? Qwen3.7 Max compare with others

This is writing about Qwen3.7 Max review and fix a random python code from github.

> **input prompt:** Review and help me optimize this python file.

? Original

Click to show code

```
from subprocess import Popen, PIPE
import gradio as gr
import numpy as np
import cv2
import os
from pathlib import Path


def inference_gfpgan(img_path, resize):
    return run_task("_gfpgan", img_path, resize=resize)


def inference_gpen(img_path, task):
    return run_task("_gpen", img_path, task=task)


def run_task(model, img_path, task="restore", resize="on"):
    # do the inference in a seperate process to avoid high gpu idle power consumption of
pytorch
    p = Popen(["python", "app/%s.py" % model, img_path, task, resize], stdout=PIPE,
stderr=PIPE)
    # info
    img_name = os.path.basename(img_path)
    basename, ext = os.path.splitext(img_name)
    model = model.strip("_")
    print(f"Started {model.upper()} {task} {img_name} ...")
    # wait for it to finish
    stdout, stderr = p.communicate()
    print("stdout:\n", stdout)
    print("stderr:\n", stderr)
    # retrieved the restored image
```

```
    extension = ext[1:]
    restored_img_path = os.path.join(
        "outputs", f"{basename}_{model}_{task}.{extension}"
    )
    if os.path.isfile(restored_img_path):
        restored_img = cv2.imread(restored_img_path)
        restored_img = cv2.cvtColor(restored_img, cv2.COLOR_BGR2RGB)
        print(f"Finished {model.upper()} {task} {restored_img_path}")
        return restored_img
    else:
        gr.Error("Result not found!")
    return None


def get_processed_files(n=4):
    """
    Return last n files
    """
    paths = sorted(Path("outputs").iterdir(), key=os.path.getmtime)
    files = []
    for i in range(min(n,len(paths))):
        files.append(str(paths[-(i + 1)]))
    return files


with gr.Blocks(title="GFPGAN") as GFPGAN_app:
    gr.Markdown(
        """
    # Generative Facial Prior GAN
    """
    )
    with gr.Row():
        input_img = gr.Image(type="filepath", label="Input")
        output_img = gr.Image(type="numpy", label="Output")
    with gr.Row():
        resize_radio = gr.Radio(
            choices=["on", "off"], label="Resize", value="on"
        )
        submit_btn = gr.Button("Submit", variant="primary")
        get_btn = gr.Button("Get")

    prev_files = gr.Gallery(
        label="Downloadables",
        show_label=False,
        elem_id="gallery",
        columns=[2],
        rows=[1],
        object_fit="contain",
        height="auto",
        allow_preview=False,
    )

    submit_btn.click(inference_gfpgan, inputs=[input_img,resize_radio],
outputs=[output_img])
    get_btn.click(get_processed_files, outputs=[prev_files])
```

```
with gr.Blocks(title="GPEN") as GPEN_app:
    gr.Markdown(
        """
    # GAN Prior Embedded Network
    """
    )
    with gr.Row():
        input_img = gr.Image(type="filepath", label="Input")
        output_img = gr.Image(type="numpy", label="Output")
    with gr.Row():
        task_radio = gr.Radio(
            choices=["colorize", "restore"], label="Task", value="colorize"
        )
        submit_btn = gr.Button("Submit", variant="primary")

    submit_btn.click(
        inference_gpen, inputs=[input_img, task_radio], outputs=[output_img]
    )
    get_btn.click(get_processed_files, outputs=[prev_files])


app = gr.TabbedInterface([GFPGAN_app, GPEN_app], ["GFPGAN", "GPEN"])

app.queue()
app.launch(server_name="0.0.0.0")
```

? Gemini

Click to show code

```
from subprocess import Popen, PIPE
import gradio as gr
import cv2
import os
from pathlib import Path

OUTPUT_DIR = Path("outputs")
OUTPUT_DIR.mkdir(exist_ok=True)

def run_task(model: str, img_path: str, task: str = "restore", resize: str = "on"):
    img_path_obj = Path(img_path)
    basename = img_path_obj.stem
    ext = img_path_obj.suffix

    model_clean = model.strip("_")
    print(f"Started {model_clean.upper()} | Task: {task} | Image: {img_path_obj.name} ...")

    p = Popen(
        ["python", f"app/{model}.py", str(img_path_obj), task, resize],
        stdout=PIPE,
        stderr=PIPE
```

```
    )
    stdout, stderr = p.communicate()

    print("--- STDOUT ---")
    print(stdout.decode("utf-8", errors="ignore"))
    print("--- STDERR ---")
    print(stderr.decode("utf-8", errors="ignore"))

    extension = ext.lstrip(".")
    restored_img_path = OUTPUT_DIR / f"{basename}_{model_clean}_{task}.{extension}"

    if restored_img_path.is_file():
        restored_img = cv2.imread(str(restored_img_path))
        restored_img = cv2.cvtColor(restored_img, cv2.COLOR_BGR2RGB)
        print(f"Finished {model_clean.upper()} -> {restored_img_path}")
        return restored_img
    else:
        raise gr.Error(f"Result not found at {restored_img_path}!")


def inference_gfpgan(img_path, resize):
    return run_task("_gfpgan", img_path, task="restore", resize=resize)


def inference_gpen(img_path, task):
    return run_task("_gpen", img_path, task=task, resize="on")


def get_processed_files(n=4):
    if not OUTPUT_DIR.exists():
        return []
    paths = sorted([p for p in OUTPUT_DIR.iterdir() if p.is_file()], key=os.path.getmtime)

    latest_files = [str(p) for p in paths[-n:]][::-1]
    return latest_files


with gr.Blocks(title="GFPGAN") as GFPGAN_app:
    gr.Markdown("# Generative Facial Prior GAN")
    with gr.Row():
        input_img = gr.Image(type="filepath", label="Input")
        output_img = gr.Image(type="numpy", label="Output")
    with gr.Row():
        resize_radio = gr.Radio(choices=["on", "off"], label="Resize", value="on")
        submit_btn = gr.Button("Submit", variant="primary")
        get_btn = gr.Button("Get Recent Files")

    prev_files = gr.Gallery(
        label="Downloadables",
        show_label=True,
        columns=4,
        rows=1,
        object_fit="contain",
        height="auto",
        allow_preview=True,
    )
```

```
    submit_btn.click(inference_gfpgan, inputs=[input_img, resize_radio],
outputs=[output_img])
    get_btn.click(get_processed_files, outputs=[prev_files])


with gr.Blocks(title="GPEN") as GPEN_app:
    gr.Markdown("# GAN Prior Embedded Network")
    with gr.Row():
        input_img = gr.Image(type="filepath", label="Input")
        output_img = gr.Image(type="numpy", label="Output")
    with gr.Row():
        task_radio = gr.Radio(choices=["colorize", "restore"], label="Task",
value="restore")
        submit_btn = gr.Button("Submit", variant="primary")
        get_btn_gpen = gr.Button("Get Recent Files") # T?o nút riêng bi?t cho tab này

    prev_files_gpen = gr.Gallery(
        label="Downloadables",
        show_label=True,
        columns=4,
        rows=1,
        object_fit="contain",
        height="auto",
        allow_preview=True,
    )

    submit_btn.click(inference_gpen, inputs=[input_img, task_radio], outputs=[output_img])
    get_btn_gpen.click(get_processed_files, outputs=[prev_files_gpen])


app = gr.TabbedInterface([GFPGAN_app, GPEN_app], ["GFPGAN", "GPEN"])

if __name__ == "__main__":
    app.queue()
    app.launch(server_name="0.0.0.0")
```

? GPT5

Click to show code

```
from __future__ import annotations

import logging
import sys
import uuid
from pathlib import Path
from subprocess import PIPE, Popen, TimeoutExpired
from typing import Literal

import gradio as gr

OUTPUT_DIR = Path("outputs")
```

```
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

APP_DIR = Path("app")

PROCESS_TIMEOUT = 300
MAX_OUTPUT_FILES = 100
CONCURRENCY_LIMIT = 1


logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s | %(levelname)s | %(message)s",
)

logger = logging.getLogger(__name__)

TaskType = Literal["restore", "colorize"]
ResizeType = Literal["on", "off"]

def cleanup_old_outputs(max_files: int = MAX_OUTPUT_FILES) -> None:
    """

    if not img_path:
        raise gr.Error("No image provided")

    path = Path(img_path)

    if not path.exists():
        raise gr.Error("Input image does not exist")

    if not path.is_file():
        raise gr.Error("Input path is not a file")

    return path



def build_output_path(
    basename: str,
    extension: str,
    model_name: str,
    task: str,
    job_id: str,
) -> Path:
    """
    Create unique output filename.
    """

    filename = (
        f"{basename}_{job_id}_{model_name}_{task}.{extension}"
    )

    return OUTPUT_DIR / filename



def get_latest_output_file(
```

```
    basename: str,
    model_name: str,
    task: str,
) -> Path | None:
    """
    Find latest generated output matching pattern.
    """

    pattern = f"{basename}_*_{model_name}_{task}.*"

    matches = sorted(
        OUTPUT_DIR.glob(pattern),
        key=lambda p: p.stat().st_mtime,
        reverse=True,
    )

    return matches[0] if matches else None

def run_task(
        stdout=PIPE,
        stderr=PIPE,
        text=True,
    )

    try:
        stdout, stderr = process.communicate(timeout=PROCESS_TIMEOUT)

    except TimeoutExpired:
        process.kill()

        logger.error(
            f"Inference timeout | "
            f"model={model_name} | "
            f"image={input_path.name}"
        )

        raise gr.Error(
            f"Inference timeout after {PROCESS_TIMEOUT} seconds"
        )

    logger.info(f"STDOUT:\n{stdout}")

    if stderr.strip():
        logger.warning(f"STDERR:\n{stderr}")

    if process.returncode != 0:
        logger.error(
            f"Inference failed | "
            f"returncode={process.returncode}"
        )

        raise gr.Error(
            f"Inference process failed:\n\n{stderr}"
        )

    output_path = get_latest_output_file(
        basename=basename,
```

```
        model_name=model_name,
        task=task,
    )

    if output_path is None:
        logger.error("Output file not found")
        raise gr.Error("Inference finished but output file not found")

    logger.info(f"Inference completed: {output_path.name}")

    cleanup_old_outputs()

    return str(output_path)

def inference_gfpgan(img_path: str, resize: ResizeType):
    return run_task(
        model="_gfpgan",
        img_path=img_path,
        task="restore",
        resize=resize,
    )



def inference_gpen(img_path: str, task: TaskType):
    return run_task(
        model="_gpen",
        img_path=img_path,
        task=task,
        resize="on",
    )

def get_processed_files(n: int = 8):
    """
    Return latest processed files.
    """

    files = sorted(
        [p for p in OUTPUT_DIR.iterdir() if p.is_file()],
        key=lambda p: p.stat().st_mtime,
        reverse=True,
    )

    return [str(file) for file in files[:n]]

with gr.Blocks(title="GFPGAN") as GFPGAN_app:
    gr.Markdown("# GFPGAN Face Restoration")

    with gr.Row():
        input_img = gr.Image(
            type="filepath",
            label="Input Image",
        )

        output_img = gr.Image(
            type="filepath",
            label="Output Image",
```

```
        )

    with gr.Row():
        resize_radio = gr.Radio(
            choices=["on", "off"],
            value="on",
            label="Resize",
        )

        submit_btn = gr.Button(
            "Restore Face",
            variant="primary",
        )

        get_btn = gr.Button("Recent Files")

    prev_files = gr.Gallery(
        label="Recent Outputs",
        columns=4,
        rows=2,
        object_fit="contain",
        height="auto",
        allow_preview=True,
    )

    submit_btn.click(
        inference_gfpgan,
        inputs=[input_img, resize_radio],
        outputs=[output_img],
    )

    get_btn.click(
        get_processed_files,
        outputs=[prev_files],
    )

with gr.Blocks(title="GPEN") as GPEN_app:
    gr.Markdown("# GPEN Face Enhancement")

    with gr.Row():
        input_img = gr.Image(
            type="filepath",
            label="Input Image",
        )

        output_img = gr.Image(
            type="filepath",
            label="Output Image",
        )

    with gr.Row():
        task_radio = gr.Radio(
            choices=["restore", "colorize"],
            value="restore",
            label="Task",
        )
```

```
        submit_btn = gr.Button(
            "Run GPEN",
            variant="primary",
        )

        get_btn = gr.Button("Recent Files")

    prev_files = gr.Gallery(
        label="Recent Outputs",
        columns=4,
        rows=2,
        object_fit="contain",
        height="auto",
        allow_preview=True,
    )

    submit_btn.click(
        inference_gpen,
        inputs=[input_img, task_radio],
        outputs=[output_img],
    )

    get_btn.click(
        get_processed_files,
        outputs=[prev_files],
    )

app = gr.TabbedInterface(
    [GFPGAN_app, GPEN_app],
    ["GFPGAN", "GPEN"],
)


if __name__ == "__main__":
    logger.info("Starting Gradio application")

    app.queue(
        default_concurrency_limit=CONCURRENCY_LIMIT,
    )

    app.launch(
        server_name="0.0.0.0",
        show_error=True,
    )
```

? Qwen3.7

Click to show code

```
from subprocess import Popen, PIPE
import gradio as gr
import cv2
```

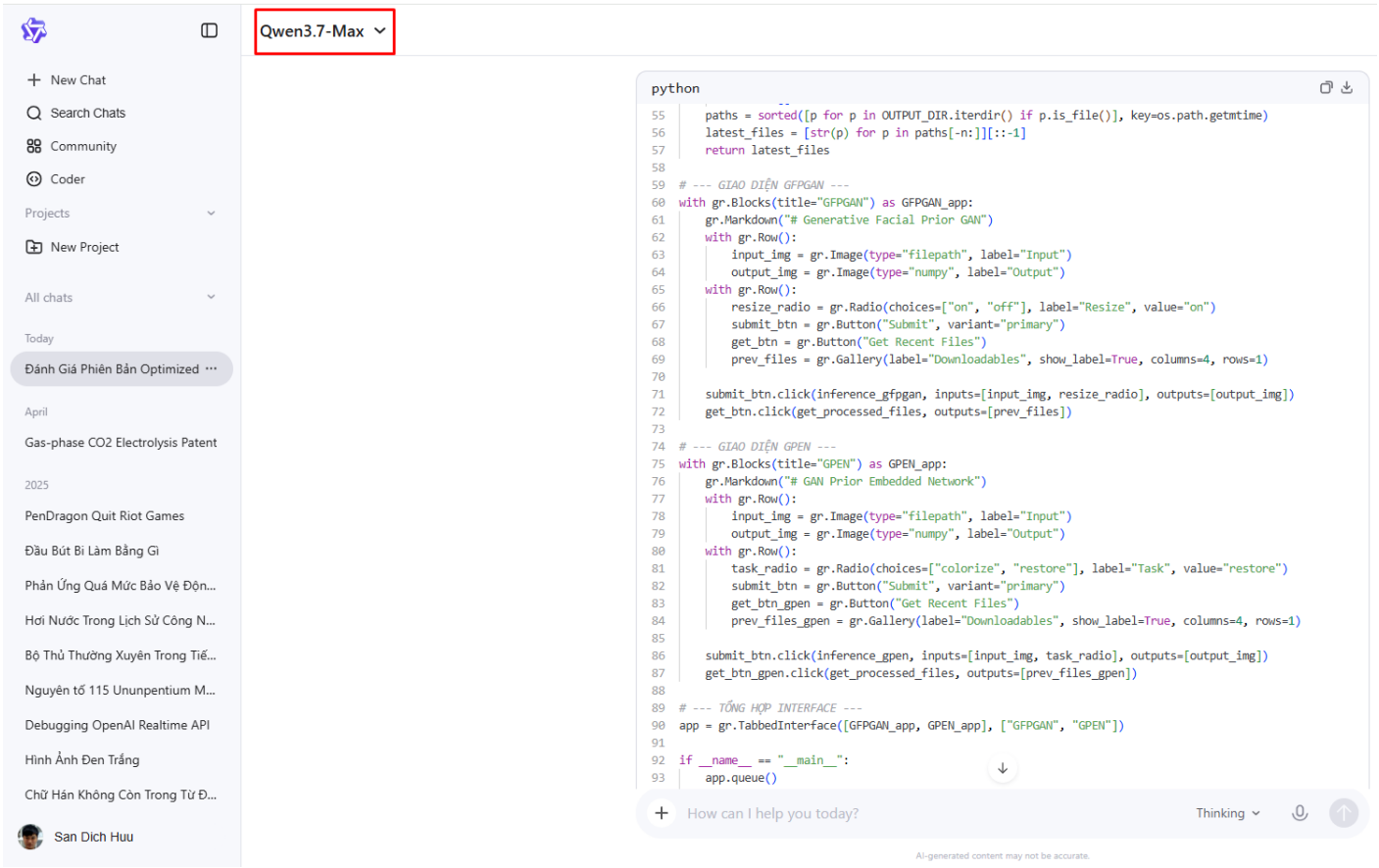
```
import os
from pathlib import Path

OUTPUT_DIR = Path("outputs")
OUTPUT_DIR.mkdir(exist_ok=True)

def run_task(model: str, img_path: str, task: str = "restore", resize: str = "on"):
    img_path_obj = Path(img_path)
    basename = img_path_obj.stem
    ext = img_path_obj.suffix

    model_clean = model.strip("_ ")
    print(f"Started {model_clean.upper()} | Task: {task} | Image: {img_path_obj.name}...")

    p = Popen(
        ["python", f"app/{model}.py", str(img_path_obj), task, resize],
        stdout=PIPE,
        stderr=PIPE
    )
    stdout, stderr = p.communicate()

    print("--- STDOUT ---")
    print(stdout.decode("utf-8", errors="ignore"))
    print("--- STDERR ---")
    print(stderr.decode("utf-8", errors="ignore"))

    extension = ext.lstrip(".")
    restored_img_path = OUTPUT_DIR / f"{basename}{model}_{task}.{extension}"

    if restored_img_path.is_file():
        restored_img = cv2.imread(str(restored_img_path))
        if restored_img is None:
            raise gr.Error(f"Không th? ??c ?nh: {restored_img_path}")
        restored_img = cv2.cvtColor(restored_img, cv2.COLOR_BGR2RGB)
        print(f"Finished {model_clean.upper()} -> {restored_img_path}")
        return restored_img
    else:
        raise gr.Error(f"Không tìm th?y k?t qu? t?i {restored_img_path}!")

def inference_gfpgan(img_path, resize):
    return run_task("_gfpgan", img_path, task="restore", resize=resize)

def inference_gpen(img_path, task):
    return run_task("_gpen", img_path, task=task, resize="on")

def get_processed_files(n=4):
    if not OUTPUT_DIR.exists():
        return []
    paths = sorted([p for p in OUTPUT_DIR.iterdir() if p.is_file()], key=os.path.getmtime)
    latest_files = [str(p) for p in paths[-n:]][::-1]
    return latest_files

with gr.Blocks(title="GFPGAN") as GFPGAN_app:
    gr.Markdown("# Generative Facial Prior GAN")
    with gr.Row():
        input_img = gr.Image(type="filepath", label="Input")
        output_img = gr.Image(type="numpy", label="Output")
```

```
    with gr.Row():
        resize_radio = gr.Radio(choices=["on", "off"], label="Resize", value="on")
        submit_btn = gr.Button("Submit", variant="primary")
        get_btn = gr.Button("Get Recent Files")
        prev_files = gr.Gallery(label="Downloadables", show_label=True, columns=4, rows=1)

    submit_btn.click(inference_gfpgan, inputs=[input_img, resize_radio],
outputs=[output_img])
    get_btn.click(get_processed_files, outputs=[prev_files])

with gr.Blocks(title="GPEN") as GPEN_app:
    gr.Markdown("# GAN Prior Embedded Network")
    with gr.Row():
        input_img = gr.Image(type="filepath", label="Input")
        output_img = gr.Image(type="numpy", label="Output")
    with gr.Row():
        task_radio = gr.Radio(choices=["colorize", "restore"], label="Task",
value="restore")
        submit_btn = gr.Button("Submit", variant="primary")
        get_btn_gpen = gr.Button("Get Recent Files")
        prev_files_gpen = gr.Gallery(label="Downloadables", show_label=True, columns=4,
rows=1)

    submit_btn.click(inference_gpen, inputs=[input_img, task_radio], outputs=[output_img])
    get_btn_gpen.click(get_processed_files, outputs=[prev_files_gpen])

app = gr.TabbedInterface([GFPGAN_app, GPEN_app], ["GFPGAN", "GPEN"])

if __name__ == "__main__":
    app.queue()
    app.launch(server_name="0.0.0.0")
```

? Summary

> In my opinion
> **Qwen3.7** is better than others in coding task, especially Python language
> GPT5 insert tripple quote markdown, this is wrong coding rule.
> Gemini looking good.

?? Screenshot



Revision #15
Created 2026-05-27 11:21:59 UTC by sandichhuu
Updated 2026-05-27 11:56:27 UTC by sandichhuu