```
from openai import OpenAI
import base64
from loguru import logger
from typing import Any, Optional



class VisionAPIWrapper:
    def __init__(
        self,
        api_key: str = None,
        system_prompt: str = None,
        model: str = "gpt-4o-mini",
        max_tokens: int = 300,
        temperature: float = 0.7,
    ):
        """
        Initializes the API wrapper with the system prompt and configuration.

        Args:
            system_prompt (str): The system prompt for the model.
            model (str): The OpenAI model to use.
            max_tokens (int): Maximum number of tokens to generate.
            temperature (float): Sampling temperature for the model.
        """
        self.client = OpenAI(api_key=api_key)
        self.system_prompt = system_prompt
```

```
    self.model = model
    self.max_tokens = max_tokens
    self.temperature = temperature

@staticmethod
def encode_image(image_path: str) -> str:
    """
    Encodes the image to base64 format.

    Args:
        image_path (str): Path to the image file.

    Returns:
        str: Base64 encoded image string.
    """
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode("utf-8")

# @retry(stop=stop_after_attempt(3), wait=wait_fixed(2))
def run(self, task: str, img: Optional[str] = None) -> Any:
    """
    Sends a request to the OpenAI API with a task and an optional image.

    Args:
        task (str): Task to send to the model.
        img (Optional[str]): Path to the image to be analyzed by the model (optional).
```

```
Returns:
    Any: The response from the model.
"""
messages = [{"role": "system", "content": self.system_prompt}]

user_message = {
    "role": "user",
    "content": [{"type": "text", "text": task}],
}

if img:
    base64_image = self.encode_image(img)
    image_message = {
        "type": "image_url",
        "image_url": {
            "url": f"{base64_image}"
        },
    }
    user_message["content"].append(image_message)

messages.append(user_message)

logger.info(
    f"Sending request to OpenAI with task: {task} and image: {img}"
)
```

```
    try:
        response = self.client.chat.completions.create(
            model=self.model,
            messages=messages,
            max_tokens=self.max_tokens,
            temperature=self.temperature,
        )
        logger.info("Received response successfully.")
        return response.choices[0].message.content
    except Exception as e:
        logger.error(
            f"An error occurred while making the API request: {e}"
        )
        raise


def __call__(self, task: str, img: Optional[str] = None) -> Any:
    """
    Makes the object callable and returns the result of the run method.

    Args:
        task (str): Task to send to the model.
        img (Optional[str]): Path to the image (optional).

    Returns:
        Any: The response from the model.
```

"""

return self.run(task, img)