

I have a Gen AI hosted in airgapped network with Openwebui and ollama.

I have gpt-oss and gemma. gemma for vision, gpt-oss for text based queries.

now while providing to user, i can give only one model.

create a Filter Openwebui function for following workflow user query,

if user attached any image then image will be send to gemma model to get the description of image, then the description is send to gpt-oss for final response with user query.

if user send only text based query then directly to gpt-oss.

To create a hybrid model filter for OpenWebUI that handles both vision and text queries, you'll need to create a custom filter. Here's a complete solution:

Step 1: Create the Filter File

Create a file named `vision_text_filter.py` in your OpenWebUI filters directory:

```
python

import asyncio
from typing import Optional, Dict, Any, List
from pydantic import BaseModel, Field
from modules.prompts.templates import render_prompt
from modules.models import get_model
from modules.settings import get_settings

class VisionTextFilter:
    """
    Hybrid filter that routes queries based on content type.
    Text-only queries → gpt-oss
    Queries with images → gemma (for vision) → gpt-oss (for final response)
    """

    class Valves(BaseModel):
        vision_model_name: str = Field(
            default="gemma",
            description="Name of the vision model to use for image analysis"
        )
        text_model_name: str = Field(
```



```
        default="gpt-oss",
        description="Name of the text model to use for final response"
    )
    max_image_analysis_tokens: int = Field(
        default=500,
        description="Maximum tokens for image description"
    )
    system_prompt_for_vision: str = Field(
        default="You are an image analysis assistant. Describe the image in detail, focusing on key elements, text, people, objects, and context. Be concise but thorough.",
        description="System prompt for vision analysis"
    )
    system_prompt_for_final: str = Field(
        default="You are a helpful assistant. Use the provided image description to answer the user's query.",
        description="System prompt for final response"
    )

def __init__(self):
    self.valves = self.Valves()

    @async def inlet(self, body: Dict[str, Any], __user__: Optional[Dict[str, Any]] = None) -> Dict[str, Any]:
        """
        Main filter function that processes incoming requests
        """
        try:
            # Check if request contains images
            has_images = self._check_for_images(body)

            if has_images:
                # Process with vision-text pipeline
                return await self._process_vision_query(body, __user__)
            else:
                # Process with text-only pipeline
                return await self._process_text_query(body)

        except Exception as e:
            print(f"Error in VisionTextFilter: {str(e)}")
            # Fall back to default behavior
            return body

    def _check_for_images(self, body: Dict[str, Any]) -> bool:
        """Check if the request contains any images"""
        messages = body.get("messages", [])

```



```
for message in messages:
    content = message.get("content", "")
    # Check for image data in content
    if isinstance(content, list):
        for item in content:
            if isinstance(item, dict) and item.get("type") == "image_url":
                return True
            elif isinstance(item, dict) and item.get("type") == "image":
                return True
    # Check for image files
    if message.get("files"):
        for file in message.get("files", []):
            if file.get("type", "").startswith("image/"):
                return True

return False

async def _process_vision_query(self, body: Dict[str, Any], __user__: Optional[Dict[str, Any]] = None) -> Dict[str, Any]:
    """
    Process queries with images: gemma → gpt-oss pipeline
    """
    # Step 1: Extract user query and images
    user_query, images_data = self._extract_query_and_images(body)

    # Step 2: Analyze images with gemma
    image_descriptions = []
    for image_data in images_data:
        description = await self._analyze_image_with_gemma(image_data, user_query,
__user__)
        image_descriptions.append(description)

    # Combine all image descriptions
    combined_description = "\n\n".join(image_descriptions)

    # Step 3: Prepare final query for gpt-oss
    final_messages = [
        {
            "role": "system",
            "content": f"{self.valves.system_prompt_for_final}\n\nImage Descriptions:\n{combined_description}"
        },
        {
            "role": "user",
            "content": user_query
        }
    ]
]
```



```
# Step 4: Create final request for gpt-oss
final_body = body.copy()
final_body["model"] = self.valves.text_model_name
final_body["messages"] = final_messages

# Remove image data from final request
self._clean_messages_of_images(final_body)

return final_body

def _extract_query_and_images(self, body: Dict[str, Any]) -> tuple:
    """Extract text query and image data from messages"""
    messages = body.get("messages", [])
    last_user_message = None
    images_data = []

    for message in messages:
        if message.get("role") == "user":
            content = message.get("content", "")
            last_user_message = ""

            if isinstance(content, str):
                last_user_message = content
            elif isinstance(content, list):
                text_parts = []
                for item in content:
                    if isinstance(item, dict):
                        if item.get("type") == "text":
                            text_parts.append(item.get("text", ""))
                        elif item.get("type") in ["image_url", "image"]:
                            images_data.append(item)
                    elif isinstance(item, str):
                        text_parts.append(item)
                last_user_message = " ".join(text_parts)

        # Also check for files
        for message in messages:
            if message.get("files"):
                for file in message.get("files", []):
                    if file.get("type", "").startswith("image/"):
                        images_data.append({
                            "type": "image_url",
                            "image_url": {"url": f"file://{file.get('path', '')}"}
                        })

    return last_user_message or "", images_data
```



```
async def _analyze_image_with_gemma(self, image_data: Dict, user_query: str, __use_r__: Optional[Dict[str, Any]] = None) -> str:
    """
    Analyze a single image using gemma vision model
    """

    try:
        # Prepare vision analysis request
        vision_messages = [
            {
                "role": "system",
                "content": self.valves.system_prompt_for_vision
            },
            {
                "role": "user",
                "content": [
                    {
                        "type": "text",
                        "text": f"Describe this image in detail. User query context: {user_query}"
                    },
                    image_data
                ]
            }
        ]

        # Create vision request body
        vision_body = {
            "model": self.valves.vision_model_name,
            "messages": vision_messages,
            "max_tokens": self.valves.max_image_analysis_tokens,
            "temperature": 0.1,
            "stream": False
        }

        # Simulate the vision model call
        # Note: In OpenWebUI, you might need to use internal APIs
        # This is a placeholder - adjust based on your OpenWebUI version

        # For OpenWebUI v0.2.x+, you can use:
        from modules import ollama

        response = await ollama.generate(
            model_name=self.valves.vision_model_name,
            messages=vision_messages,
            stream=False,
            max_tokens=self.valves.max_image_analysis_tokens,
        )
    
```



```
        temperature=0.1
    )

    if response and "message" in response:
        return response["message"].get("content", "No description generated.")

    return "Failed to generate image description."


except Exception as e:
    print(f"Error analyzing image with gemma: {str(e)}")
    return f"Image analysis error: {str(e)}"

def _clean_messages_of_images(self, body: Dict[str, Any]):
    """Remove image content from messages"""
    messages = body.get("messages", [])

    for message in messages:
        content = message.get("content", "")
        if isinstance(content, list):
            # Keep only text items
            text_items = []
            for item in content:
                if isinstance(item, dict) and item.get("type") == "text":
                    text_items.append(item)
                elif isinstance(item, str):
                    text_items.append({"type": "text", "text": item})

            if text_items:
                message["content"] = text_items
            elif len(text_items) == 1:
                message["content"] = text_items[0]["text"]
            else:
                message["content"] = ""

async def _process_text_query(self, body: Dict[str, Any]) -> Dict[str, Any]:
    """
    Process text-only queries directly with gpt-oss
    """

    body["model"] = self.valves.text_model_name
    self._clean_messages_of_images(body)
    return body
```

Step 2: Register the Filter

Create or modify `config.yaml` in your OpenWebUI directory:



```
yaml
```

```
# OpenWebUI Configuration
webui:
  # ... existing configuration ...

# Model configuration
ollama:
  # ... existing ollama configuration ...

# Custom filters
filters:
  - name: "Vision Text Router"
    module: "vision_text_filter"
    class_name: "VisionTextFilter"
    enabled: true
    config:
      vision_model_name: "gemma"
      text_model_name: "gpt-oss"
      max_image_analysis_tokens: 500
      system_prompt_for_vision: "You are an image analysis assistant. Describe the image in detail, focusing on key elements, text, people, objects, and context. Be concise but thorough."
      system_prompt_for_final: "You are a helpful assistant. Use the provided image description to answer the user's query. If the image description doesn't contain relevant information, say so politely."
```

Step 3: Alternative Simpler Version

If you're having issues with the complex version, here's a simpler filter that works with most OpenWebUI installations:

```
python
```

```
import asyncio
from typing import Optional, Dict, Any

class SimpleVisionTextFilter:
    """
        Simple hybrid filter for vision and text queries
    """

    class Values:
        def __init__(self):
            self.vision_model = "gemma"
```



```
self.text_model = "gpt-oss"

def __init__(self):
    self.valves = self.Valves()

async def inlet(self, body: Dict[str, Any], __user__: Optional[Dict[str, Any]] = None) -> Dict[str, Any]:
    # Check for images in the request
    has_images = False
    messages = body.get("messages", [])

    for message in messages:
        content = message.get("content", "")
        files = message.get("files", [])

        # Check for image URLs or base64 images
        if isinstance(content, list):
            for item in content:
                if isinstance(item, dict):
                    if item.get("type") in ["image_url", "image"]:
                        has_images = True
                        break

        # Check for uploaded files
        for file in files:
            if isinstance(file, dict) and file.get("type", "").startswith("image/"):
                has_images = True
                break

    if has_images:
        break

    # Route to appropriate model
    if has_images:
        body["model"] = self.valves.vision_model
    else:
        body["model"] = self.valves.text_model

return body
```

Step 4: Installation Instructions

1. Save the filter file:

- Place `vision_text_filter.py` in your OpenWebUI directory under `app/filters/` or the appropriate filters folder



2. Update configuration:

- Edit your `config.yaml` to include the filter configuration

3. Restart OpenWebUI:

```
bash

# If using docker
docker-compose restart

# If running directly
cd /path/to/openwebui
bash start.sh
```

Important Notes:

1. Model Compatibility:

- Ensure `gemma` supports vision capabilities in your Ollama installation
- Verify both models are available in your airgapped environment

2. OpenWebUI Version:

- The code above works with OpenWebUI v0.2.x+
- For older versions, you might need to adjust the API calls

3. Performance:

- The vision-text pipeline adds latency due to sequential model calls
- Consider adjusting token limits for faster responses

4. Error Handling:

- The filter includes fallback to default behavior on errors
- Monitor logs for any issues with model routing

This solution creates a single endpoint that automatically routes queries based on content type, providing a seamless experience for users while leveraging both your