معماری و پایپلاین پردازش تصویر حرفه‌ای با FastAPI

۱. بررسی کلی مسئله و رویکرد

هدف ساخت یک سرویس پردازش تصویر است که تصویر ورودی را دریافت کرده و مراحل زیر را روی آن اجرا کند:

1. **بهبود کیفیت تصویر** (Image Enhancement / Super-Resolution)
2. **حذف نویز** (Denoising)
3. **حذف سایه‌های اضافه** (Shadow Removal)
4. **حذف پس‌زمینه** (Background Removal) — با حفظ جزئیات ریز مثل پره‌های دوچرخه

نکته کلیدی: ترتیب اجرای مراحل اهمیت زیادی دارد. بهترین ترتیب:

```
ورودی → حذف نویز → حذف سایه → حذف پس‌زمینه → بهبود کیفیت (Super-Resolution) →
```

دلیل این ترتیب:

- اول نویز حذف می‌شود تا مراحل بعدی روی تصویر تمیزتری کار کنند[1][2]
- سایه قبل از حذف پس‌زمینه برداشته می‌شود تا edge detection بهتری داشته باشیم
- Super-Resolution در آخر اعمال می‌شود تا روی تصویر نهایی (که ممکن است کوچک‌تر شده) کیفیت را بالا ببرد

---

۲. انتخاب تکنولوژی‌ها برای هر مرحله

۲.۱. حذف نویز (Denoising): NAFNet

NAFNet (Nonlinear Activation Free Network) از Megvii Research، بهترین انتخاب برای denoising است:[3]

- **PSNR: 40.30 dB** روی دیتاست SIDD — بالاترین نتیجه SOTA[3]
- محاسبات بسیار کمتر نسبت به رقبا (کمتر از نصف هزینه محاسباتی)[3]
- هم denoising و هم deblurring را پشتیبانی می‌کند[4]
- ساده است و با PIL Image استفاده می‌شود[4]

```
# استفاده ساده:
from nafnetlib import DenoiseProcessor
processor = DenoiseProcessor(model_id="sidd_width64", device="cuda")
denoised = processor.process(image)
```

جایگزین: Restormer — عملکرد نزدیک ولی سنگین‌تر[5]

۲.۲. حذف سایه (Shadow Removal): ShadowFormer + OpenCV Hybrid

دو رویکرد ترکیبی پیشنهاد می‌شود:

رویکرد اول (سبک‌تر): OpenCV Multi-Scale Retinex

- از Multi-Scale Retinex (MSR) برای illumination normalization استفاده می‌کند[2]
- Shadow masking در فضاهای رنگی HSV و LAB[2]
- مناسب برای سایه‌های ساده و عملیاتی سبک[2]
- بدون نیاز به GPU

رویکرد دوم (دقیق‌تر): ShadowFormer

- برای حذف سایه استفاده global context که از Transformer-based مدل می‌کند[6]
- برای مدل‌سازی همبستگی بین مناطق Shadow-Interaction Module (SIM) سایه‌دار و بدون سایه[6][7]
- عملکرد SOTA روی دیتاست‌های ISTD، ISTD+، و SRD[6]
- تا ۱۵۰ برابر پارامتر کمتر از رقبا[6]

به عنوان ShadowFormer به عنوان پیش‌فرض استفاده شود و OpenCV **پیشنهاد:** از premium گزینه فعال باشد.

۲.۳. حذف پس‌زمینه (Background Removal): BiRefNet

BiRefNet (Bilateral Reference Network) بهترین انتخاب برای حذف پس‌زمینه با جزئیات بالاست:[8][9][10]

چرا BiRefNet؟

- open-source: IoU = 0.87, Dice = 0.92[9] **بالاترین دقت** بین تمام مدل‌های
- **جزئیات ریز:** پره‌های دوچرخه، ساقه گل، مو، و ساختارهای نازک را بی‌نقص جدا می‌کند[10]
- پردازش دوطرفه (bidirectional): هم از global به local و هم برعکس، تا جزئیات ریز با ساختار کلی هماهنگ باشند[9]
- پشتیبانی از رزولوشن بالا: مدل‌های 1024×1024 تا 2048×2048[8]
- مدل Matting جداگانه برای لبه‌های نرم و شفاف[8]
- قابل بارگذاری از HuggingFace در یک خط[8]

```
from transformers import AutoModelForImageSegmentation
birefnet = AutoModelForImageSegmentation.from_pretrained(
    'zhengpeng7/BiRefNet', trust_remote_code=True
)
```

مقایسه با رقبا:

مدل	IoU (DIS5K)	Dice	جزئیات ریز	سرعت
BiRefNet	**0.87**	**0.92**	عالی [10]	~95ms (4090) [8]
IS-Net	0.82	0.89	خوب [9]	~351ms
U2-Net	0.39	0.52	ضعیف [9]	~307ms
rembg (U2netp)	~37.5% acc	—	ضعیف [11]	سریع
RMBG-2.0 (BRIA)	بسیار بالا	بسیار بالا	خوب [12]	متوسط

> ⚠ **RMBG-2.0** از BRIA هم روی معماری BiRefNet ساخته شده ولی لایسنس MIT اصلی لایسنس BiRefNet دارد. آن برای استفاده تجاری نیاز به قرارداد دارد [8][13]

ترکیب **BiRefNet + ViTMatte برای بهبود بیشتر لبه‌ها:**

- ابتدا BiRefNet ماسک اولیه را تولید می‌کند
- سپس ViTMatte با استفاده از trimap خودکار، لبه‌های نرم و شفاف تولید می‌کند[14][15]
- این ترکیب مشابه پایپلاین Matte Anything است[15]

۲.۴. بهبود کیفیت (Super-Resolution): Real-ESRGAN

برای افزایش رزولوشن و بهبود جزئیات:**Real-ESRGAN** [16][17]

- آموزش با داده‌های سینتتیک واقع‌گرایانه[17]
- حذف artifact[17] ها ضمن افزایش جزئیات
- پشتیبانی از scale 2x 4 وx[17]
- نصب ساده و inference سریع[17]

```
from RealESRGAN import RealESRGAN
model = RealESRGAN(device, scale=4)
model.load_weights('weights/RealESRGAN_x4.pth', download=True)
sr_image = model.predict(image)
```

---

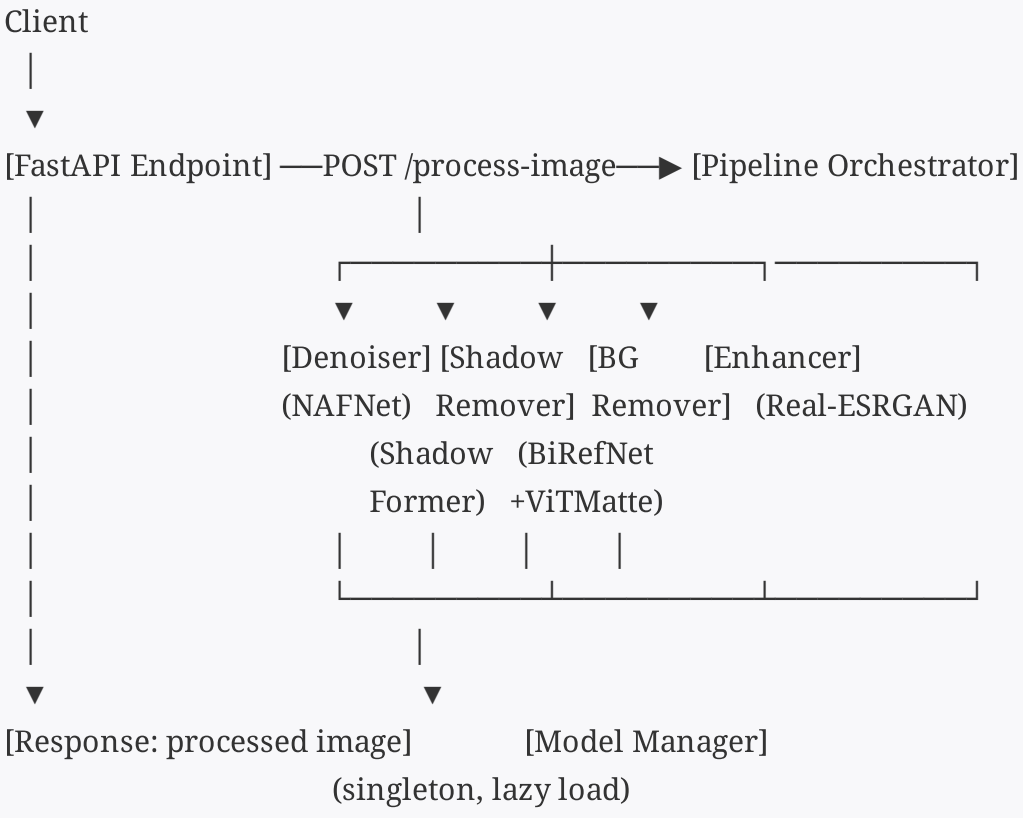
۳. معماری سیستم (FastAPI)

۳.۱. ساختار پروژه

```
image-processor/
├── app/
│   ├── main.py              # FastAPI app + lifespan
│   ├── config.py            # تنظیمات (مدل‌ها، مسیرها، GPU)
│   ├── api/
│   │   ├── __init__.py
│   │   ├── router.py        # API endpoints
│   │   └── schemas.py       # Pydantic models (request/response)
│   ├── core/
│   │   ├── __init__.py
│   │   ├── pipeline.py      # پایپلاین اصلی پردازش
│   │   └── model_manager.py # مدیریت بارگذاری مدل‌ها
│   ├── processors/
│   │   ├── __init__.py
│   │   ├── base.py          # BaseProcessor (abstract)
│   │   ├── denoiser.py      # NAFNet denoiser
│   │   ├── shadow_remover.py # ShadowFormer / OpenCV
│   │   ├── bg_remover.py    # BiRefNet + ViTMatte
│   │   └── enhancer.py      # Real-ESRGAN
│   ├── utils/
│   │   ├── __init__.py
│   │   ├── image_utils.py   # تبدیل فرمت، resize، ...
│   │   └── file_utils.py    # ذخیره و خواندن فایل
│   └── workers/
│       └── celery_worker.py # (اختیاری) برای پردازش async
├── models/                  # وزن‌های مدل‌ها
```

```
├── tests/
│   ├── test_denoiser.py
│   ├── test_shadow_remover.py
│   ├── test_bg_remover.py
│   └── test_pipeline.py
├── Dockerfile
├── docker-compose.yml
├── requirements.txt
└── README.md
```

۳.۲. معماری نرم‌افزار



۳.۳. الگوهای طراحی کلیدی

Strategy Pattern برای پردازشگرها:

```python
from abc import ABC, abstractmethod
from PIL import Image
```

```python
class BaseProcessor(ABC):
    @abstractmethod
    def load_model(self) -> None: ...

    @abstractmethod
    def process(self, image: Image.Image, **kwargs) -> Image.Image: ...

    @abstractmethod
    def cleanup(self) -> None: ...
```

Pipeline Pattern برای ترکیب مراحل:

```python
class ImagePipeline:
    def __init__(self, steps: list[BaseProcessor]):
        self.steps = steps

    async def execute(self, image: Image.Image, config: dict) -> Image.Image:
        for step in self.steps:
            if config.get(step.name, {}).get("enabled", True):
                image = step.process(image, **config.get(step.name, {}))
        return image
```

Singleton Model Manager برای مدیریت حافظه GPU:

```python
class ModelManager:
    _instance = None
    _models: dict = {}

    @classmethod
    def get_model(cls, name: str):
        if name not in cls._models:
            cls._models[name] = cls._load_model(name)
        return cls._models[name]
```

۳.۴. FastAPI Endpoint طراحی

```
from fastapi import FastAPI, UploadFile, File
from pydantic import BaseModel

class ProcessingConfig(BaseModel):
    denoise: bool = True
    remove_shadow: bool = True
    remove_background: bool = True
    enhance: bool = True
    enhance_scale: int = 2  # 2x or 4x
    output_format: str = "png"  # png, webp

@app.post("/api/v1/process")
async def process_image(
    file: UploadFile = File(...),
    config: ProcessingConfig = ProcessingConfig()
):
    ...
```

۳.۵. نکات Production

- **Worker ها:** استفاده از Gunicorn + Uvicorn workers، تعداد worker برابر تعداد هسته CPU[18]
- **پردازش سنگین:** استفاده از background task یا Celery برای تصاویر بزرگ[18]
- **GPU Memory:** بارگذاری lazy مدل‌ها و آزادسازی حافظه بعد از استفاده
- **Rate Limiting:** محدودیت درخواست برای جلوگیری از overload[18]
- **Health Check:** endpoint /health برای monitoring[18]
- **CORS:** تنظیم صحیح برای اتصال فرانت‌اند[18]
- **Streaming Response:** برای تصاویر بزرگ[19]

---

۴. تقسیم‌بندی به تسک‌های مستقل

فاز ۱: زیرساخت (Foundation)

#	تسک	شرح	وابستگی
T1	**Project Setup**	ساختار پروژه، requirements.txt، Docker setup	—
T2	**FastAPI Skeleton**	main.py، router، schemas، health endpoint	T1
T3	**BaseProcessor**	کلاس abstract و Pipeline orchestrator	T1
T4	**Image Utils**	توابع تبدیل فرمت، resize، validate	T1
T5	**Model Manager**	سیستم بارگذاری و مدیریت مدل‌ها	T3

فاز ۲: پردازشگرها (Processors) — قابل توسعه موازی

#	تسک	شرح	وابستگی
T6	**Denoiser (NAFNet)**	پیاده‌سازی DenoiserProcessor	T3, T5
T7	**Shadow Remover (OpenCV)**	پیاده‌سازی ساده با Retinex	T3
T8	**Shadow Remover (ShadowFormer)**	پیاده‌سازی پیشرفته	T3, T5
T9	**BG Remover (BiRefNet)**	حذف پس‌زمینه با BiRefNet	T3, T5
T10	**BG Refiner (ViTMatte)**	بهبود لبه‌ها با matting	T9
T11	**Enhancer (Real-ESRGAN)**	افزایش کیفیت و رزولوشن	T3, T5

فاز ۳: یکپارچه‌سازی (Integration)

#	تسک	شرح	وابستگی
T12	**Pipeline Assembly**	اتصال پردازشگرها به pipeline	T6-T11
T13	**API Integration**	اتصال pipeline به FastAPI endpoints	T2, T12
T14	**Error Handling**	مدیریت خطا، logging، retry	T13
T15	**Config System**	تنظیمات قابل تغییر (env vars, yaml)	T2

فاز ۴: بهینه‌سازی و استقرار

#	تسک	شرح	وابستگی
T16	**Testing**	unit tests + integration tests	T12-T14
T17	**Docker Build**	Dockerfile بهینه با CUDA	T13
T18	**Async Processing**	Celery/background tasks برای تصاویر بزرگ	T13
T19	**GPU Optimization**	FP16 inference, batch processing	T12
T20	**Documentation**	API docs (Swagger), README	T13

---

۵. Requirements اصلی

```
# requirements.txt
fastapi>=0.115.0
uvicorn[standard]>=0.34.0
python-multipart>=0.0.18
Pillow>=10.4.0
torch>=2.5.0
torchvision>=0.20.0
transformers>=4.45.0      # برای BiRefNet
onnxruntime-gpu>=1.19.0   # برای - اختیاری inference سریع‌تر
numpy>=1.26.0
opencv-python>=4.10.0
pydantic>=2.9.0

# مدل‌های خاص
# Real-ESRGAN: pip install git+https://github.com/sberbank-ai/Real-ESRGAN.g
```

```
# NAFNet: از مخزن اصلی clone شود
# withoutbg (Focus model): pip install withoutbg  — جایگزین اختیاری برای BiRefN
```

---

۶. نقشه اجرایی پیشنهادی

```
هفته ۱: T1 → T2 → T3 → T4 → T5          (زیرساخت)
هفته ۲: T9 → T10 (BiRefNet - مهم‌ترین)  (حذف پس‌زمینه)
هفته ۳: T6 + T7 (موازی)                 (denoising + shadow ساده)
هفته ۴: T11 + T8 (موازی)                (enhancer + shadow پیشرفته)
هفته ۵: T12 → T13 → T14 → T15           (یکپارچه‌سازی)
هفته ۶: T16 → T17 → T18 → T19 → T20      (بهینه‌سازی و استقرار)
```

---

۷. نکات مهم فنی

GPU Memory Management

BiRefNet 3.5حدود GB حافظه GPU نیاز دارد (FP16). Real-ESRGAN -2 هم حدود 3GB. اگر GPU [8] محدود دارید:

- از FP16 inference [8] استفاده کنید
- مدل‌ها را به صورت lazy load کنید و بعد از استفاده آزاد کنید
- TensorRT conversion سرعت را تا ۱۰ برابر افزایش می‌دهد (150~ms به 11~ms)[8]

حفظ جزئیات دوچرخه

ساختارهای نازک مثل **پره‌های دوچرخه**، ساقه segmentation مخصوصاً در BiRefNet گل، و حصار عملکرد فوق‌العاده دارد. برای بهترین نتیجه:[10]

- از مدل BiRefNet_HR (2048×2048) [8] استفاده کنید
- گزینه refine_foreground [8] را فعال کنید
- در صورت نیاز به لبه‌های نرم‌تر، ViTMatte را به pipeline [14] اضافه کنید

Shadow Removal برای محصولات

drop shadow برای عکس محصولات (مثل دوچرخه) سایه‌ها معمولاً هستند. رویکرد OpenCV Retinex برای اکثر موارد کافی است. ShadowFormer را برای سایه‌های پیچیده‌تر نگه دارید.[2][6]

References

1. OCT-GAN: single step shadow and noise removal from optical ... - We developed a single process that successfully removed both noise and retinal shadows from unseen s...
2. Enhancing Images: Adaptive Shadow Correction Using OpenCV - Remove shadows from images using OpenCV and Python with an adaptive Retinex-based approach and real-...
3. megvii-research/NAFNet: The state-of-the-art image restoration ... - We derive a Nonlinear Activation Free Network, namely NAFNet, from the baseline. SOTA results are ac...
4. mikestealth/nafnet-models - Hugging Face - This repository contains implementations of the NAFNet (Nonlinear Activation Free Network) for image...
5. The Tenth NTIRE 2025 Image Denoising Challenge Report - arXiv - This competition seeks to foster innovative solutions, establish performance benchmarks, and explore...
6. ShadowFormer: Global Context Helps Image Shadow Removal - arXiv - Recent deep learning methods have achieved promising results in image shadow removal. However, most ...
7. ShadowFormer: Global Context Helps Image Shadow Removal - Liner - This research aims to develop an end-to-end shadow removal model that exploits global context to ens...
8. ZhengPeng7/BiRefNet: [CAAI AIR'24] Bilateral Reference for High ... - Feb 1, 2025 : We released the BiRefNet_HR for general use, which was trained on images in 2048x2048 ...
9. Evaluating image segmentation models for background removal for ... - BiRefNet (Bilateral Reference Network): Specifically designed to segment complex and high-resolution...
10. Introduction to BiRefNet - DebuggerCafe - BiRefNet is a segmentation model for high-resolution dichotomous image segmentation based on the Swi...

11. Which background removal tools to use and why? - Velebit AI - rembg is an open source background removal project that offers 3 different models for removing the b...
12. briaai/RMBG-2.0 - Hugging Face - RMBG v2.0 is our new state-of-the-art background removal model significantly improves RMBG v1.4. The...
13. pinokiofactory/RMBG-2-Studio: Enhanced background remove and ... - Background Removal: Powered by BRIA-RMBG-2.0; Drag-and-Drop Gallery: View your processed images and ...
14. ViTMatte: A Leap Forward in Image Matting with Vision Transformers - ViTMatte is pioneering in leveraging ViTs for image matting with a streamlined adaptation. ViTMatte ...
15. Interactive Natural Image Matting with Segment Anything Model - arXiv - We propose Matte Anything model (MatAny), an interactive natural image matting model that could prod...
16. Real-ESRGAN: Practical Algorithms for General Image/Video ... - Real-ESRGAN aims at developing Practical Algorithms for General Image/Video Restoration. We extend t...
17. taylor-s-amarel/Real-ESRGAN-2025 - GitHub - PyTorch implementation of a Real-ESRGAN model trained on custom dataset. This model shows better res...
18. FastAPI production deployment best practices - Render - Learn FastAPI production deployment with ASGI servers, async optimization, security middleware, JWT ...
19. FastAPI Mistakes That Kill Your Performance - Use FastAPI CLI production mode instead of development settings ... Some good use cases are include ...