

0. ORGANIZATION
1. DIGITAL TECHNOLOGIES
2. SENSORS
3. ACTUATORS
4. COMPUTER VISION
5. GENERATIVE AI
6. NATURAL LANGUAGE PROCESSING
7. USER INTERFACES
8. CLOUD SERVICES
9. DATABASES

The slides are meant as visual support for the lecture.  
They are neither a documentation nor a script.

Please do not print the slides.

Comments and feedback at [n.meseth@hs-osnabrueck.de](mailto:n.meseth@hs-osnabrueck.de)

# ORGANIZATION

ILIAS

Microsoft Teams

sessions

group work

examination

working environment



visual studio code

python

tinkerforge

git

# DIGITAL TECHNOLOGIES

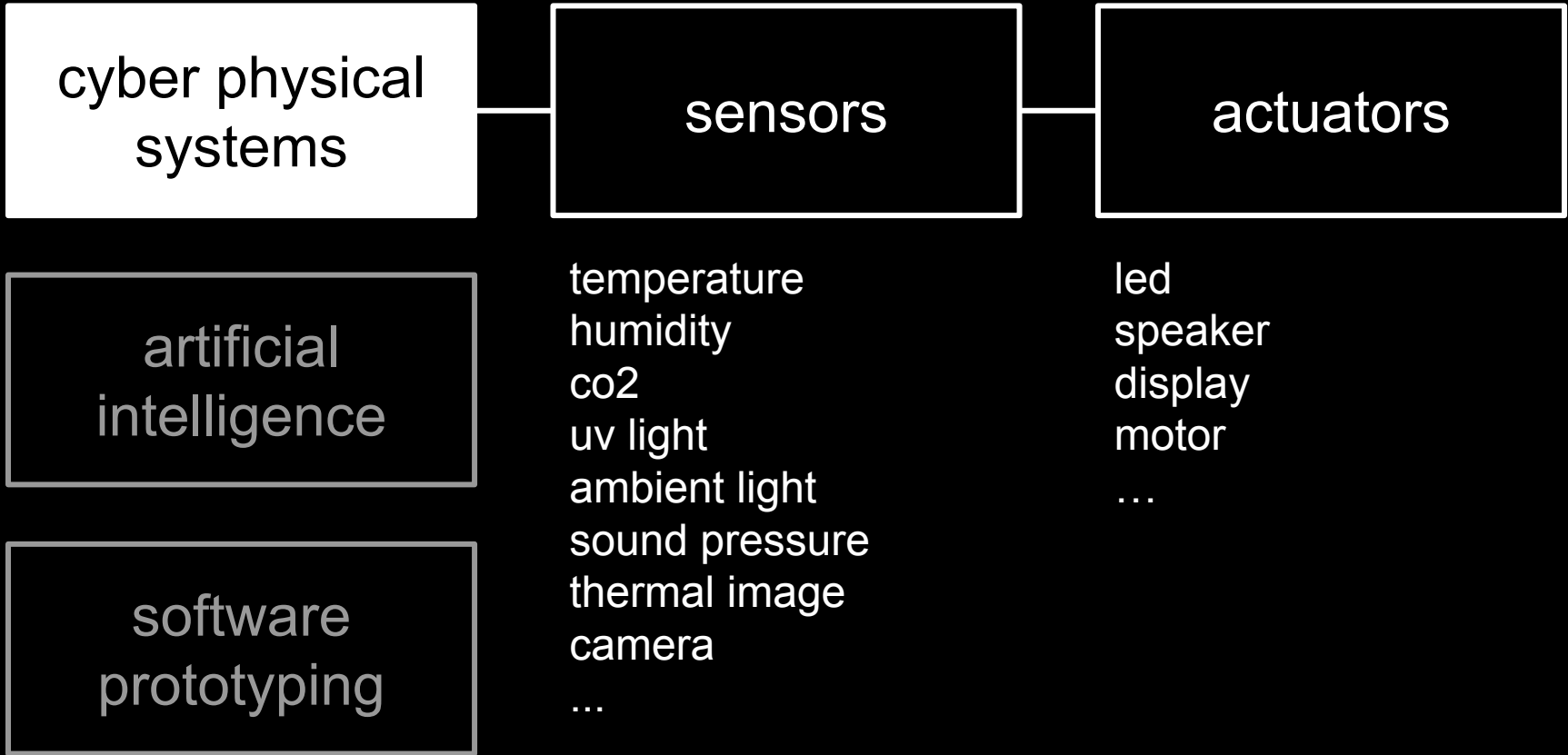
# a model for solving problems

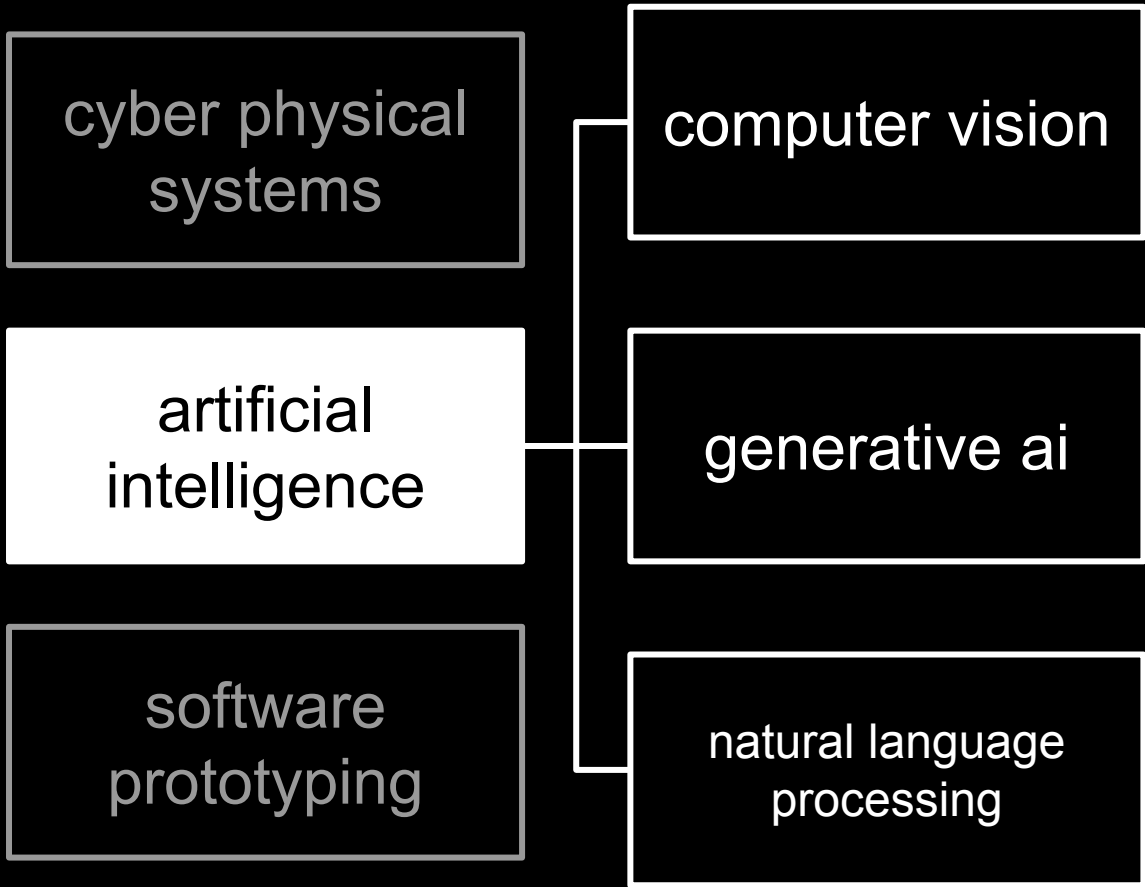


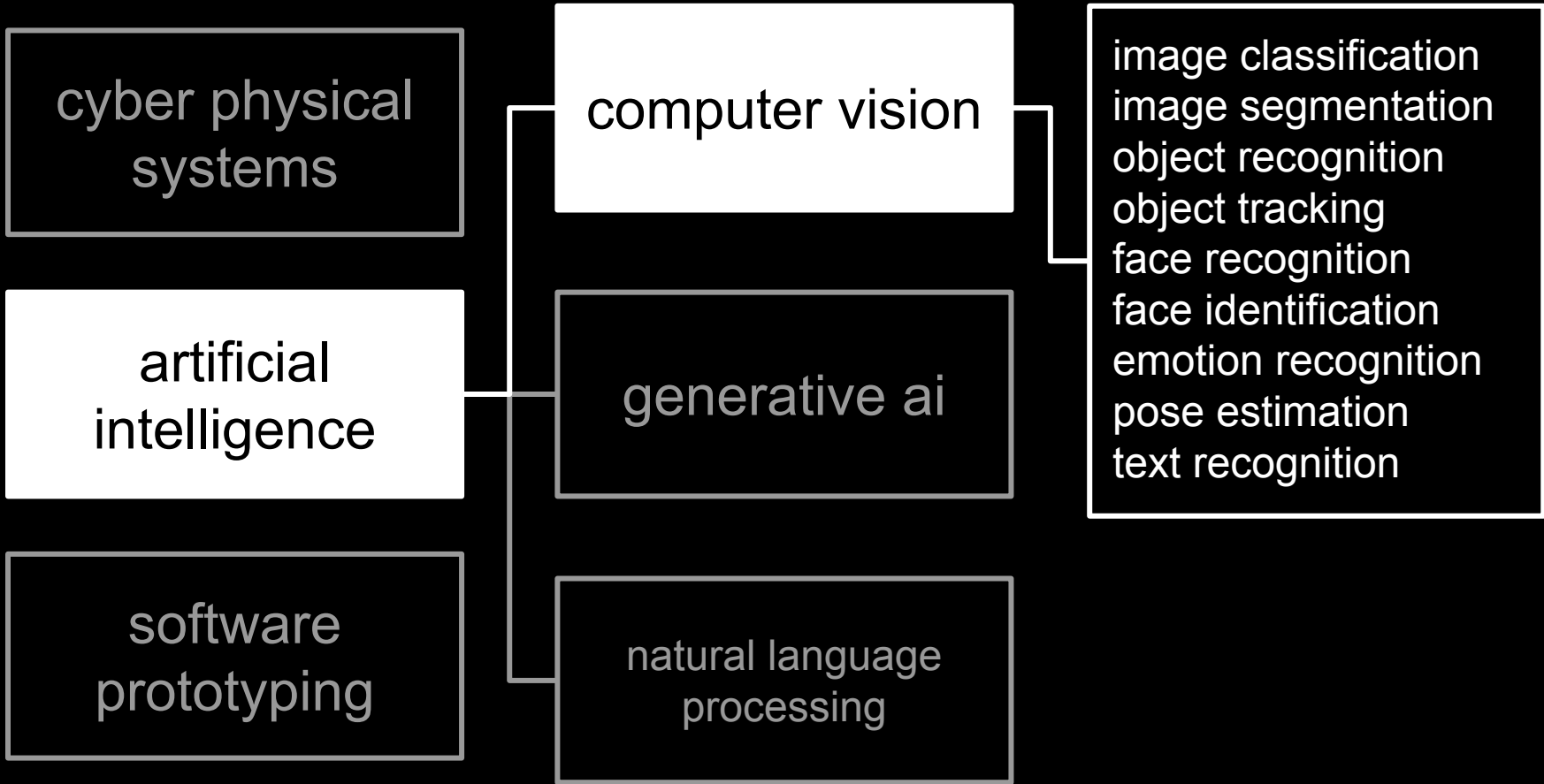
cyber physical  
systems

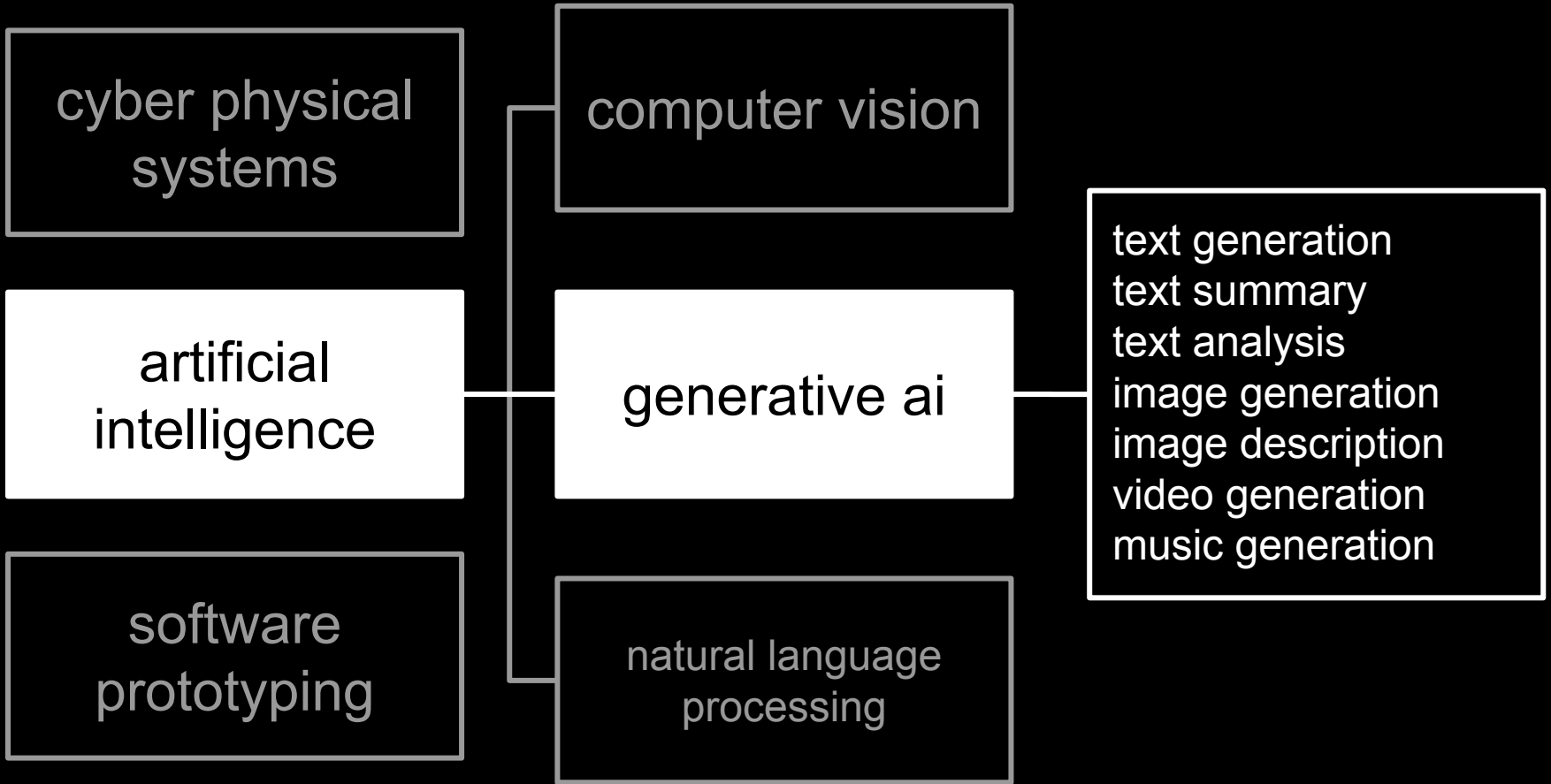
artificial  
intelligence

software  
prototyping

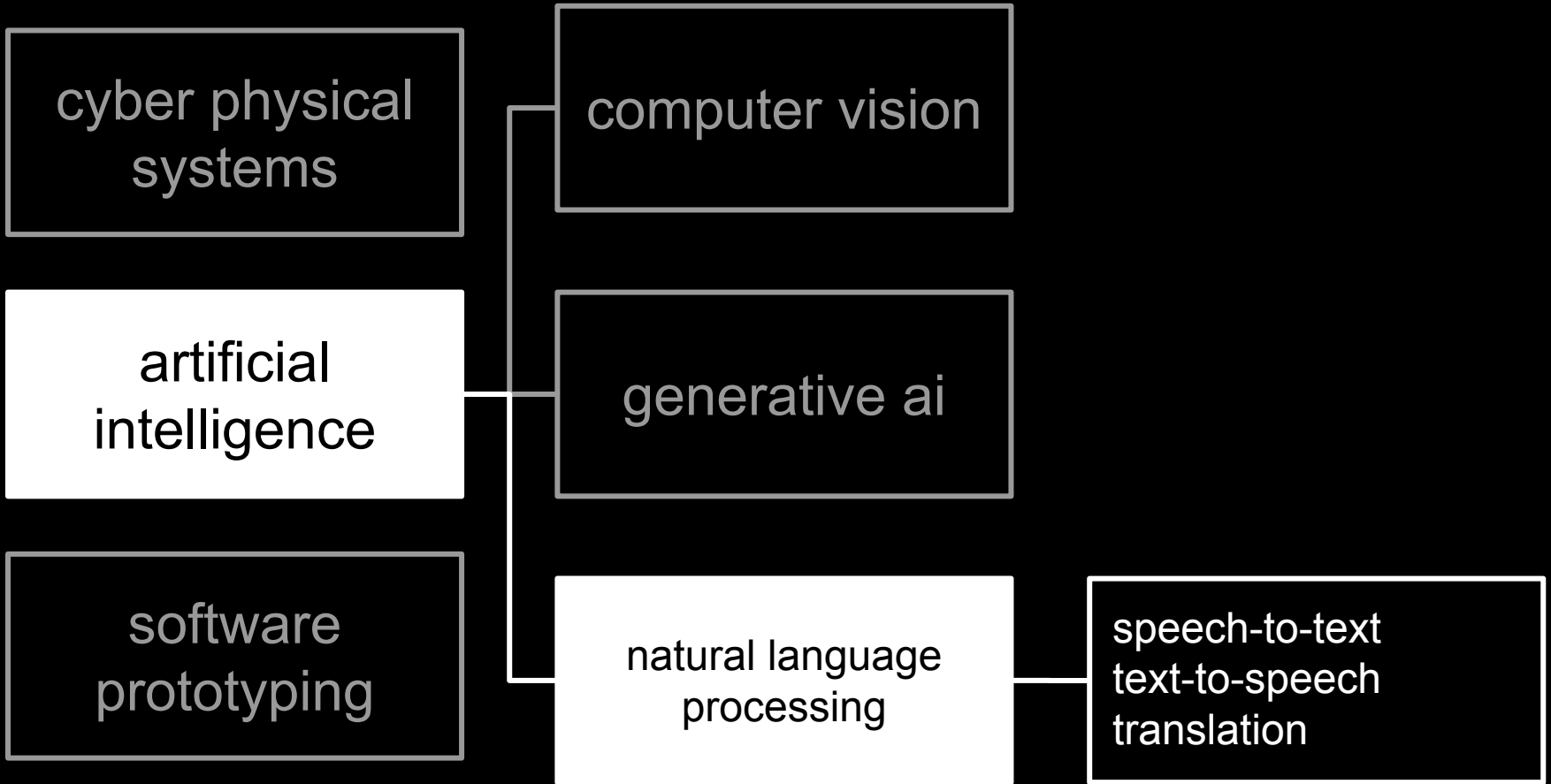


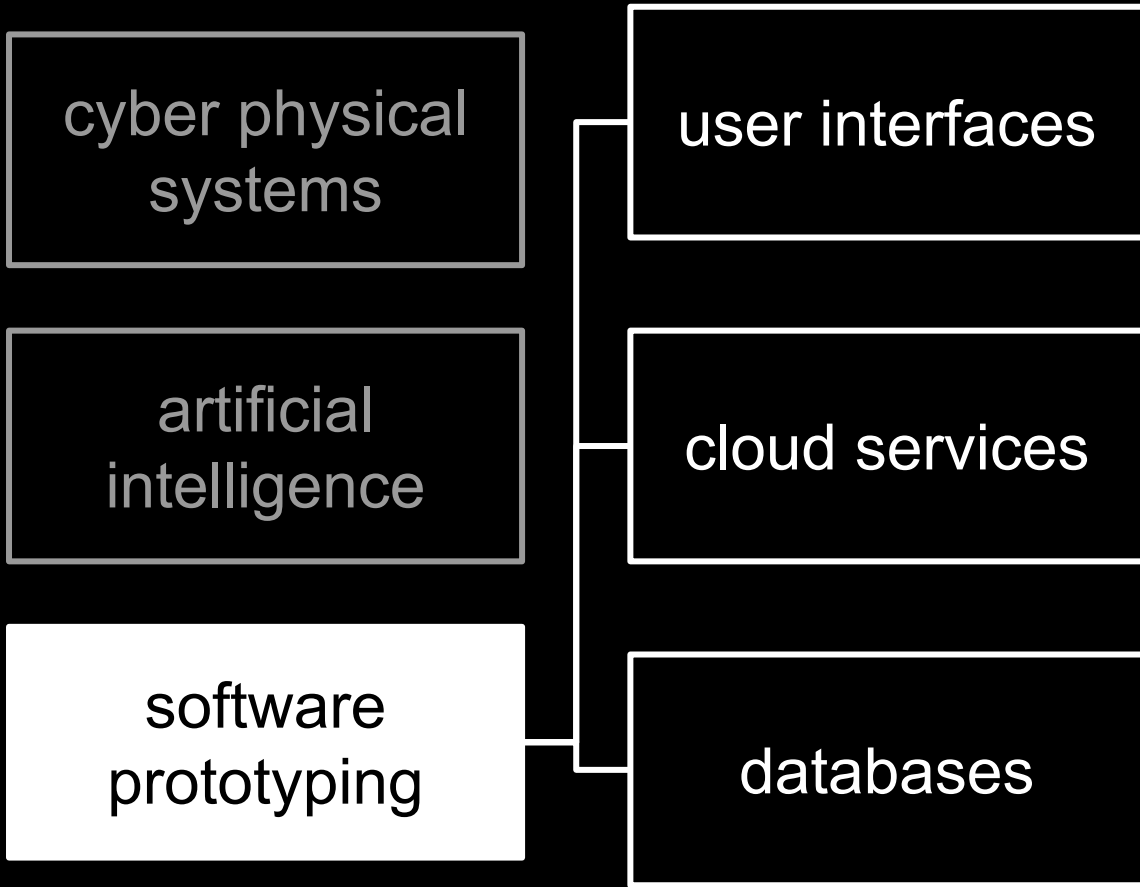












introductory example

visual studio code  
programs  
python

LEDs

# large language models

speech-to-text

user interface



# SENSORS

temperature / humidity

rgb led button

camera

thermal imaging camera

microphone

keyboard

temperature / humidity

```
th = BrickletHumidityV2(UID, ipcon)...
```

```
th.get_humidity()
```

```
th.get_temperature()
```

```
th.register_callback(th.CALLBACK_HUMIDITY, cb_humidity)  
th.register_callback(th.CALLBACK_TEMPERATURE, ...)
```

```
th.set_humidity_callback_configuration(250, False, "x", 0, 0)  
th.set_temperature_callback_configuration(...)
```

rgb led button



```
btn = BrickletRGBLEDButton(UID, ipcon)...
```

```
btn.set_color(255, 0, 0)
```

```
btn.get_button_state()
```

```
btn.register_callback(...)
```

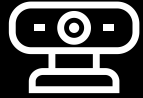
camera

# OpenCV

```
import cv2
```



```
# Get video capture device (webcam)  
webcam = cv2.VideoCapture(0)
```



```
# Read a frame
```

```
success, frame = webcam.read()
```





```
# Show the image from the frame  
cv2.imshow("Webcam", frame)
```



```
# Save the frame as .png
```

```
cv2.imwrite("screenshot.png", frame)
```

thermal imaging camera

OpenCV  
Tinkerforge



```
ti = BrickletThermalImaging(UID, ipcon)
ti.set_image_transfer_config(...)
img = ti.get_high_contrast_image()
```

```
ti.register_callback(...)
```

microphone

```
import pyaudio
```



```
# Define recording parameters  
FORMAT = pyaudio.paInt16  
CHANNELS = 1  
RATE = 44100  
CHUNK = 1024
```

```
# Get access to the microphone  
audio = pyaudio.PyAudio()
```

```
# Start listening  
stream = audio.open(...)
```

```
# Read a chunk of frames  
stream.read(CHUNK)
```

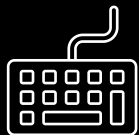
```
# Stop and close stream  
stream.stop_stream()  
stream.close()
```

```
# Terminate access to microphone  
audio.terminate()
```

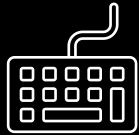
keyboard

```
import keyboard
```



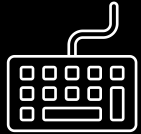


```
# Define a callback function for a key  
def record_audio():  
    print("Recording audio...")
```



```
# Add key listener
```

```
keyboard.add_hotkey("r", record_audio)
```



```
# Wait until a specific key was pressed  
keyboard.wait("esc")
```

# ACTUATORS

rgb led  
OLED display  
speaker

rgb led

```
led = BrickletRGBLEDV2(UID, ipcon)
led.set_rgb_value(255, 0, 0)
```

# OLED display



```
oled = BrickletOLED128x64V2(UID, ipcon)
oled.clear_display()
oled.write_line(0, 0, "Welcome!")
```

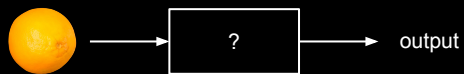
speaker

```
import simpleaudio as sa
```



```
# Create a wave object from .wav-file and play it  
wav = sa.WaveObject.from_wave_file("sound.wav")  
wav.play().wait_done()
```

# COMPUTER VISION



finding oranges in images

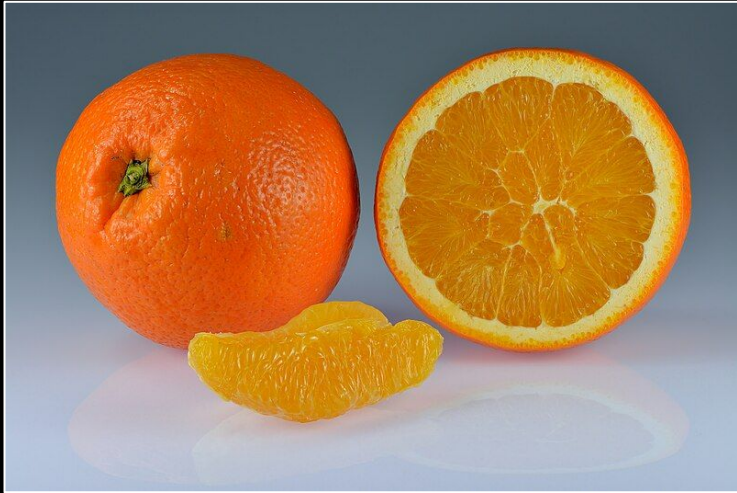


Image source: [Wikimedia](#)

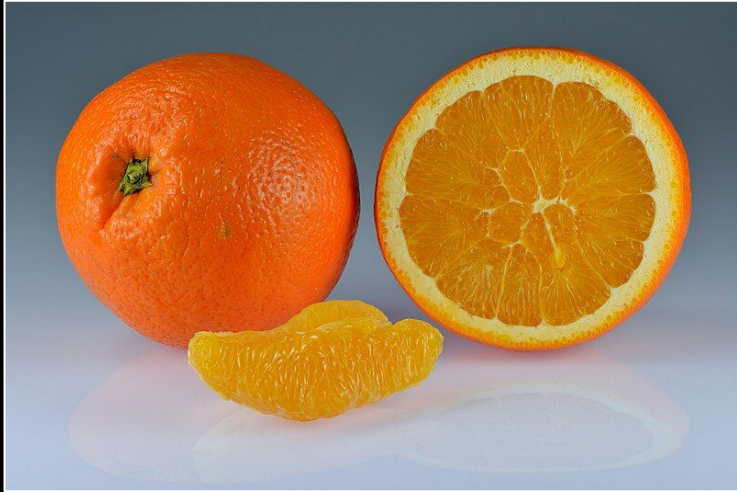


Image source: [Wikimedia](#)

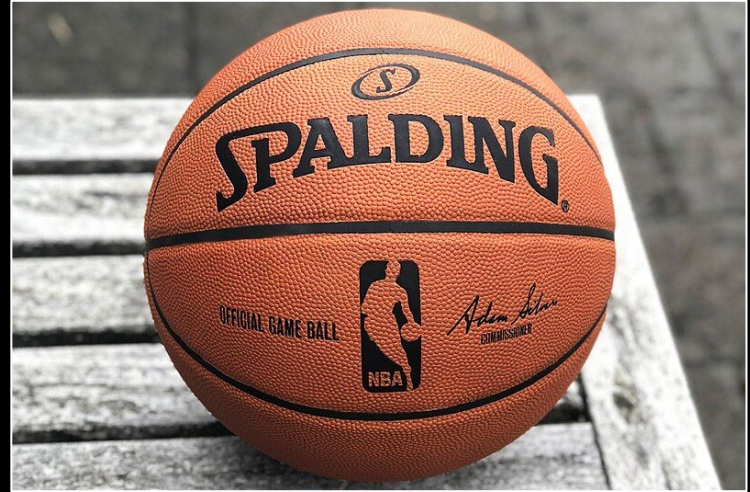
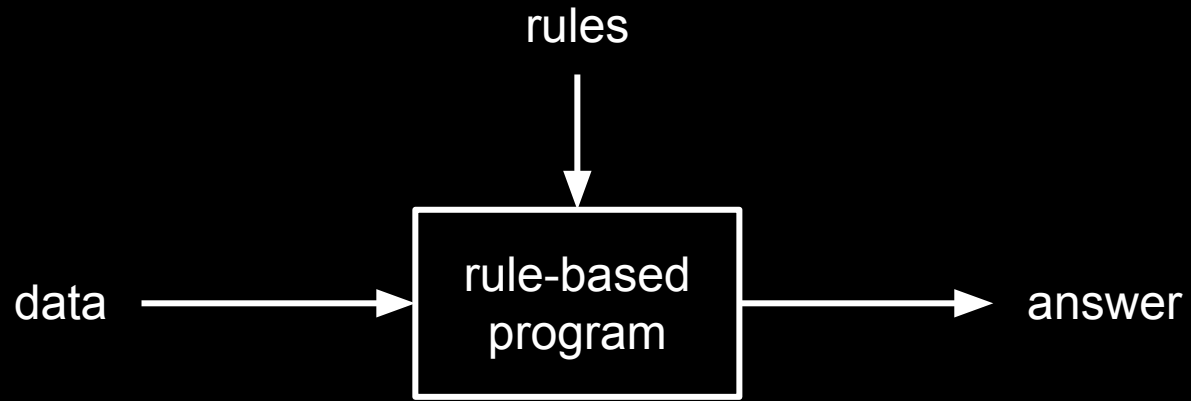


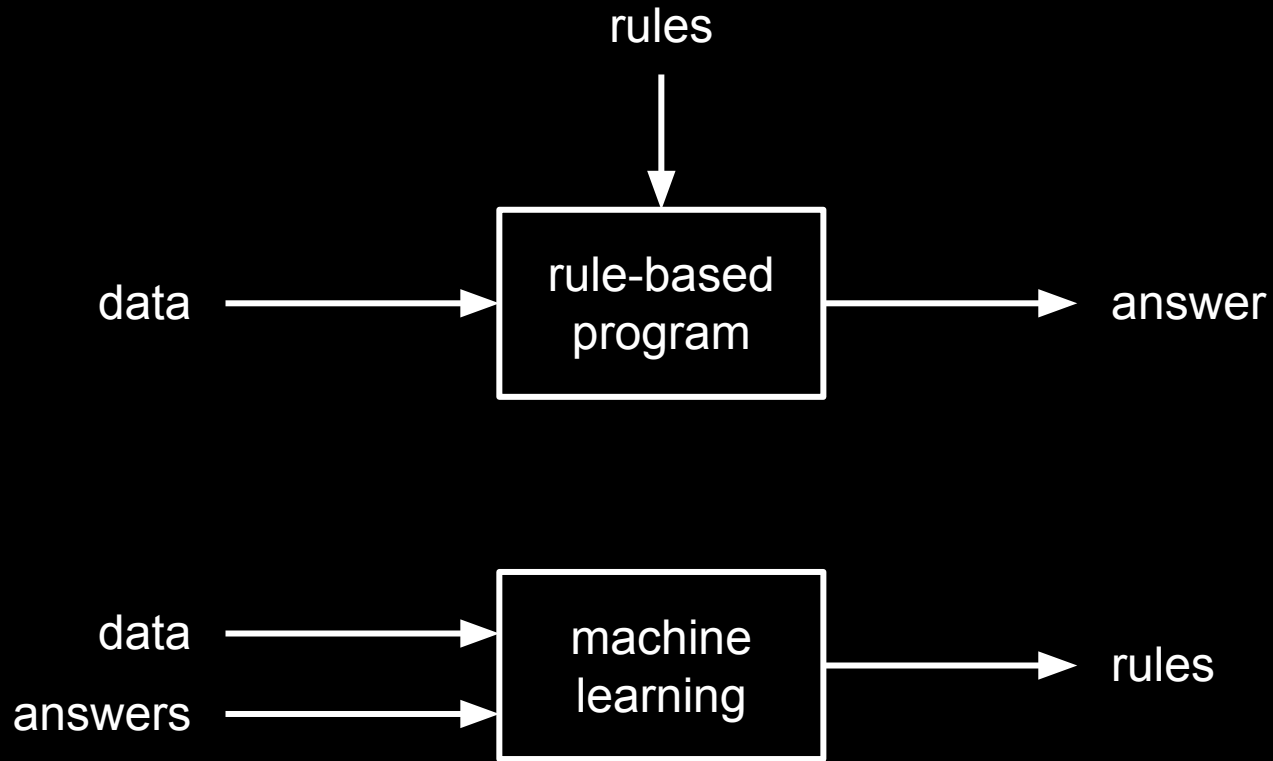
Image source: [Wikimedia](#)



what set of rules can solve this?

# machine learning algorithms



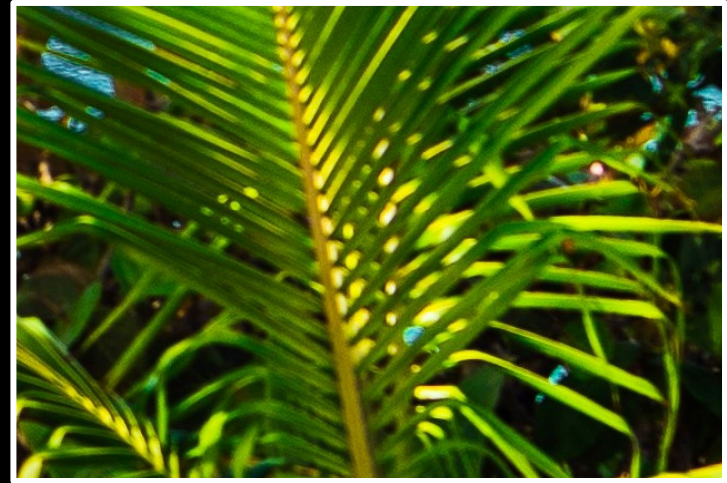


images in a computer

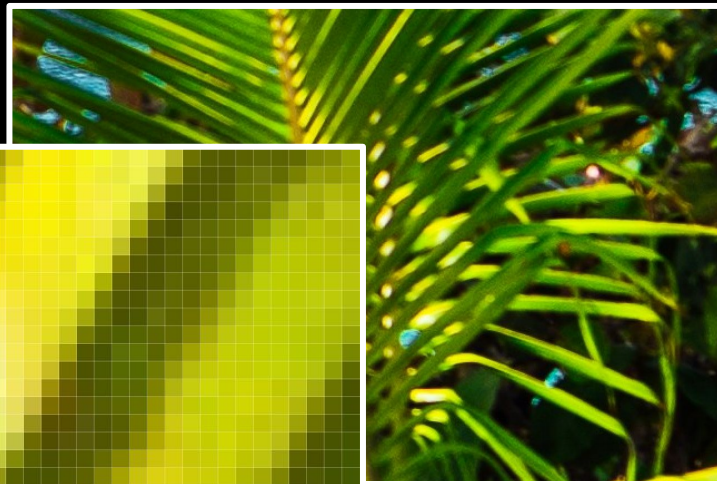
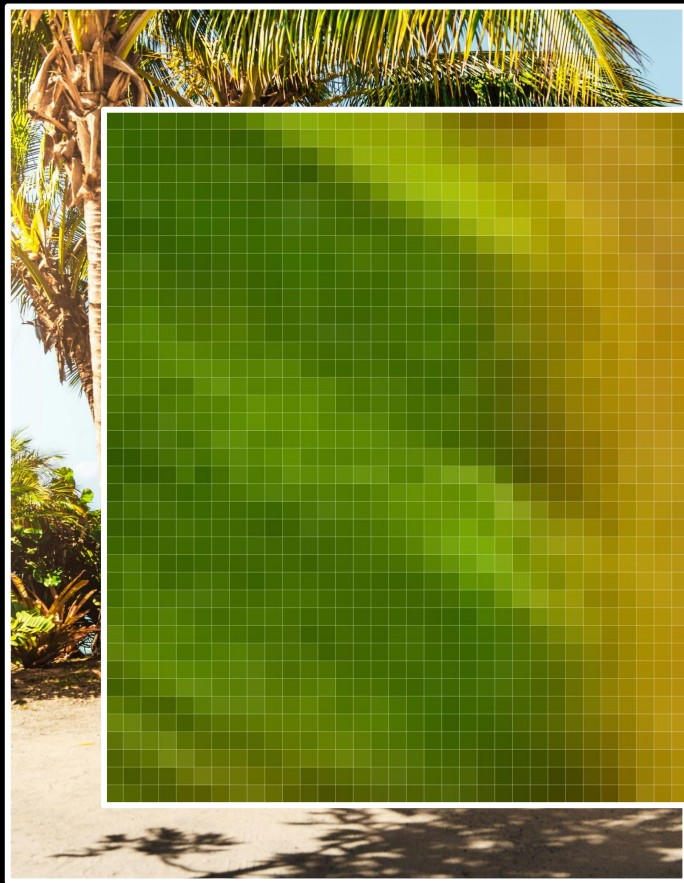


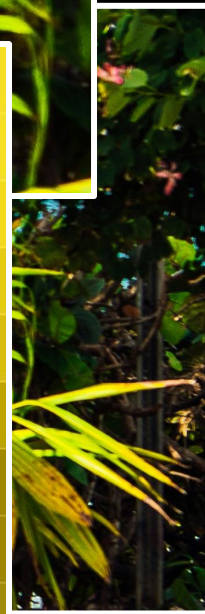
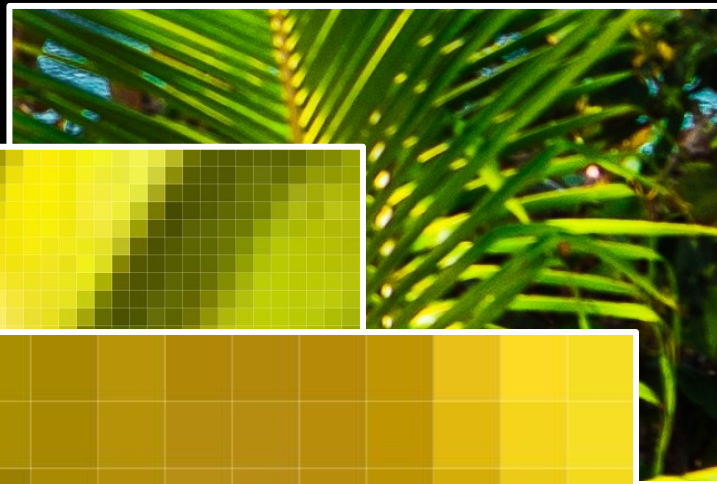
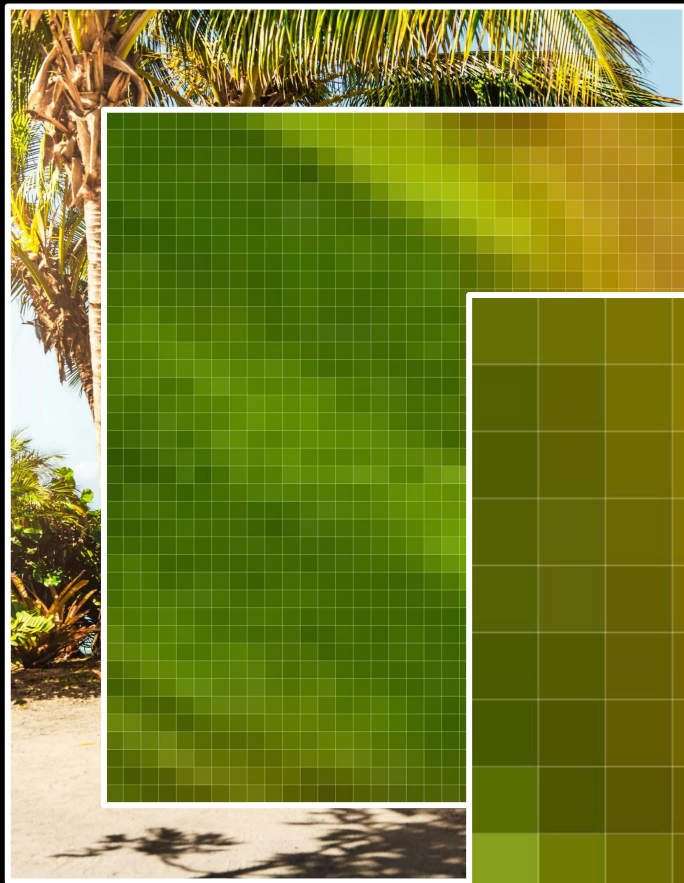


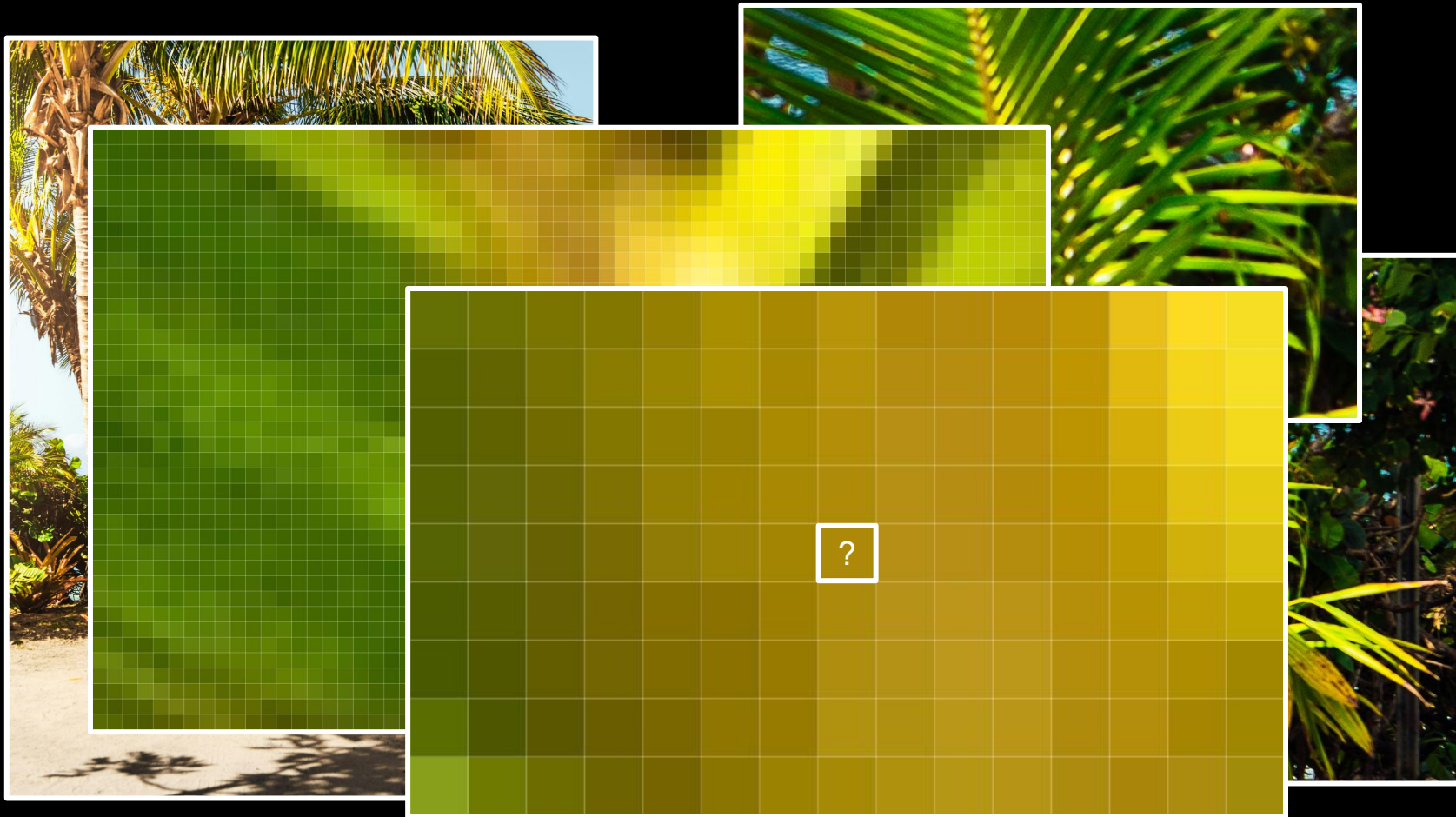




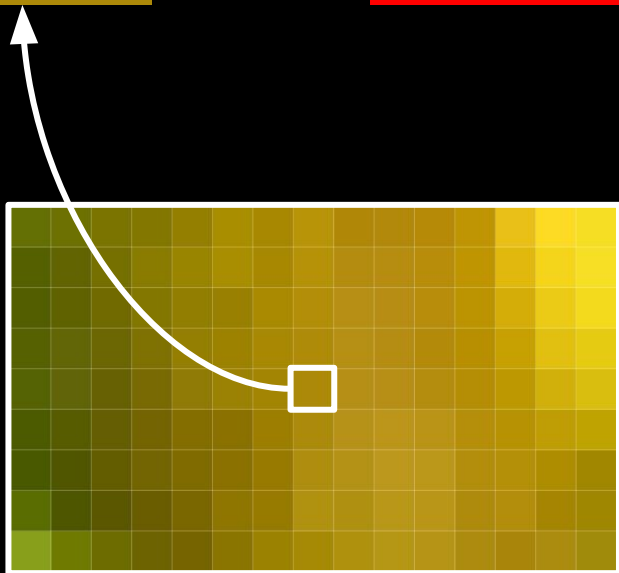






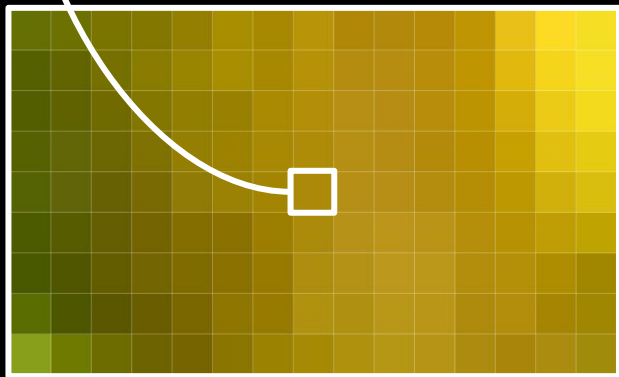


$$\text{[Olive Box]} = \text{[Red Box R]} + \text{[Green Box G]} + \text{[Blue Box B]}$$



$$\text{[Olive Box]} = \text{[Red Box R]} + \text{[Green Box G]} + \text{[Blue Box B]}$$

172                      137                      9



# image classification



Q: Does an image belong to one or the other class from a fixed set of classes?

Cat or Dog?



model



"cat"



# Cat or Dog?



model

"cat"



model

"dog"

# Google's teachable machine

<https://teachablemachine.withgoogle.com>

```
pip install keras
```

```
pip install tensorflow==2.12.0
```

```
# Load the classifier and class names
model = load_model("my_model.h5")
class_names = open("labels.txt", "r").readlines()
```

```
# Convert the image to 224 x 224
image = cv2.resize(image, (224, 224), interpolation=cv2.INTER_AREA)

# Turn into a list of pixels
image = np.asarray(image, dtype=np.float32).reshape(1, 224, 224, 3)

# Normalize each pixel's color value (-1/1)
image = (image / 127.5) - 1
```

```
# Make a prediction for the class
prediction = model.predict(image)

# Get the class with the highest confidence value
index = np.argmax(prediction)
class_name = class_names[index]

# Get the confidence score for the predicted class
confidence_score = prediction[0][index]
```

Class: Labeler  
Confidence: 1.00



# YOLO v8 Image Classification

<https://docs.ultralytics.com/>



```
pip install ultralytics
```

```
# Load the classifier
from ultralytics import YOLO
model = YOLO("yolov8n-cls.pt")
```

```
# Make a prediction  
results = model('cat.jpg')
```

```
# Show result  
results[0].show()
```



```
# Get the top result
top = results[0].probs.top1
class_name = results[0].names[top]
print(class_name)
```

# zero-shot image classification

Q: Which classes do you train your model on?

# GPT-4 Vision



```
pip install openai
```

```
# import openai API and set api key
from openai import OpenAI
os.environ["OPENAI_API_KEY"] = "..."
client = OpenAI()
```

```
# define a suitable prompt for the task
prompt = "Classify the image into 'dog' or 'cat'. Return
only the word for the class of the image."
```

```
# This function is needed to encode an image to base64 for OpenAI's API
def encode_image(image_path):
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode('utf-8')

image_path = "cat.webp"
image = encode_image(image_path)
```

```
response = client.chat.completions.create(
    model="gpt-4-turbo",
    messages = [
        { "role": "user", "content": [
            { "type": "text", "text": prompt },
            { "type": "image_url", "image_url": { "url": f"data:image/jpeg;base64,{image}" } }
        ]
    },
    max_tokens=300,
)
```

```
# Show the answer of the classification
print(response.choices[0].message.content)
```

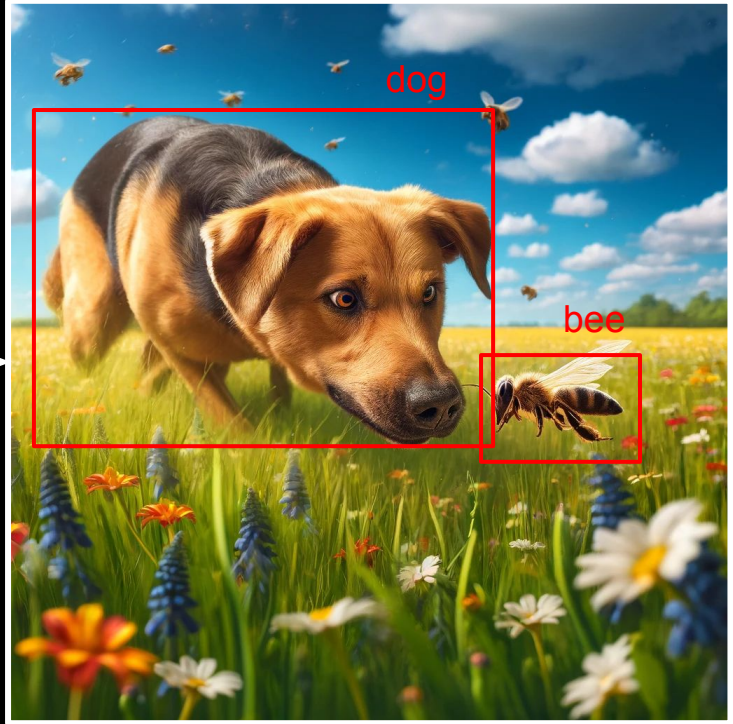
object detection

Q: Which objects are in the image and where?



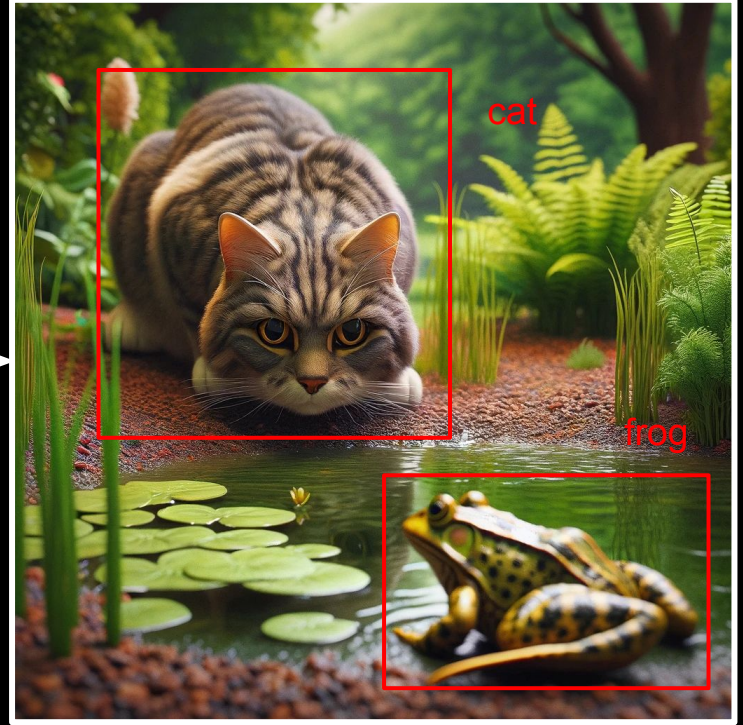


AI





AI



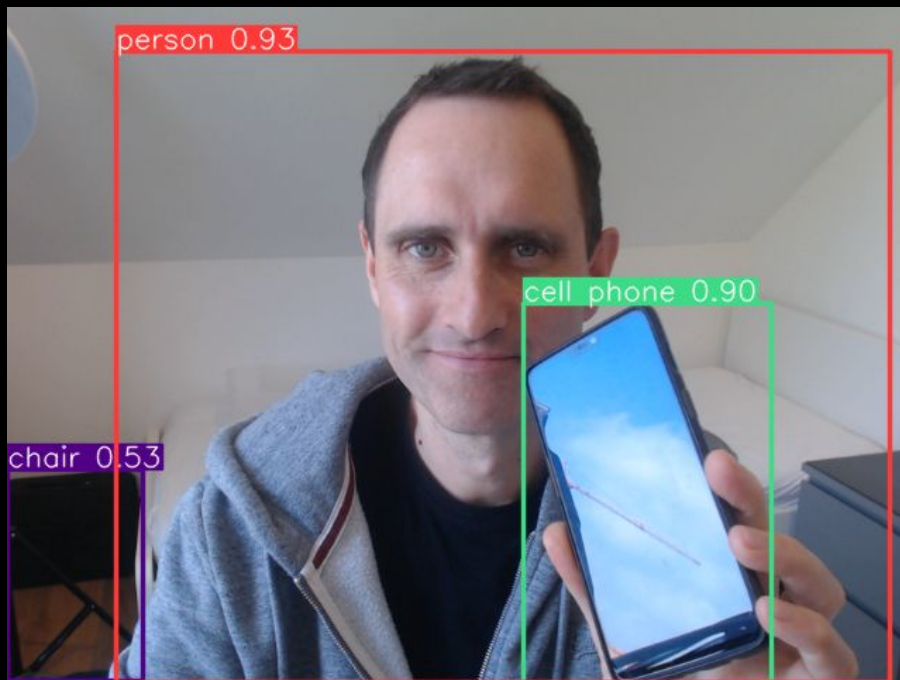
# YOLO v8 Object Detection

<https://docs.ultralytics.com/>

```
# Load the detector
from ultralytics import YOLO
model = YOLO("yolov8n.pt")
```

```
# Make a prediction one each frame
results = model(frame)

# Annotate frame
annotated_frame = results[0].plot()
```



Q: Which objects do you teach your model to recognize?

# zero-shot object detection



# "Simple Open-Vocabulary Object Detection with Vision Transformers"

<https://arxiv.org/abs/2205.06230>

```
# Load the open world detector
from ultralytics import YOLO
model = YOLO("yolov8s-world.pt")
```

```
# Define custom objects to look for  
model.set_classes(["person with glasses"])
```

```
# Make a prediction one each frame
results = model(frame)

# Annotate frame
annotated_frame = results[0].plot()
```

# optical character recognition (OCR)



tesseract

# GPT-4 Vision

```
# define a suitable prompt for the task
prompt = "Extract all food and beverage items with their
quantity and price from this receipt into a JSON list. The
receipt is in German."
```



```
response = client.chat.completions.create(
    model="gpt-4o",
    response_format={ "type": "json_object" },
    messages = [
        { "role": "user", "content": [
            { "type": "text", "text": prompt },
            { "type": "image_url", "image_url": { "url": f"data:image/jpeg;base64,{image}" } }
        ]
    }
],
    max_tokens=300,
)
```

# GENERATIVE AI

# LARGE LANGUAGE MODELS

what has been said so far?  
(*history + prompt*)



prediction of next token based on  
learnt probability distribution

what has been said so far?  
(*history + prompt*)



prediction of next token based on  
learnt probability distribution

+

(randomness)

what has been said so far?  
(*history + prompt*)



prediction of next token based on  
learnt probability distribution

+

(randomness)

+

(filter)

(*discriminating, insulting content*)

what has been said so far?  
(*history + prompt*)



prediction of next token based on  
learnt probability distribution

+

(randomness)

+

(filter)

(*discriminating, insulting content*)

next word (*token*)



what has been said so far?  
(*history + prompt*)



prediction of next token based on  
learnt probability distribution

+

(randomness)

+

(filter)

(*discriminating, insulting content*)



next word (*token*)





# PROMPTING

<https://www.promptingguide.ai/>



elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

example prompt

Explain the binary number system.

## elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

## example prompt

Explain the binary number system.

start simple

## elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

## example prompt

You are a friendly tutor and your task is to explain complex concepts as simple as possible.

Explain the binary number system.

## elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

## example prompt

You are a friendly tutor and your task is to explain complex concepts as simple as possible.

Your answers are never longer than 10 sentence.

Explain the binary number system.

# ZERO-SHOT PROMPTING



## elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

## example prompt

Classify the text into neutral,  
negative or positive.

Text: "What a great dinner!"

Sentiment:

## elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

## example prompt

Classify the text into neutral,  
negative or positive.

Text: "What a great dinner!"

Sentiment:

this will be replaced with  
data later...

# FEW-SHOT PROMPTING

IN-CONTEXT LEARNING

## examples in the context to learn from

Extract all references to countries and their continent in the following text using the format from the examples below.

Example 1: "They played the team called 'Die Mannschaft' in the world cup final"

Correct answer: Germany, Europe

Example 2: "The Three Lions once again lost to Germany in a semi final"

Correct answer: England, Europe, Germany, Europe

Text: "The Selecao was destroyed 1:7 by the DFB selection in their home stadium."

Answer:

## examples in the context to learn from

Extract all references to countries and their continent in the following text using the format from the examples below.

Example 1: "They played the team called 'Die Mannschaft' in the world cup final"

Correct answer: Germany, Europe

Example 2: "The Three Lions once again lost to Germany in a semi final"

Correct answer: England, Europe, Germany, Europe

Text: "The Selecao was destroyed 1:7 by the DFB selection in their home stadium."

Answer:

more prompting strategies

chain-of-thought (CoT)

self-consistency

generate knowledge prompting

prompt chaining (subtasks)

tree-of-thoughts (ToT)

retrieval-augmented-generation (RAG)

...

OpenAI 

```
pip install openai
```



```
from openai import OpenAI
import os
```

```
os.environ["OPENAI_API_KEY"] = "<YOUR_API_KEY>"
client = OpenAI()
```

```
# define a system message
system_message = """
    You are a world-famous 5-star chef. Based on ingredients the user has at home,
    you suggest easy-to-cook recipes. """
```

```
# define a prompt for the task
prompt = """
    Suggest a recipe for lunch.

    List of ingredients:
    - butter
    - eggs
    - flour
    - salt
    - milk

    Recipe: """
```

```
response = client.chat.completions.create(  
    model="gpt-4o",  
    messages = [  
        {"role": "system", "content": system_message },  
        {"role": "user", "content": prompt },]  
    ],  
    max_tokens=2000  
)
```

# USER INTERFACES

# streamlit

<https://docs.streamlit.io/> # official documentation  
<https://streamlit.io/components> # third-party extensions

```
pip install streamlit
```

- Home.py
- pages/
  - 1\_Speech.py
  - 2\_Webcam.py
  - 3\_Microphone.py
- lib/
  - speech\_to\_text.py
  - text\_to\_speech.py
  - vision.py



entry point to our UI

- Home.py
- pages/
  - 1\_Speech.py
  - 2\_Webcam.py
  - 3\_Microphone.py
- lib/
  - speech\_to\_text.py
  - text\_to\_speech.py
  - vision.py

entry point to our UI

- Home.py

- pages/

- 1\_Speech.py

- 2\_Webcam.py

- 3\_Microphone.py

- lib/

- speech\_to\_text.py

- text\_to\_speech.py

- vision.py

more pages in our app

entry point to our UI

- Home.py

- pages/

- 1\_Speech.py

- 2\_Webcam.py

- 3\_Microphone.py

more pages in our app

- lib/

- speech\_to\_text.py

- text\_to\_speech.py

- vision.py

our custom functions we'd like to use from our UI

## Home.py

```
import streamlit as st

st.title("My first UI")
st.write("This is a simple UI for prototyping our application.")

name = st.text_input("Enter your name")

if st.button("Greet me"):
    st.write(f"Hello {name} 🙌")
```

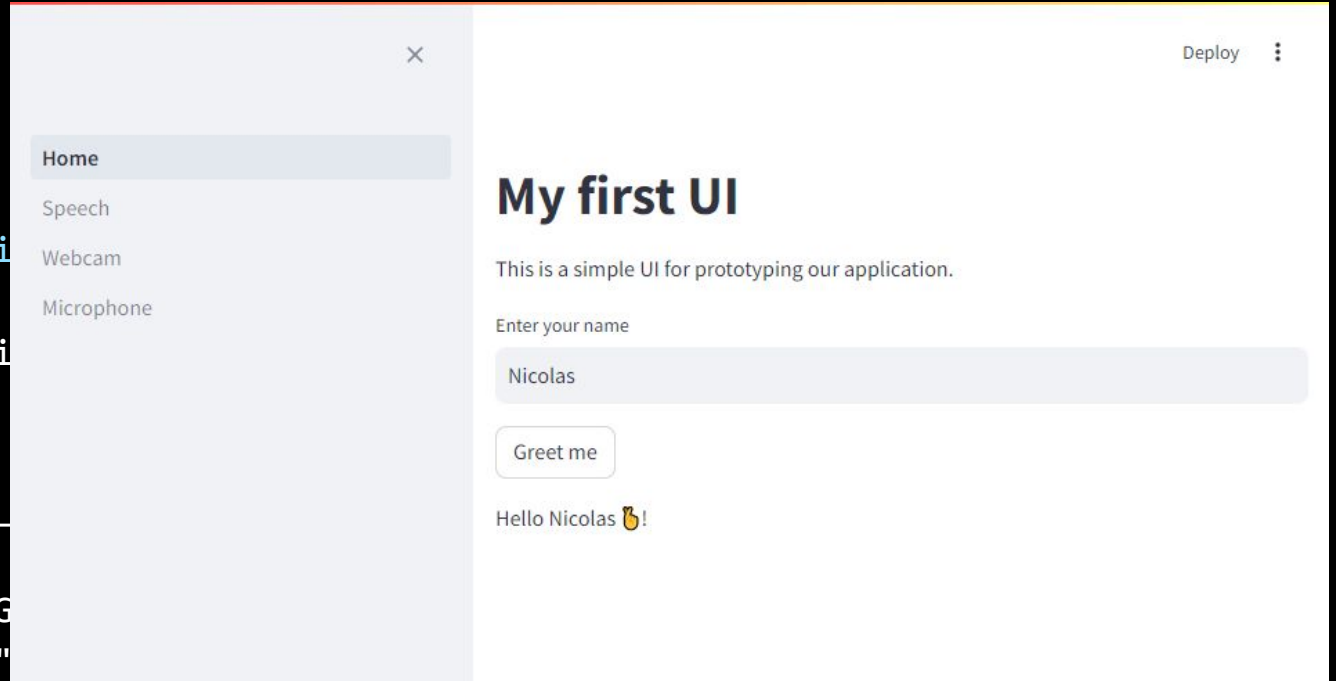
# Home.py

```
import streamlit as st

st.title("My first UI")
st.write("This is a simple UI for prototyping our application.")

name = st.text_input("Enter your name")

if st.button("Greet me"):
    st.write(f"Hello {name} 🎉!")
```



## 1\_Speech.py

```
import streamlit as st
from pages.lib.text_to_speech import text_to_speech

st.title("Speech demo")
st.write("Enter a text and it will be converted to speech.")

text = st.text_input("Enter some text")
voice = st.selectbox("Select a voice", ["alloy", ... "shimmer"])

if st.button("Turn to speech"):
    audio_file = text_to_speech(text, voice=voice)
    st.audio(audio_file.as_posix(), format="audio/mpeg")
```

## 1\_Speech.py

```
import streamlit as st
from pages.lib.text_to_speech import text_to_speech from lib/text_to_speech.py

st.title("Speech demo")
st.write("Enter a text and it will be converted to speech.")

text = st.text_input("Enter some text")
voice = st.selectbox("Select a voice", ["alloy", ... "shimmer"])

if st.button("Turn to speech"):
    audio_file = text_to_speech(text, voice=voice)
    st.audio(audio_file.as_posix(), format="audio/mpeg")
```

## lib/text\_to\_speech.py

```
from openai import OpenAI
import os
os.environ["OPENAI_API_KEY"] = "..."
client = OpenAI()

def text_to_speech(text, voice="alloy"):
    speech_file_path = Path(__file__).parent / "speech.mp3"
    response = client.audio.speech.create(
        model="tts-1",
        voice=voice,
        input=text
    )

    response.write_to_file(speech_file_path)
    return speech_file_path
```



## lib/text\_to\_speech.py

```
from openai import OpenAI
import os
os.environ["OPENAI_API_KEY"] = "..."
client = OpenAI()
```

setup OpenAI  
API

```
def text_to_speech(text, voice="alloy"):
    speech_file_path = Path(__file__).parent / "speech.mp3"
    response = client.audio.speech.create(
        model="tts-1",
        voice=voice,
        input=text
    )

    response.write_to_file(speech_file_path)
    return speech_file_path
```

## lib/text\_to\_speech.py

```
from openai import OpenAI
import os
os.environ["OPENAI_API_KEY"] = "..."
client = OpenAI()
```

setup OpenAI  
API

```
def text_to_speech(text, voice="alloy"):
    speech_file_path = Path(__file__).parent / "speech.mp3"
    response = client.audio.speech.create(
        model="tts-1",
        voice=voice,
        input=text
    )

    response.write_to_file(speech_file_path)
    return speech_file_path
```

define  
custom  
function

## 2\_Webcam.py

```
import streamlit as st
from pages.lib.vision import ask_gpt4o

st.title("Video camera test")

picture = st.camera_input("Take a picture")

if picture:
    st.image(picture)
    answer = ask_gpt4o("What is in this picture?", picture)
    st.write(answer)
```

## 2\_Webcam.py

```
import streamlit as st
from pages.lib.vision import ask_gpt4o from lib/vision.py

st.title("Video camera test")

picture = st.camera_input("Take a picture")

if picture:
    st.image(picture)
    answer = ask_gpt4o("What is in this picture?", picture)
    st.write(answer)
```

## lib/vision.py

```
from openai import OpenAI
import os
os.environ["OPENAI_API_KEY"] = "..."
client = OpenAI()

def encode_image(image_buffer):

def ask_gpt4o(prompt, image_buffer):
    image = encode_image(image_buffer)
    response = client.chat.completions.create(
        model="gpt-4o",
        messages=[
            {
                "role": "user", "content": [
                    { "type": "text", "text": prompt },
                    { "type": "image_url", "image_url": { "url": f"data:image/jpeg;base64,{image}" } }
                ],
            }
        ]
    )

    return response.choices[0].message.content
```

## lib/vision.py

```
from openai import OpenAI
import os
os.environ["OPENAI_API_KEY"] = "..."
client = OpenAI()
```

```
def encode_image(image_buffer):
```

```
def ask_gpt4o(prompt, image_buffer):
    image = encode_image(image_buffer)
    response = client.chat.completions.create(
        model="gpt-4o",
        messages=[
            {
                "role": "user", "content": [
                    { "type": "text", "text": prompt },
                    { "type": "image_url", "image_url": { "url": f"data:image/jpeg;base64,{image}" } }
                ],
            }
        ]
    )

    return response.choices[0].message.content
```

setup OpenAI  
API

define  
custom  
function

### 3\_Microphone.py

```
import streamlit as st
from streamlit_mic_recorder import mic_recorder
from pages.lib.text_to_speech import speech_to_text

st.title("Microphone test")

def callback():
    if st.session_state.my_recorder_output:
        audio = st.session_state.my_recorder_output
        text = text_to_speech(audio)
        st.success(text)

audio = mic_recorder(key='my_recorder', callback=callback)
```

## 3\_Microphone.py

pip install streamlit-mic-recorder

```
import streamlit as st
from streamlit_mic_recorder import mic_recorder
from pages.lib.text_to_speech import speech_to_text

st.title("Microphone test")

def callback():
    if st.session_state.my_recorder_output:
        audio = st.session_state.my_recorder_output
        text = text_to_speech(audio)
        st.success(text)

audio = mic_recorder(key='my_recorder', callback=callback)
```



## 3\_Microphone.py

pip install streamlit-mic-recorder

```
import streamlit as st
from streamlit_mic_recorder import mic_recorder
from pages.lib.text_to_speech import speech_to_text from lib/speech_to_text.py

st.title("Microphone test")

def callback():
    if st.session_state.my_recorder_output:
        audio = st.session_state.my_recorder_output
        text = text_to_speech(audio)
        st.success(text)

audio = mic_recorder(key='my_recorder', callback=callback)
```

## lib/speech\_to\_text.py

```
from openai import OpenAI
import os
import io

os.environ["OPENAI_API_KEY"] = "..."
client = OpenAI()

def speech_to_text(audio):
    audio_bio = io.BytesIO(audio['bytes'])
    audio_bio.name = 'audio.mp3'

    transcription = client.audio.transcriptions.create(
        model="whisper-1",
        file=audio_bio
    )
    return transcription.text
```

## lib/speech\_to\_text.py

```
from openai import OpenAI
import os
import io

os.environ["OPENAI_API_KEY"] = "..."
client = OpenAI()
```

setup OpenAI  
API

```
def speech_to_text(audio):
    audio_bio = io.BytesIO(audio['bytes'])
    audio_bio.name = 'audio.mp3'

    transcription = client.audio.transcriptions.create(
        model="whisper-1",
        file=audio_bio
    )
    return transcription.text
```

define  
custom  
function