give it an input as “prompt” and the large language model returns me the output I’m looking for. This is as simple as the code snippet you see below.

messages = [

{"role": "user", "content": prompt},

]

outputs = pipe(messages, max\_new\_tokens=256)

assistant\_response =

outputs[0]

["generated\_text"][-1]["content"].strip()

Feel free to run this, and you’ll be able to get an answer to any question you may have. But that’s not what we’re trying to do. We want our bot to be smarter. We want it to understand context. So if I say, “Let’s talk about dogs” with a subsequent question about “What is bark?”, I want to know about dogs barking. But if I say, “Let’s talk about trees” with a subsequent question of “What is bark?”, I wish to be told about tree bark.

Context is important. For example, “my dogs love to play in leaves” or “my dogs are not happy when their owner leaves”, have two entirely different meanings for the same word.

There are two ways to attach context. One is that instead of **role : user**, you can just attach all previously said text as **role : assistant**. Alternatively, the prompt can just remember all previously generated text. I’ll use the latter approach and put together a full code example, as can be seen in **Listing 6.** Notice that in **Listing 6** I have also added a prompt of “Answer in brief.” I found Gemma to be a bit too wordy. Or maybe I’m just impatient.

My interaction with Gemma can be seen in **Figure 3**. Remember, this is me typing into a keyboard.

A screenshot of a computer

Description automatically generated

**Figure 3:** Me talking with my chatbot

Notice that all the processing is being done locally; look at how hard my GPU is working in **Figure 4** when I ask it all these questions.

A screen shot of a computer

Description automatically generated

**Figure 4:** My hardworking GPU

There are some other interesting things you can see in **Figure 3**. My second question said, “Answer in pirate style” and the model did. This is so funny. But in my next question, I just said “What can I do there?” But what is “there”? My model understood from the context that I’m still talking about New York City.

If you’re curious, I played the “Let’s talk about dogs” and “Let’s talk about trees” game and asked “Tell me about bark.” Here are the outputs I received:

[AI Response]

Bark is a dog's way of communicating, like a "hello" or a "warning." It's a complex sound with many meanings depending on the context and the dog's tone.

Bark is fascinating! It's the tree's protective outer layer, a shield against insects, disease, and the elements. It also plays a role in water and nutrient transport, and can even change color and texture with age.

## Putting It All Together

I think we have a pretty impressive application in the works. Let’s put all this together now. I can convert text to audio, I can listen continuously. I can converse with an LLM. I can chat based on context. Putting **Listing 4** and **Listing 6** together, I get **Listing 7**, which is my fully functional chatbot that I can speak with.

Let’s give it a try. If you’re online, I put together a few examples of this program running.

You can see me chat about Microsoft Graph here: <https://www.youtube.com/watch?v=8vJtldKwxKw>. Or you can see me chatting about airplanes here: <https://www.youtube.com/watch?v=rnn9hLdvWv4>. Or you can see me learn about securing JavaScript applications here: <https://www.youtube.com/watch?v=OGnwGJgiABQ>.

It’s probably more compelling to watch a video and hear me talk and get a feel for how this works interactively than to see it in text, but if you’re not online, let’s see some fun interaction here.

Let’s say I’m throwing a party and I’ve never cooked anything. My first question to AI is:

[AI Query]

Teach me how to cook.

And it gives me a nice, detailed output, as can be seen in **Figure 5**.

A screenshot of a computer program

Description automatically generated

**Figure 5:** AI helping me with cooking

I like the “Start simple” bit, so let me ask how to make a grilled cheese.

[AI Query]

How do I make a grilled cheese?

As expected, AI gives me quick five steps to make a grilled cheese. This can be seen in **Figure 6**.

A black background with yellow text

Description automatically generated

**Figure 6:** AI helping me with cooking

Now, that I’ve established a context, I can just ask a subsequent question.

[AI Query]

Will it make me fat?

To which AI promptly replies,:

[AI Response]

No, a grilled cheese is unlikely to make you fat if you eat it in moderation as part of a balanced diet.

Well, that’s good to know! Feel free to keep this conversation going about any other topic you wish.

## Text to Audio

All this chat about grilled cheese is making me hungry, so let me leave you with a little teaser. You’ve so far built a fully functional chatbot. I’ve played a bit with it: I asked it about Microsoft Graph, programming in general, security-related stuff, I asked it about cooking as you saw, and I asked it about touristy stuff. I asked it about investing. I asked it about historical events. In every instance, my jaw was on the floor.

I did talk about a few extensibility points, like detecting language, and adding sentiment analysis. But to round up the chatbot, let’s add the last bit, which is text to audio.

This is where the real world kicks in. In the movie, HAL answered questions that were suited to short audio interactions. That isn’t how the real world operates. Sometimes the output is code. Sometimes it’s images. Other times it’s bulleted lists. Just look at **Figure 5**. Now imagine closing your eyes and having that text read out to you as audio. I wouldn’t find that very useful, to be frank. It’s easier to read bulleted lists yourself than to have them read out to you.

Still, because I set out to do a full audio-based interaction, let’s add text to speech also.

Back on Hugging Face, I found the **2noise/chatts** model as the most popular text-to-audio model. It was easy to put together a code example that converted any input text to a pretty decent quality of spoken audio. You can see the code for text-to-audio in **Listing 8**. In fact, I was able to visit [https://chattts.com](https://chattts.com/) and tweak the inputs to figure out what parameters worked best for me.

I’ll leave it up to you to integrate it into **Listing 7**, but as I said, I didn’t find it very useful except for simple questions that had straightforward and to-the-point answers. For example, quick math questions, or asking factual questions that didn’t need long drawn-out answers.

## Summary

Did I just build a super intelligent chat bot that I can talk with about any topic and it replies intelligently? Yeah, I just did. And I’ve been using it to learn about all sorts of stuff. I’ve heard stuff like the federal reserve is lowering rates and it’s taught me all about it.

I find it simply amazing that what was purely science fiction in 1964, I was able to effectively build over the weekend. Just to drive home the point of how incredible this is, what else was considered science fiction in 1964? What’s considered science fiction today? Teleportation? Cloning? Travel at the speed of light? Time travel? Admittedly, 1964 was 60 years ago, but that’s still within one person’s lifetime.

Can you imagine, in the year 2094, some random guy using off-the-shelf hardware to clone himself, time travel back to 2024, and write an article to show how to build HAL?

Yeah, neither can I.

Well, I’m off to make a grilled cheese. You have fun.

Sponsored Sidebar

Stay Connected with CODE Magazine on LinkedIn

Join our professional community on LinkedIn to stay ahead in the tech world.

**Exclusive Content**: Get the latest tech insights and articles.

**Industry Updates**: Stay informed with cutting-edge news.

**Networking:** Connect with peers and industry leaders.

Help us grow by sharing with colleagues and friends.

**Follow us today:** [www.linkedin.com/company/**code-magazine**](https://www.linkedin.com/company/code-magazine)

Listing 1: My launch.json to enable Python debugging

{

"version": "0.2.0",

"configurations": [

{

"name": "Python Debugger: Current File",

"type": "debugpy",

"request": "launch",

"program": "${file}",

"console": "integratedTerminal"

}

]

}

Listing 2: Live audio transcribing using Whisper

import speech\_recognition as sr

from speech\_recognition import

Microphone, Recognizer, UnknownValueError

r = sr.Recognizer()

with sr.Microphone(device\_index=7) as source:

print("Say something!")

audio = r.listen(source)

try:

print("You said:" +

r.recognize\_whisper(audio,

language="english"))

except sr.UnknownValueError:

print("Didn't understand")

except sr.RequestError as e:

print(f"Could not request results; {e}")

Listing 3: Audio transcribing in any language

import whisper

model = whisper.load\_model("base")

audio = whisper.load\_audio("audio.mp3")

audio = whisper.pad\_or\_trim(audio)

mel = whisper.log\_mel\_spectrogram(audio).to(model.device)

\_, probs = model.detect\_language(mel)

print(f"Detected language: {max(probs, key=probs.get)}")

options = whisper.DecodingOptions()

result = whisper.decode(model, mel, options)

print(result.text)

Listing 4: Listen for spoken text continuously

import os

import speech\_recognition as sr

from speech\_recognition import

Microphone, Recognizer, UnknownValueError

def audio\_callback(recognizer, audio):

try:

prompt = recognizer.recognize\_whisper(

audio, model="base", language="english")

print(prompt)

if "bye" in prompt.lower():

stop\_listening(wait\_for\_stop=False)

os.\_exit(0)

except UnknownValueError:

print("There was an error processing the audio.")

recognizer = Recognizer()

microphone = Microphone(device\_index=7)

with microphone as source:

recognizer.adjust\_for\_ambient\_noise(source)

stop\_listening =

recognizer.listen\_in\_background(microphone,

audio\_callback)

input() # wait to exit

stop\_listening(wait\_for\_stop=False)

Listing 5: An example of a pipeline for sentiment analysis

import pandas as pd

from transformers import

pipeline, AutoTokenizer,

AutoModelForSequenceClassification

# Load pre-trained model and tokenizer

model\_name =

"distilbert-base-uncased-finetuned-sst-2-english"

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

model =

AutoModelForSequenceClassification.from\_pretrained(

model\_name)

# Create pipeline

classifier = pipeline("sentiment-analysis",

model=model, tokenizer=tokenizer)

# Example text

text = "I loved the new movie!"

# Run pipeline

result = classifier(text)

print(result)

Listing 6: My fancy text based chat bot

import torch

from transformers import pipeline

pipe = pipeline(

"text-generation",

model="google/gemma-2-2b-it",

model\_kwargs={"torch\_dtype": torch.bfloat16},

device="mps",

)

def generate\_text(prompt, previousResponses):

prompt = prompt + ". Answer in brief."

allPrevResponses = ""

for previousResponse in previousResponses:

allPrevResponses += previousResponse + "\n"

messages = [

{"role": "user", "content": allPrevResponses +

"\n" + prompt},

]

outputs = pipe(messages, max\_new\_tokens=256)

assistant\_response =

outputs[0]["generated\_text"][-1]["content"].strip()

return assistant\_response

previousResponses = []

while True:

user\_input = input("\033[92m >> You: \033[0m")

response = generate\_text(user\_input, previousResponses)

previousResponses.append(response)

print("\033[93m >> AI:", response, "\033[0m")

Listing 7: My audio driven chat bot

import os

import speech\_recognition as sr

from speech\_recognition

import Microphone, Recognizer, UnknownValueError

import torch

import torch

from transformers import pipeline

import logging

import warnings

warnings.filterwarnings('ignore')

for name in logging.Logger.manager.loggerDict.keys():

logging.getLogger(name).setLevel(logging.CRITICAL)

pipe = pipeline(

"text-generation",

model="google/gemma-2-2b-it",

model\_kwargs={"torch\_dtype": torch.bfloat16},

device="mps",

)

def AskAI(prompt, previousResponses):

prompt = prompt + ". Answer in brief."

allPrevResponses = ""

for previousResponse in previousResponses:

allPrevResponses += previousResponse + "\n"

messages = [

{"role": "user",

"content": allPrevResponses + "\n" + prompt},

]

outputs = pipe(messages, max\_new\_tokens=256)

assistant\_response =

outputs[0]["generated\_text"][-1]["content"].strip()

return assistant\_response

previousResponses = []

def audio\_callback(recognizer, audio):

try:

prompt = recognizer.recognize\_whisper(

audio, model="base", language="english")

print("\033[92m >> You: " + prompt + " \033[0m")

print("\r Thinking ")

response = AskAI(prompt, previousResponses)

previousResponses.append(response)

print("\r\033[93m >> AI:", response, "\033[0m\n")

if "bye" in prompt.lower():

stop\_listening(wait\_for\_stop=False)

os.\_exit(0)

except UnknownValueError:

print("There was an error processing the audio.")

recognizer = Recognizer()

microphone = Microphone(device\_index=7)

with microphone as source:

recognizer.adjust\_for\_ambient\_noise(source)

stop\_listening =

recognizer.listen\_in\_background(

microphone, audio\_callback)

print("\n ------------------------------ \n

I am your super friendly AI,

what do you wanna chat about today? \n ")

input() # wait to exit

stop\_listening(wait\_for\_stop=False)

Listing 8: Text to audio

import sounddevice as sd

import ChatTTS

chat = ChatTTS.Chat()

chat.load(compile=True)

texts = [

"how are you?"

]

params\_infer\_code =

ChatTTS.Chat.InferCodeParams(

temperature = .3,

top\_P = 0.7,

top\_K = 20,

)

wavs = chat.infer(texts,

params\_infer\_code=params\_infer\_code)

sd.play(wavs[0][0], 24000, blocking=True)