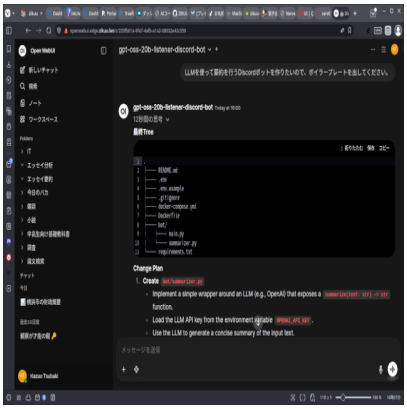


gpt-oss:20bã??ä½¿ã•£ã•¦ã??ã?ã?°ã?©ã? ã??é??ç?ºã•?ã??ä¸?æ??

Description

default watermark

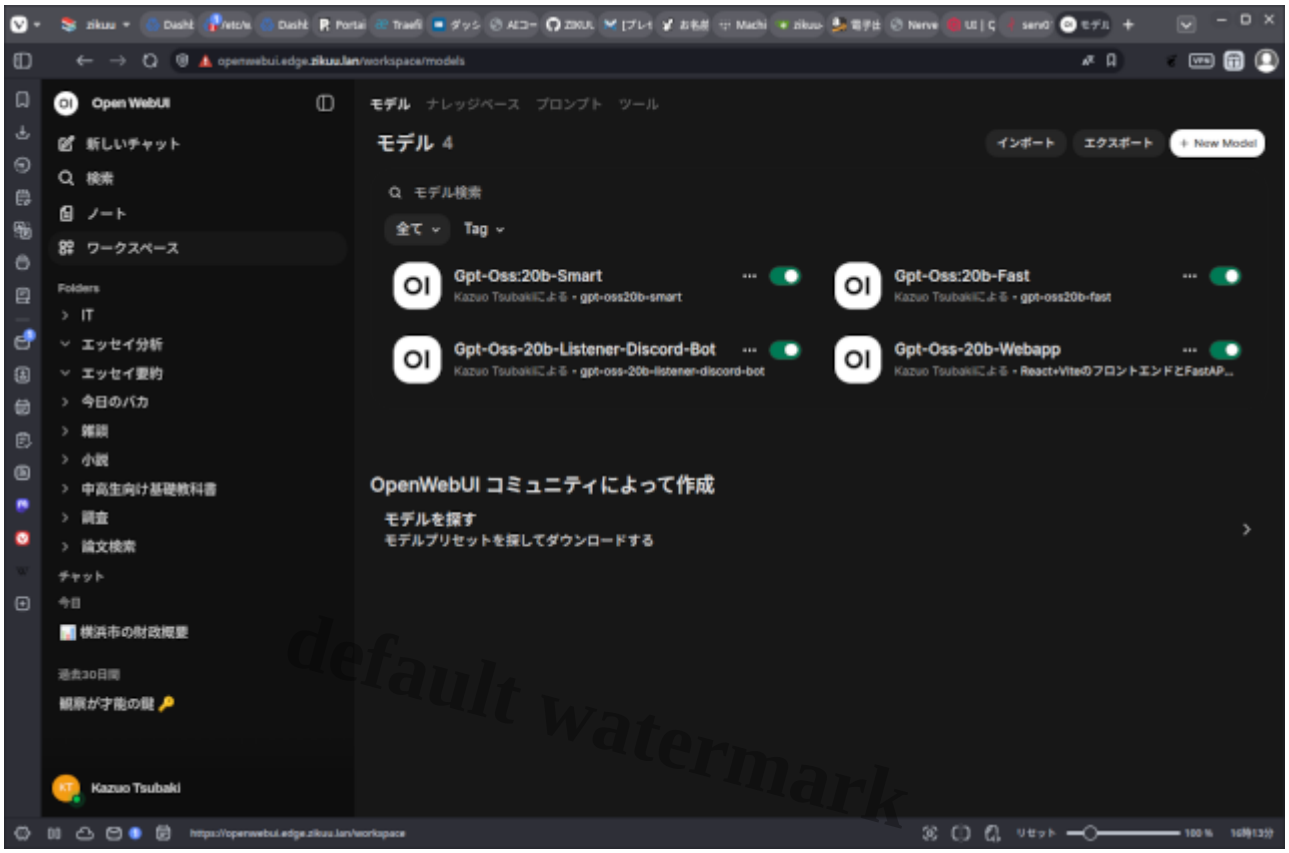
ã•ªã•?ã•ªã•?**ã?ã?¼ã?«ã?«LLM**ã??ä½¿ã•?å??ã??ã•®ã•¯é?£ã?ã•?ã•¨æ?•ã•£ã•¦ã•¾ã•?ã??ã??ã?-ã?³ã??ã??ã?•ã?¥ã?¼ã??ã?³ã?°ã??ã•?ã??ã•?ã?•RAGã•?ã?•ã??ã?¡ã?¤ã?³ã?•ã?¥ã?¼ã??ã?³ã?°ã•?ã•¨ã•?ã•?é•¸æ??ã?»çµ?ã•¿å•?ã?•ã•?ã•®ä»?ã•«ã??ã?•ã•©ã•?ã??ã?ã?°ã?©ã? ã•®ä¸ã•«çµ?ã•¿è¾¼ã??ã•§ã?•ã?·ã?¹ã??ã? ã•«çµ±å•?ã•?ã•¦ã•?ã••ã•®ã•?ã??è??ã•?ã??ã•?ã•¯ã•¯ã?ã••ã•?ã??ã•?ã??ã•¾ã•?ã??å®?é¨?ã•?ã•ªã•?ã??ã?•è?¯ã?ã••ã•?ã•ªæ?¹æ³?ã??è¦?ã•¤ã•?ã??ã??ã••ã??ã•ªæ?¥ã? ã•?ç¶?ã•?ã•¦ã•?ã•¾ã•?ã??

ä»?å??ã•¯ã?**OpenWebUI**ã•¨**Ollama**ã•§ã?•**gpt-oss:20b**ã??ä½¿ã•£ã•?ã??ã?ã?°ã?©ã? ã??ç??æ?•ã•?ã??ã•?ã?•ã•®ã??ã?ã?³ã??ã??ã??æ?¸ã•?ã•¦å®?é¨?ã•?ã•¾ã•?ã•?çµ•æ??ã??ã•?è¦?ã•?ã•?ã•¾ã•?ã??ã?ã?°ã?©ã??ã?³ã?°æ?ªçµ?é¨?è? ç?¨ã•§ã•¯ã?ã??ã•¾ã•?ã??ã?ã•?ã??ç¨?åº¦ã??ã?-ã?°ã?©ã??ã?³ã?°çµ?é¨?ã•?ã•?ã??äººã??æ?³å®?ã•?ã•¦ã•?ã•¾ã•?ã??

OpenWebUIã•§Ollamaã•®è¨å®?ã??ã•?ã•¾ã•?

OpenWebUIã••ã•¯ã?«ã?¹ã?¿ã? ã?¢ã??ã?«ã??ä½?ã??æ©?è?½ã?ã•?ã??ã?•ã•?ã•®æ©?è?½ã??ä½¿ã•?ã•¨ã?•æ?¢å?ã•®ã?¢ã??ã?«ã••ç?¬è?ªã•®è¨å®?ã??ã•?ã•?ã??ã•®ã??ã?•è?ªå??ã•®ã?¢ã??ã?«ã•¨ã•?ã•¦æ?±ã•?ã•?ã•¨ã•?ã•§ã••ã•¾ã•?ã??

ã?µã?¤ã??ã?•ã?¼ã•®ã??ã?¯ã?¼ã?¯ã?¹ã??ã?¼ã?¹ã?ã??é•¸æ??ã•?ã?•ç?»é•¢å•³å•´ã•®ã??ã?¢ã??ã?«ã?•ã?¿ã??ã??é•¸ã•¶ã•¨ã?•ã?«ã?¹ã?¿ã? ã?¢ã??ã?«ã??ç®¡ç•?ã•?ã??ç?»é•¢ã••ã•ªã??ã•¾ã•?ã??ã•?ã•§å•³ä¸?ã•®ã??**+ New Model**ã?•ã??æ?¼ã•?ã•¾ã•?ã??



ã?¢ã??ã?«ã•«å¥½ã•¿ã•®å••å?•ã??ä»?ã•?ã?•å?ºæ?¬ã•¨ã•ªã??ã?¢ã??ã?«ã??é•¸æ??ã•?ã•¾ã•?ã??ã•?ã•?ã•§**gpt-oss:20b**ã??å?ºæ?¬ã?¢ã??ã?«ã•¨ã•?ã•¦é•¸ã??ã•§ã•?ã•¾ã•?ã??

ä»?å??ã•¯Discordã??ã??ã??ã??é??ç?ºã?ã??ã•?ã?•ã•®ã?¢ã??ã?«ã??ä½?æ?•ã•?ã•¦ã•¿ã•¾ã•?ã??

ã?·ã?¹ã??ã? ã??ã?ã?³ã??ã??

æ¬¡ã•®ã??ã•?ã•«ã?·ã?¹ã??ã? ã??ã?ã?³ã??ã??ã??è¨å®?ã•?ã•¾ã•?ã??

ã•?ã•ªã•?ã•¯é??ç?ºæ?¯æ•´AIã??æ?¨æ¸¬ã•§æ?¢å?ã??ã?¡ã?¤ã?«å??å®¹ã??æ±ºã?•ã•ªã•?ã??ã¿?è¦•ã•ªã??å??æ?¬ã??è¦•æ±?ã•?ã??ã??

* é?²ã?•æ?¹ã•¯å¿?ã•? CP (Change Plan)â??å??æ?¬è«?æ±?â??æ?¿èª•â??å®?è£?â??æ¤?è¨¼ã??æ?¿èª•ã•ªã•?ã•«å®?è£?ã•?ã•ªã•?ã??
* å¤?æ?´ã•¯å??å?? å? ç®?ï¼?+N/-0ï¼?ã??ã?ªã?•ã?¼ã? ã?»ç§»å??ã?»å??é?¤ã?»ã•¾ã•¨ã?•ã•¦ã?ªã??ã?¡ã?¯ã?¿ç¦•æ¢ã??
* å?ºå??ã•¯ã•¾ã•?ã??æ??çµ?Treeã?•ã??ç¢ºå®?ã•?ã?•Change Plan ï¼?å®?è£?æ¡?ï¼?ã??å?ºã•?ã?•ã••ã??ä»¥å¤?ã•®ã??ã?¹ã??å?ºã•?ã•ªã•?ï¼?è?ªå·±ã?•ã?§ã??ã?¯ï¼?ã??
* .gitignore ã•§ docker-compose.yml ã•¨ Dockerfile ã??ç?¡è¦?ã•?ã??è¡?ã•¯ç¦•æ¢
* CPã??å?ºã•?ã•?ã•?ã•¨ã•¯ã?•"æ?¿èª•"ã?•"OK"ã?•"ç¶?ã•?ã•¦"ã•ªã•©ã•®ã?¡ã??ã?»ã?¼ã?¸ã•?ã•?ã•£ã•?ã??ã?•å®?è£?ã•?å??æ?¬è«?æ±?ã??ã•?ã??ã??
* ã?½ã?¼ã?¹ã??å¤?æ?´ã•?ã??å ´å•?ã•¯ã?•å??æ?¬è«?æ±?ã??ã•?ã•¦ã•?ã??ã?•å®?è£?ã•®æ?¿èª•ã??å¾?ã•¤ã??
* Dockerfile ã•¯å¿?é ?ã•ªã•®ã•§ Treeã•«å•«ã?•ã??ã??
* .env.example ã??Treeã•«å•«ã?•ã??ã??
* Dockerfileã•®ã?•ã?¼ã??EXPOSEã•¯ä¸•è¦•ã??
* ALLOWED_CHANNEL_IDS ä»¥å¤?ã•¯å®?å?¨ç?¡è¦?
* botã•®ç?ºè¨?ã•¯ç?¡è¦?ï¼?è?ªå??ã?»ä»?ã??ã??ã??ï¼?
* â??é?¢é?£è³ªå?•/æ¬¡ã?¿ã?¹ã?¯â?•ã??å?•æ??ã•«ç??ã??ã•?ã•ªã•?
* æ?¿èª•å?•ã•¯ Tree+CP ã•®ã•¿ï¼?æ?¬æ??ç¦•æ¢ï¼?

ã??ã?ã?¸ã?§ã?¯ã??å??:

* Python / discord.py
* Listeneri¼?on_messageï¼?ã??ä¸»è»¸
* è¨±å•¯ã?•ã?£ã?³ã?•ã?«IDï¼?è¤?æ?°ï¼?ã• ã•?å••å¿?
* å••å¿?ã•¯ã??æ??å°•ã•®è¿?ä¿¡ã?•ã• ã•?ï¼?ä¾?ï¼?å•?ã•?å•?ã•£ã•?ã•?ã•¨ã•?å??ã•?ã??ç?æ??ï¼?

æ¨?æº?Treeã•¯ä»¥ä¸?ã•«å?ºå®?ã??

```
.
â??â??â?? README.md
â??â??â?? .env
â??â??â?? .gitignore
â??â??â?? docker-compose.yml
â??â??â?? bot/
â??   â??â??â?? main.py
â??â??â?? requirements.txt
```

.envã•®ã?ã?¼:

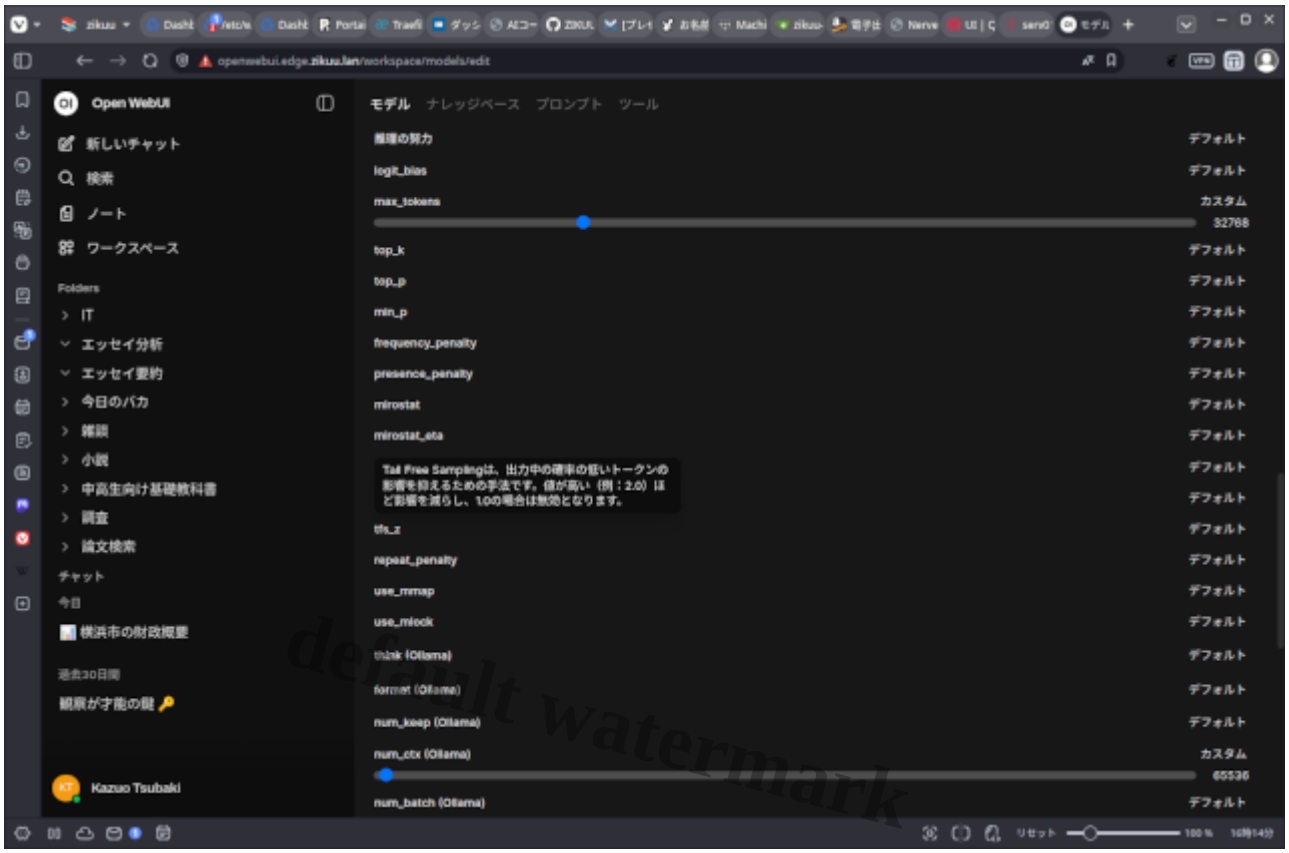
* DISCORD_TOKEN
* ALLOWED_CHANNEL_IDSï¼?ã?«ã?³ã??å?ºå??ã??ï¼?

ã??ã?¤ã?©ã?¼ã??ã?¬ã?¼ã??ã•®æ®µé??ã•§ã•¯ã?•backendã•®APIè¿?ä¿¡ã•¯â?•OK"ã•ªã•©ã•®ç?æ??ã•§è?¯ã•?ã??

ã??ã?©ã?¡ã?¼ã?¿ã?¼

æ¬¡ã•«ã?¢ã??ã?«ã??ã?©ã?¡ã?¼ã?¿ã?¼ã??è¨å®?ã•?ã•¾ã•?ã??

default watermark



ã•?ã•¯æ¤?è¨?ã®ä½?å?°ã•?ã•¾ã• ã•?ã??ã•¾ã•?ã•?ã?•ä½¿ç?¨ã•?ã•¦ã•?ã??GPUã•®æ?§è?½ã??VRAMå®¹é?•ã•«ã??ã•£ã•¦è¹¿æ?´ã•?ã•ªã•?ã??ã•°ã•ªã??ã•¾ã•?ã??ã??

ä»?å??è¨å®?ã•?ã•?ã•®ã•¯**Temperature**ã?•**max_tokens**ã?•**num_ctx (Ollama)**ã•®3ã•¤ã??

- Temperature = 0.2ã•?0.4
- max_token = 32K (32768)
- num_ctx (Ollama) = 64K (65536)

ä½¿ç?¨ã•?ã•¦ã•?ã??GPUã•¯RTX 2000 Ada Lovelace 16GB VRAMã??
ã•?ã•¾ã??é??ã••ã•¯ã?ã??ã•¾ã•?ã??ã•?ã?•ç?•é?»å??ã•§ç?±å¯¾ç?ã?æ¥½ã•§é•?ã•?ã•ªè?¯ã?GPUã•§ã•?ã??
ã•?ã•®ç?¶æ ?ã•§å??ã•?ã•?ã•¨ã?•VRAMã•¯14GBã»ã•©å°?æ??ã•?ã?•ã?¢ã??ã?«ã•¨KVã?-ã?£ã??ã?·ã?¥ã•?VRAMã•«ä¹?ã??ã•®ã•§ã?•RAMã•¨ã•®é??ã•®ã??ã?¼ã?¿ã??ã??å•?ã??ã•?æ¸?ã•£ã•¦é??ã••å??ä½?ã•?ã•¾ã•?ã??

LMStudioã•ªã•©ã??ä½¿ã•?å´å•?ã??ã?•å•?æ§?ã•®è¨å®?ã??ã•?ã??ã•°è?¯ã?ã•¨æ?•ã•?ã•¾ã•?ã??

å®?é??ã•«ã??ã?ã?°ã?©ã? ã??ç??æ?•ã•?ã•¦ã•¿ã??

ã•§ã•¯ã?æ?©é??ã?•ã??ã?ã?°ã?©ã? ã??ç??æ?•ã•?ã•¦ã•¿ã•¾ã•?ã??

ã?¿ã?¼ã?³ï¼?

ã??ã?ã?³ã??ã??

LLMã??ä½¿ã•£ã•¦è¦•ç´?ã??è¡?ã•?Discordã??ã??ã??ã??ä½?ã??ã•?ã•?ã•®ã•§ã?•
ã??ã?¤ã?©ã?¼ã??ã?¬ã?¼ã??ã??å?ºã•?ã•¦ã••ã• ã•?ã•?ã??

ã?¬ã?¹ã?•ã?³ã?¹

æ??çµ?Tree

```
.
â??â??â?? README.md
â??â??â?? .env
â??â??â?? .env.example
â??â??â?? .gitignore
â??â??â?? docker-compose.yml
â??â??â?? Dockerfile
â??â??â?? bot/
â??   â??â??â?? main.py
â??   â??â??â?? summarizer.py
â??â??â?? requirements.txt
```

default watermark

Change Plan

1. **Create `bot/summarizer.py`**

- Implement a simple wrapper around an LLM (e.g., OpenAI) that exposes a `summarize(text: str) -> str` function.
- Load the LLM API key from the environment variable `OPENAI_API_KEY`.
- Use the LLM to generate a concise summary of the input text.

1. **Modify `bot/main.py`**

- Import `summarizer`.
- Add an `on_message` listener that:
  - Checks if the messageâ??s channel ID is in the list obtained from `ALLOWED_CHANNEL_IDS`.
  - If allowed, calls `summarizer.summarize(message.content)` to get a summary.
  - Replies to the message with the summary (or a minimal acknowledgement if summarization fails).
- Ensure the bot only reacts to messages in the allowed channels.

1. **Add `.env.example`**

- Include placeholders for:
  - `DISCORD_TOKEN`
  - `ALLOWED_CHANNEL_IDS` (commaâ??separated)
  - `OPENAI_API_KEY`

1. **Update `requirements.txt`**

- Add `discord.py` and `openai` (or the relevant LLM client library).

1. **Create `Dockerfile`**

- Base image: `python:3.11-slim`.
- Copy `requirements.txt` and install dependencies.
- Copy the `bot/` directory.
- Set environment variables (e.g., `DISCORD_TOKEN`, `ALLOWED_CHANNEL_IDS`, `OPENAI_API_KEY`).
- Run the bot with `python -m bot.main`.

1. **Ensure `docker-compose.yml` references the new Dockerfile**

- No changes needed if it already builds the image from the Dockerfile.

This plan adds the necessary files and modifications to implement a Discord bot that uses an LLM to summarize messages in allowed channels, while keeping the project structure and constraints intact.

ã?¿ã?¼ã?³ï¼?

default watermark

ã??ã?ã?³ã??ã??

docker-compose.ymlã•®ä¸è°«ã??å?ºã•?ã•¦ã??

ã?¬ã?¹ã?•ã?³ã?¹

```
version: "3.9"

services:
  bot:
    build: .
    env_file: .env
    restart: unless-stopped
```

ã?¿ã?¼ã?³ï¼?

ã??ã?ã?³ã??ã??

Dockerfileã•®å? å®¹ã??æ??ã•?ã•¦ã••ã• ã•?ã•?ã??

ã?¬ã?¹ã?•ã?³ã?¹

```
# ã??ã?¼ã?¹ã?¤ã?¡ã?¼ã?¸
FROM python:3.11-slim

# ä½?æ¥ã??ã?£ã?¬ã?¯ã??ã?ªã??ä½?æ?•
WORKDIR /app

# ä¾•å?é?¢ä¿?ã??ã?¤ã?³ã?¹ã??ã?¼ã?«
COPY requirements.txt .
```

```
RUN pip install --no-cache-dir -r requirements.txt

# ã?¢ã??ã?ªã?±ã?¼ã?·ã?§ã?³ã?³ã?¼ã??ã??ã?³ã??ã?¼
COPY bot ./bot

# ç?°å¢?å¤?æ?°ã•¯ dockerâ??compose ã•§èªã•¿è¾¼ã??ã•®ã•§ã•?ã•?ã•§ã•¯è¨å®?ã•?ã
•ªã•?

# ã??ã??ã?©ã?«ã??ã•®å®?è¡?ã?³ã??ã?³ã??
CMD ["python", "-m", "bot.main"]
```

ã?¿ã?¼ã?³ï¼?

ã??ã?ã?³ã??ã??

requirements.txtã•®ä¸è°«ã??æ??ã•?ã•¦ã••ã• ã•?ã•?ã??

ã?¬ã?¹ã?•ã?³ã?¹

default watermark

requirements.txt

```
discord.py==2.3.2
openai==1.12.0
```

ã?¿ã?¼ã?³ï¼?

ã??ã?ã?³ã??ã??

bot/main.pyã•®ä¸è°«ã??æ??ã•?ã•¦ã••ã• ã•?ã•?ã??

ã?¬ã?¹ã?•ã?³ã?¹

bot/main.py

```
import os
import discord
from discord.ext import commands
from bot import summarizer

# ç?°å¢?å¤?æ?°ã•?ã??è¨å®?ã??å•?å¾?
DISCORD_TOKEN = os.getenv("DISCORD_TOKEN")
ALLOWED_CHANNEL_IDS = os.getenv("ALLOWED_CHANNEL_IDS", "")
# ã?«ã?³ã??å?°å??ã??ã•®æ??å?å??ã??æ?´æ?°ã•®ã?»ã??ã??ã•«å¤?æ•?
ALLOWED_CHANNEL_IDS = {
    int(ch.strip()) for ch in ALLOWED_CHANNEL_IDS.split(",") if ch.strip().isd
}

intents = discord.Intents.default()
intents.message_content = True  # ã?¡ã??ã?»ã?¼ã?¸æ?¬æ??ã??å•?å¾?ã•?ã??ã•?ã?•
ã•«å¿?è¦•
```

```
bot = commands.Bot(command_prefix="!", intents=intents)


@bot.event
async def on_ready():
    print(f"Logged in as {bot.user} (ID: {bot.user.id})")


@bot.event
async def on_message(message: discord.Message):
    # è?ªå??ã•®ã?¡ã??ã?»ã?¼ã?¸ã•«ã•¯å••å¿?ã•?ã•ªã•?
    if message.author.id == bot.user.id:
        return

    # è¨±å•¯ã•?ã??ã•?ã?•ã?£ã?³ã?•ã?«ã•?ç¢ºèª•
    if message.channel.id not in ALLOWED_CHANNEL_IDS:
        return

    # è¦•ç´?ã??å•?å¾?
    try:
        summary = await summarizer.summarize(message.content)
    except Exception as e:
        # ä½?ã??ã•?ã•®ã?¨ã?©ã?¼ã•?èµ·ã••ã•?ã??ç°¡æ??å¿?ç?
        summary = "è¦•ç´?ã•§ã••ã•¾ã•?ã??ã•§ã•?ã•?ã??"

    # æ??å°•é?•ã•®è¿?ä¿¡
    await message.reply(summary, mention_author=False)


if __name__ == "__main__":
    bot.run(DISCORD_TOKEN)
```

default watermark

ã?¿ã?¼ã?³ï¼?

ã??ã?ã?³ã??ã??

summarizerã?¢ã?¸ã?¥ã?¼ã?«ã•®å®?è£ ã•¯ã•©ã•?ã•ªã•£ã•¦ã•?ã•¾ã•?ã•?ï¼?

ã?¬ã?¹ã?•ã?³ã?¹

bot/summarizer.py

```
"""
A very small wrapper around the OpenAI API that turns a piece of text
into a concise summary.

The module exposes a single coroutine:

    async def summarize(text: str) -> str

It uses the `openai` Python package (see requirements.txt) and expects
```

```
the environment variable `OPENAI_API_KEY` to be set.  If the key is
missing or the API call fails, the function returns a short error
message so that the bot can still reply with something useful.
"""

import os
import asyncio
from typing import Optional

import openai

# Load the API key once at import time
OPENAI_API_KEY: Optional[str] = os.getenv("OPENAI_API_KEY")
if OPENAI_API_KEY:
    openai.api_key = OPENAI_API_KEY


async def summarize(text: str) -> str:
    """
    Return a short summary of *text* using the OpenAI ChatCompletion API.

    Parameters
    ----------
    text : str
        The original message content to summarise.

    Returns
    -------
    str
        A concise summary.  If the request fails, a short error string
        is returned instead.
    """
    if not OPENAI_API_KEY:
        return "OpenAI API key not configured."

    # The request is performed in a threadâ??pool to avoid blocking the
    # event loop.  The OpenAI library is synchronous, so we wrap it
    # with asyncio.to_thread.
    try:
        response = await asyncio.to_thread(
            openai.ChatCompletion.create,
            model="gpt-3.5-turbo",
            messages=[
                {
                    "role": "system",
                    "content": (
                        "You are a helpful assistant that produces short "
                        "summaries of the given text. Keep the summary to "
                        "one or two sentences."
                    ),
                },
                {"role": "user", "content": text},
            ],
            temperature=0.2,
            max_tokens=60,
```

default watermark

```
        )
        # The assistant's reply is in the first choice's message content.
        summary = response.choices[0].message.content.strip()
        return summary or "è¦•ç´?ã•§ã••ã•¾ã•?ã??ã•§ã•?ã•?ã??"
    except Exception as exc:
        # Log the exception if you have a logger; for now we just return
        # a userâ??friendly message.
        return "è¦•ç´?ã•§ã••ã•¾ã•?ã??ã•§ã•?ã•?ã??"
```

ã?¿ã?¼ã?³ï¼?

ã??ã?ã?³ã??ã??

.env.exampleã•®å? å®¹ã??æ??ã•?ã•¦ã??

ã?¬ã?¹ã?•ã?³ã?¹

.env.example

default watermark

```
# Discord bot token
DISCORD_TOKEN=your_discord_bot_token_here

# Commaâ??separated list of channel IDs the bot should listen to
ALLOWED_CHANNEL_IDS=123456789012345678,987654321098765432

# OpenAI API key for summarisation
OPENAI_API_KEY=sk-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

ã?¿ã?¼ã?³ï¼?

ã??ã?ã?³ã??ã??

.gitignoreã•®å? å®¹ã??æ??ã•?ã•¦ã??

ã?¬ã?¹ã?•ã?³ã?¹

.gitignore

```
# Byteâ??compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
```

```
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
pip-wheel-metadata/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

# Virtual environment
.venv/
env/
ENV/
env.bak/
venv/
ENV.bak/

# PyInstaller
#  Usually these files are written by a python script from a template
#  when you run pyinstaller.
*.manifest
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

# Unit test / coverage reports
htmlcov/
.tox/
.nox/
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
*.cover
*.py,cover
*.db
*.sqlite

# PyCharm
.idea/

# VS Code
.vscode/
```

default watermark

```
# Environment variables
.env

# Logs
*.log

# Docker
# (Do NOT ignore docker-compose.yml or Dockerfile)
# docker-compose.yml
# Dockerfile

# Misc
*.swp
*.swo
*.swn
*.tmp
*.bak
*.orig
*.rej
*.sublime-workspace
*.sublime-project
```

default watermark

ã?¿ã?¼ã?³ï¼?

ã??ã?ã?³ã??ã??

summerizer.pyã•¯OpenAIã??ä½¿ã?ã•®ã•§ã•¯ãªã••ã?•ã?ã?¼ã?«ã?«ã•§å??ã•?ã•¦ã•?ã??Ollamaã??ä½¿ã•?ã•?ã•?ã??

ã?¬ã?¹ã?•ã?³ã?¹

```
"""
A very small wrapper around the Ollama API that turns a piece of text
into a concise summary.

The module exposes a single coroutine:

    async def summarize(text: str) -> str

It uses the `ollama` Python package (see requirements.txt) and expects
the environment variable `OLLAMA_MODEL` to be set.  If the variable is
missing, it defaults to "llama3".  If the request fails, the function
returns a short error message so that the bot can still reply with
something useful.
"""

import os
import asyncio
from typing import Optional

import ollama
```

```
# Load the model name once at import time
OLLAMA_MODEL: str = os.getenv("OLLAMA_MODEL", "llama3")


async def summarize(text: str) -> str:
    """
    Return a short summary of *text* using the Ollama chat API.

    Parameters
    ----------
    text : str
        The original message content to summarise.

    Returns
    -------
    str
        A concise summary.  If the request fails, a short error string
        is returned instead.
    """
    # The request is performed in a threadâ??pool to avoid blocking the
    # event loop.  The Ollama client is synchronous, so we wrap it
    # with asyncio.to_thread.
    try:
        response = await asyncio.to_thread(
            ollama.chat,
            model=OLLAMA_MODEL,
            messages=[
                {
                    "role": "system",
                    "content": (
                        "You are a helpful assistant that produces short "
                        "summaries of the given text. Keep the summary to "
                        "one or two sentences."
                    ),
                },
                {"role": "user", "content": text},
            ],
            temperature=0.2,
            max_tokens=60,
        )
        # The assistant's reply is in the 'message' field of the response.
        summary = response.get("message", {}).get("content", "").strip()
        return summary or "è¦•ç´?ã•§ã••ã•¾ã•?ã??ã•§ã•?ã•?ã??"
    except Exception:
        # Return a userâ??friendly message on failure.
        return "è¦•ç´?ã•§ã••ã•¾ã•?ã??ã•§ã•?ã•?ã??"
```

default watermark

æ??å¾?ã•«

ã•?ã??ã•ªæ??ã•?ã•§ã?•ã??ã??å•?ã??ã•?ã•ªã•?ã??ã?•ã??ã?ã?°ã?©ã? ã??ä½?ã•£ã•¦ã•?ã••ã•¾ã•?ã??
LLMã•®å?ºå??ã•?é??é•?ã•?ã•¦ã•?ã??ã•?ã•¨ã??ã•?ã??ã•®ã•§ã?•æ£ã•?ã•?ã•®å?¤æ?ã•¯è?ªå??ã•
§ã??ã??ã•¾ã•?ã??

gpt-oss:20bã??æ??å¤§ã?³ã?³ã??ã?¯ã?¹ã??(128K)ã§ä½¿ã•?ã•«ã•¯ã?ã??ã•£ã•¨å¤§ã••ã•ªVRAMã??æ•è¼?ã•?ã•?GPUã•?å¿ è¦•ã•§ã•?ã??æ??å¤§ã?³ã?³ã??ã?¯ã?¹ã??ã§ä½¿ã•?ã??ã•°ã?•é?·ã•?ã?½ã?¼ã?¹ã?³ã?¼ã??ã??ã?•ã?½ã?¼ã?¹é??ã•®é?¢ä¿?ã??è¨?æ?¶ã•?ã•¦å?¦ç•?ã??ã•?ã•¦ã••ã??ã??ã•®ã•§ã?•ã••ã•?ã••ã•?å¤§ã••ã•ªã?¢ã??ã?ªã?±ã?¼ã?·ã?§ã?³ã??é??ç?ºã•§ã••ã??ã•§ã•?ã??ã•?ã??

ã•?ã??ã•§ã??å••å??ã•«ã?•ä¸?ã•?ã??äºæ??ã•§ã??ã??ã??ã??ã•¯é??ã•ä½?æ¥ã??é?²ã?•ã??ã??ã•¾ã•?ã•?ã?•ã?·ã?¹ã??ã? ã??ã?ã?³ã??ã??ã??ã??ã?©ã?¡ã?¼ã?¿ã?¼ã??èª¿æ?´ã•?ã??ã•°ã?•ã??ã•£ã•¨å¿«é•©ã•«ã??ã??ã??ã•§ã•?ã??ã•?ã??

Alã•«å?¤æ?ã??ä»»ã•?ã•ªã•?

ã•?ã??ã• ã•?ã•¯å¸¸ã«æ°?ã•«ç??ã?•ã•¦ã••ã• ã•?ã•?ã??

Category

1. æ?¥ã? ã•®æ´»å??

Tags

1. AI
2. LLM
3. ã??ã?ã?°ã?©ã??ã?³ã?°
4. ä¸?é??å??å¸?
5. å±±æ¢¨ç??

default watermark

Date Created
2026å¹´1æ??22æ?¥
Author
kazuo-tsubaki