📋 대본 → 이미지 자동 생성 프로그램 PRD (역프롬프팅)

1. 프로젝트 개요

1.1 목적

1만자 한글 대본을 입력하면 최대 100장의 이미지를 **오류 없이** 자동 생성하는 파이썬 프로그램 개발

1.2 핵심 목표

- **입력**: 1만자 이내 한글 대본 (.txt)
- **출력**: 최대 100장 고품질 이미지 (.png)
- **안정성**: 중간 실패 시에도 나머지 계속 진행, 실패분만 재시도

---

2. 기술 스택 & API Rate Limits

2.1 사용 모델

용도	모델명	역할
텍스트 분석	`gemini-2.5-flash`	대본 분석 → 장면 분할 → 프롬프트 생성
이미지 생성	`gemini-2.5-flash-image`	이미지 프롬프트 → 이미지 생성

2.2 Tier 1 Rate Limits (공식 문서 기준)

Gemini 2.5 Flash (텍스트)

항목	제한
RPM (분당 요청)	1,000
TPM (분당 토큰)	1,000,000
RPD (일일 요청)	10,000

Gemini 2.5 Flash Image (이미지 생성)

항목	제한
RPM (분당 요청)	500
TPM (분당 토큰)	500,000
RPD (일일 요청)	2,000

2.3 100장 생성 시 Rate Limit 분석

✅ RPD 2,000 > 100장 → 충분
✅ RPM 500 → 100장 생성 시 약 12초 내 완료 가능 (이론상)
⚠️ 안전을 위해 요청 간 0.2초 딜레이 권장 → 약 20초 소요

2.4 가격

- 이미지당: **$0.039** (1,290 output tokens)
- 100장 생성 비용: **약 $3.90** (약 5,200원)

---

3. API 공식 문서

3.1 Rate Limits 문서

URL: https://ai.google.dev/gemini-api/docs/rate-limits

Tier 1 주요 내용

```
| Model                | RPM   | TPM       | RPD    |
|----------------------|-------|-----------|--------|
| Gemini 2.5 Flash       | 1,000 | 1,000,000 | 10,000 |
| Gemini 2.5 Flash Image  | 500   | 500,000   | 2,000  |
```

Tier 업그레이드 조건

- **Free → Tier 1**: Billing 계정 연결
- **Tier 1 → Tier 2**: $250 이상 누적 지출 + 30일 경과
- **Tier 2 → Tier 3**: $1,000 이상 누적 지출 + 30일 경과

3.2 이미지 생성 API 문서

URL: https://ai.google.dev/gemini-api/docs/image-generation

지원 화면 비율 (10종)

```
1:1, 3:2, 2:3, 3:4, 4:3, 4:5, 5:4, 9:16, 16:9, 21:9
```

기본 이미지 생성 코드 예시

```python
from google import genai
from google.genai import types
from PIL import Image

client = genai.Client()

response = client.models.generate_content(
    model="gemini-2.5-flash-image",
    contents="이미지 프롬프트",
    config=types.GenerateContentConfig(
        response_modalities=["IMAGE"],
        image_config=types.ImageConfig(
            aspect_ratio="16:9",  # 비율 선택
        )
    )
)

for part in response.parts:
    if part.inline_data is not None:
        generated_image = part.as_image()
        generated_image.save("output.png")
```

---

4. 필수 기능 명세

4.1 화면 비율 선택

```python
```

```
ASPECT_RATIOS = {
    "가로형 (16:9)": "16:9",   # YouTube, 영상용
    "정사각형 (1:1)": "1:1",   # Instagram, 썸네일
    "세로형 (9:16)": "9:16",   # TikTok, Reels, Shorts
}
```

4.2 화풍 선택

python

```
STYLES = {
    "실사": {
        "en": "photorealistic",
        "prompt_prefix": "Photorealistic, ultra-detailed, professional photography style, "
    },
    "3D": {
        "en": "3D render",
        "prompt_prefix": "3D rendered, Pixar-style animation, smooth shading, high-quality CGI, "
    },
    "애니메이션": {
        "en": "animation",
        "prompt_prefix": "2D animation style, anime-inspired, vibrant colors, cel-shaded, "
    }
}
```

---

5. 핵심 요구사항 (프롬프트 엔지니어링)

5.1 한국인 캐릭터 (한복 금지)

python

```
CHARACTER_CONSTRAINTS = """
CRITICAL RULES FOR ALL CHARACTERS:
- All characters must be Korean/East Asian ethnicity
- Characters must wear MODERN clothing (casual, business, etc.)
- NEVER use traditional Korean clothing (hanbok) - this is STRICTLY PROHIBITED
- Clothing examples: t-shirts, jeans, hoodies, suits, dresses, skirts
"""
```

5.2 캐릭터 일관성 유지

캐릭터 프로필 생성 시스템

```python
CHARACTER_PROFILE_PROMPT = """
Based on the script, create detailed character profiles:

For EACH character (main and supporting), define:
1. NAME: Character name
2. ROLE: Main/Supporting
3. GENDER: Male/Female
4. AGE: Specific age (e.g., 28)
5. PHYSICAL FEATURES:
   - Face shape: (oval/round/square/heart)
   - Hair: color, length, style (e.g., "black, shoulder-length, straight")
   - Eyes: color, shape
   - Skin tone: (fair/medium/tan)
   - Height: (tall/average/short)
   - Build: (slim/average/athletic/heavy)
6. OUTFIT (MODERN - NO HANBOK):
   - Top: specific description
   - Bottom: specific description
   - Shoes: type
   - Accessories: if any
7. UNIQUE IDENTIFIERS:
   - Distinctive features (mole, scar, dimples, etc.)
   - Signature item or accessory

Output as JSON format.
"""
```

프롬프트에 캐릭터 정보 주입

```python
```

```
def build_scene_prompt(scene_desc, characters_in_scene, character_profiles, style):
    """각 장면 프롬프트에 캐릭터 정보를 일관되게 주입"""

    character_descriptions = []
    for char_name in characters_in_scene:
        profile = character_profiles[char_name]
        char_desc = f"""
        [{char_name}]:
        - {profile['age']} year old {profile['gender']}
        - Face: {profile['face_shape']}, {profile['eyes']}, {profile['skin_tone']}
        - Hair: {profile['hair']}
        - Wearing: {profile['outfit']['top']}, {profile['outfit']['bottom']}, {profile['outfit']['shoes']}
        - Distinctive feature: {profile['unique_identifiers']}
        """
        character_descriptions.append(char_desc)

    final_prompt = f"""
    {style['prompt_prefix']}

    SCENE: {scene_desc}

    CHARACTERS IN THIS SCENE (maintain EXACT appearance):
    {''.join(character_descriptions)}

    CRITICAL:
    - Maintain EXACT same appearance for each character across ALL scenes
    - Use "this exact character" phrasing for consistency
    - Characters must wear MODERN clothing only (NO traditional Korean hanbok)
    - All characters are Korean ethnicity
    """
    return final_prompt
```

---

6. 안정성 & 에러 핸들링

6.1 핵심 원칙

❌ 1장 실패 → 전체 중단 (BAD)
✅ 1장 실패 → 나머지 계속 진행 → 실패분만 재시도 (GOOD)

6.2 에러 핸들링 로직

python

```python
import time
import asyncio
from dataclasses import dataclass
from enum import Enum
from typing import Optional

class ImageStatus(Enum):
    PENDING = "pending"
    SUCCESS = "success"
    FAILED = "failed"
    RETRYING = "retrying"

@dataclass
class ImageTask:
    scene_id: int
    prompt: str
    status: ImageStatus = ImageStatus.PENDING
    image_path: Optional[str] = None
    error_message: Optional[str] = None
    retry_count: int = 0

class RobustImageGenerator:
    MAX_RETRIES = 3
    RETRY_DELAYS = [2, 5, 10]  # 지수 백오프
    REQUEST_DELAY = 0.2  # Rate limit 대비 요청 간 딜레이

    def __init__(self, api_key: str):
        self.client = genai.Client(api_key=api_key)
        self.tasks: list[ImageTask] = []
        self.results = {
            "success": [],
            "failed": []
        }

    async def generate_single_image(self, task: ImageTask, config: dict) -> bool:
        """단일 이미지 생성 (실패 시 False 반환)"""
        try:
            response = self.client.models.generate_content(
                model="gemini-2.5-flash-image",
                contents=task.prompt,
                config=types.GenerateContentConfig(
                    response_modalities=["IMAGE"],
                    image_config=types.ImageConfig(
                        aspect_ratio=config["aspect_ratio"],
                    )
                )
```

```
            )

            for part in response.parts:
                if part.inline_data is not None:
                    image = part.as_image()
                    image_path = f"output/scene_{task.scene_id:03d}.png"
                    image.save(image_path)
                    task.image_path = image_path
                    task.status = ImageStatus.SUCCESS
                    return True

            raise Exception("No image data in response")

        except Exception as e:
            task.error_message = str(e)
            task.status = ImageStatus.FAILED
            return False

    async def generate_all_with_retry(self, tasks: list[ImageTask], config: dict):
        """모든 이미지 생성 (실패 시 재시도)"""

        # 1차 시도: 모든 태스크 실행
        print(f"🚀 1차 생성 시작: {len(tasks)}장")
        for i, task in enumerate(tasks):
            print(f"  [{i+1}/{len(tasks)}] Scene {task.scene_id} 생성 중...")
            success = await self.generate_single_image(task, config)

            if success:
                self.results["success"].append(task)
                print(f"    ✅ 성공")
            else:
                print(f"    ❌ 실패: {task.error_message}")

            await asyncio.sleep(self.REQUEST_DELAY)  # Rate limit 대비

        # 실패한 태스크 수집
        failed_tasks = [t for t in tasks if t.status == ImageStatus.FAILED]

        # 재시도 루프
        for retry_round in range(self.MAX_RETRIES):
            if not failed_tasks:
                break

            delay = self.RETRY_DELAYS[retry_round]
            print(f"\n⏳ {delay}초 대기 후 재시도 ({retry_round + 1}/{self.MAX_RETRIES})...")
            await asyncio.sleep(delay)
```

```
        print(f"🔄 재시도: {len(failed_tasks)}장")
        still_failed = []

        for task in failed_tasks:
            task.retry_count += 1
            task.status = ImageStatus.RETRYING

            success = await self.generate_single_image(task, config)

            if success:
                self.results["success"].append(task)
                print(f"  ✅ Scene {task.scene_id} 재시도 성공")
            else:
                still_failed.append(task)
                print(f"  ❌ Scene {task.scene_id} 재시도 실패")

            await asyncio.sleep(self.REQUEST_DELAY)

        failed_tasks = still_failed

    # 최종 실패 목록
    self.results["failed"] = failed_tasks

    return self.results

def print_summary(self):
    """결과 요약 출력"""
    total = len(self.results["success"]) + len(self.results["failed"])
    success_count = len(self.results["success"])
    failed_count = len(self.results["failed"])

    print("\n" + "="*50)
    print("📊 생성 결과 요약")
    print("="*50)
    print(f"✅ 성공: {success_count}/{total}장")
    print(f"❌ 실패: {failed_count}/{total}장")

    if self.results["failed"]:
        print("\n실패한 장면:")
        for task in self.results["failed"]:
            print(f"  - Scene {task.scene_id}: {task.error_message}")
```

6.3 Rate Limit 에러 처리

python

```python
def handle_rate_limit_error(error: Exception, task: ImageTask):
    """429 Rate Limit 에러 특별 처리"""
    if "429" in str(error) or "quota" in str(error).lower():
        # Rate limit 초과 시 더 긴 대기
        print(f"⚠️ Rate limit 도달. 60초 대기...")
        time.sleep(60)
        return True  # 재시도 허용
    return False
```

---

7. 전체 워크플로우

7.1 처리 단계

```
[입력] 1만자 대본 (.txt)
        ↓
[1단계] 대본 분석 (gemini-2.5-flash)
    - 장면 분할 (최대 100개)
    - 캐릭터 추출 및 프로필 생성
    - 각 장면별 이미지 프롬프트 생성
        ↓
[2단계] 이미지 생성 (gemini-2.5-flash-image)
    - 캐릭터 일관성 유지 프롬프트 적용
    - 선택된 화면 비율/화풍 적용
    - 실패 시 자동 재시도
        ↓
[출력] 100장 이미지 + 결과 리포트
```

7.2 메인 코드 구조

python

```python
import os
import json
import asyncio
from pathlib import Path
from google import genai
from google.genai import types

# ============================================
# 설정
# ============================================
class Config:
    # API
    API_KEY = os.getenv("GEMINI_API_KEY")

    # 모델
    TEXT_MODEL = "gemini-2.5-flash"
    IMAGE_MODEL = "gemini-2.5-flash-image"

    # 화면 비율
    ASPECT_RATIOS = {
        "1": ("16:9", "가로형 (YouTube, 영상)"),
        "2": ("1:1", "정사각형 (Instagram)"),
        "3": ("9:16", "세로형 (TikTok, Shorts)"),
    }

    # 화풍
    STYLES = {
        "1": {
            "name": "실사",
            "prefix": "Photorealistic, ultra-detailed, professional photography, "
        },
        "2": {
            "name": "3D",
            "prefix": "3D rendered, Pixar-style, smooth CGI, high-quality animation, "
        },
        "3": {
            "name": "애니메이션",
            "prefix": "2D animation, anime-inspired, vibrant colors, cel-shaded, "
        },
    }

    # 제약조건
    CONSTRAINTS = """
    CRITICAL RULES:
    - All characters: Korean/East Asian ethnicity
    - Clothing: MODERN only (NO traditional Korean hanbok - STRICTLY PROHIBITED)
```

```
    - Maintain character consistency across all scenes
    """

# ============================================
# 1단계: 대본 분석
# ============================================
class ScriptAnalyzer:
    def __init__(self, client: genai.Client):
        self.client = client

    def analyze_script(self, script_text: str, max_scenes: int = 100) -> dict:
        """대본 분석 → 장면 분할 + 캐릭터 프로필"""

        prompt = f"""
        You are a professional script analyzer for image generation.

        Analyze the following Korean script and extract:

        1. CHARACTERS: Create detailed profiles for ALL characters (main and supporting)
           - Name
           - Role (main/supporting)
           - Age (specific number)
           - Gender
           - Physical appearance:
             * Face shape, skin tone
             * Hair (color, length, style)
             * Eye color/shape
           - Outfit (MODERN clothing only - NO hanbok):
             * Top, Bottom, Shoes
             * Accessories
           - Distinctive features (mole, scar, etc.)

        2. SCENES: Divide into {max_scenes} or fewer visual scenes
           For each scene:
           - scene_id: Sequential number (1, 2, 3...)
           - description: What's happening (visual description)
           - characters: List of character names in this scene
           - setting: Location/environment
           - mood: Emotional tone
           - camera_angle: Suggested shot type (close-up, wide, etc.)

        {Config.CONSTRAINTS}

        SCRIPT:
        {script_text}

        OUTPUT FORMAT (JSON):
```

```
    {{
      "characters": {{
        "character_name": {{
          "role": "main/supporting",
          "age": 28,
          "gender": "male/female",
          "face_shape": "oval",
          "skin_tone": "fair",
          "hair": "black, short, neat",
          "eyes": "dark brown, almond-shaped",
          "outfit": {{
            "top": "white button-up shirt",
            "bottom": "navy slacks",
            "shoes": "brown leather shoes",
            "accessories": "silver watch"
          }},
          "distinctive_features": "small mole near left eye"
        }}
      }},
      "scenes": [
        {{
          "scene_id": 1,
          "description": "...",
          "characters": ["character_name"],
          "setting": "...",
          "mood": "...",
          "camera_angle": "..."
        }}
      ]
    }}
    """

    response = self.client.models.generate_content(
      model=Config.TEXT_MODEL,
      contents=prompt
    )

    # JSON 파싱
    result_text = response.text
    # JSON 블록 추출
    if "```json" in result_text:
      result_text = result_text.split("```json")[1].split("```")[0]

    return json.loads(result_text)

# ==========================================
# 2단계: 프롬프트 생성
```

```python
# ============================================
class PromptBuilder:
    def __init__(self, style_config: dict, aspect_ratio: str):
        self.style_prefix = style_config["prefix"]
        self.aspect_ratio = aspect_ratio

    def build_prompt(self, scene: dict, characters: dict) -> str:
        """장면 + 캐릭터 정보 → 이미지 프롬프트"""

        # 등장 캐릭터 설명 구성
        char_descriptions = []
        for char_name in scene.get("characters", []):
            if char_name in characters:
                char = characters[char_name]
                desc = f"""
                [{char_name}] - this exact character:
                - {char['age']} year old {char['gender']}
                - Face: {char['face_shape']}, {char['skin_tone']} skin
                - Hair: {char['hair']}
                - Eyes: {char['eyes']}
                - Wearing: {char['outfit']['top']}, {char['outfit']['bottom']}, {char['outfit']['shoes']}
                - Distinctive: {char.get('distinctive_features', 'none')}
                """
                char_descriptions.append(desc)

        prompt = f"""
        {self.style_prefix}

        SCENE: {scene['description']}
        SETTING: {scene.get('setting', '')}
        MOOD: {scene.get('mood', '')}
        CAMERA: {scene.get('camera_angle', 'medium shot')}

        CHARACTERS (maintain EXACT same appearance):
        {''.join(char_descriptions)}

        STRICT RULES:
        - Korean/East Asian characters only
        - MODERN clothing only (absolutely NO traditional Korean hanbok)
        - Maintain perfect character consistency
        - High quality, detailed image
        """

        return prompt

# ============================================
# 3단계: 이미지 생성
```

```python
# ========================================
class ImageGenerator:
    MAX_RETRIES = 3
    RETRY_DELAYS = [2, 5, 10]
    REQUEST_DELAY = 0.2

    def __init__(self, client: genai.Client, aspect_ratio: str):
        self.client = client
        self.aspect_ratio = aspect_ratio
        self.results = {"success": [], "failed": []}

    async def generate_image(self, scene_id: int, prompt: str) -> tuple[bool, str]:
        """단일 이미지 생성"""
        try:
            response = self.client.models.generate_content(
                model=Config.IMAGE_MODEL,
                contents=prompt,
                config=types.GenerateContentConfig(
                    response_modalities=["IMAGE"],
                    image_config=types.ImageConfig(
                        aspect_ratio=self.aspect_ratio,
                    )
                )
            )

            for part in response.parts:
                if part.inline_data is not None:
                    image = part.as_image()
                    path = f"output/scene_{scene_id:03d}.png"
                    image.save(path)
                    return True, path

            return False, "No image data"

        except Exception as e:
            return False, str(e)

    async def generate_all(self, prompts: list[tuple[int, str]]) -> dict:
        """전체 이미지 생성 (재시도 포함)"""

        pending = list(prompts)

        # 1차 시도
        print(f"\n🚀 이미지 생성 시작: {len(pending)}장")
        failed = []

        for i, (scene_id, prompt) in enumerate(pending):
```

```
            print(f"  [{i+1}/{len(pending)}] Scene {scene_id}...", end=" ")
            success, result = await self.generate_image(scene_id, prompt)

            if success:
                self.results["success"].append((scene_id, result))
                print("✅")
            else:
                failed.append((scene_id, prompt, result))
                print(f"❌ ({result[:50]}...)")

            await asyncio.sleep(self.REQUEST_DELAY)

        # 재시도
        for retry in range(self.MAX_RETRIES):
            if not failed:
                break

            delay = self.RETRY_DELAYS[retry]
            print(f"\n⏳ {delay}초 후 재시도 ({retry+1}/{self.MAX_RETRIES})...")
            await asyncio.sleep(delay)

            still_failed = []
            for scene_id, prompt, _ in failed:
                print(f"  🔄 Scene {scene_id}...", end=" ")
                success, result = await self.generate_image(scene_id, prompt)

                if success:
                    self.results["success"].append((scene_id, result))
                    print("✅")
                else:
                    still_failed.append((scene_id, prompt, result))
                    print(f"❌")

                await asyncio.sleep(self.REQUEST_DELAY)

            failed = still_failed

        self.results["failed"] = [(sid, err) for sid, _, err in failed]
        return self.results

# ============================================
# 메인 실행
# ============================================
async def main():
    print("="*60)
    print("🎬 대본 → 이미지 자동 생성 프로그램")
    print("="*60)
```

```
# 1. 사용자 입력
print("\n📁 대본 파일 경로를 입력하세요:")
script_path = input("> ").strip()

print("\n🖼️  화면 비율 선택:")
for key, (ratio, desc) in Config.ASPECT_RATIOS.items():
    print(f"  {key}. {desc} ({ratio})")
ratio_choice = input("> ").strip()
aspect_ratio = Config.ASPECT_RATIOS.get(ratio_choice, ("16:9", ""))[0]

print("\n🎨 화풍 선택:")
for key, style in Config.STYLES.items():
    print(f"  {key}. {style['name']}")
style_choice = input("> ").strip()
style_config = Config.STYLES.get(style_choice, Config.STYLES["1"])

# 2. 대본 로드
with open(script_path, "r", encoding="utf-8") as f:
    script_text = f.read()

print(f"\n✅ 대본 로드 완료: {len(script_text)}자")

# 3. 초기화
client = genai.Client(api_key=Config.API_KEY)
os.makedirs("output", exist_ok=True)

# 4. 대본 분석
print("\n🔍 대본 분석 중...")
analyzer = ScriptAnalyzer(client)
analysis = analyzer.analyze_script(script_text)

characters = analysis["characters"]
scenes = analysis["scenes"]

print(f"  - 캐릭터 {len(characters)}명 추출")
print(f"  - 장면 {len(scenes)}개 분할")

# 5. 프롬프트 생성
print("\n📝 이미지 프롬프트 생성 중...")
builder = PromptBuilder(style_config, aspect_ratio)
prompts = []

for scene in scenes:
    prompt = builder.build_prompt(scene, characters)
    prompts.append((scene["scene_id"], prompt))
```

```
    # 6. 이미지 생성
    generator = ImageGenerator(client, aspect_ratio)
    results = await generator.generate_all(prompts)

    # 7. 결과 출력
    print("\n" + "="*60)
    print("📊 최종 결과")
    print("="*60)
    print(f"✅ 성공: {len(results['success'])}장")
    print(f"❌ 실패: {len(results['failed'])}장")

    if results["failed"]:
        print("\n실패한 장면:")
        for scene_id, error in results["failed"]:
            print(f"  - Scene {scene_id}: {error[:100]}")

    # 결과 저장
    with open("output/result.json", "w", encoding="utf-8") as f:
        json.dump({
            "analysis": analysis,
            "results": {
                "success": results["success"],
                "failed": results["failed"]
            }
        }, f, ensure_ascii=False, indent=2)

    print(f"\n📁 결과 저장: output/result.json")
    print("🎉 완료!")

if __name__ == "__main__":
    asyncio.run(main())
```

---

8. 설치 및 실행

8.1 필요 패키지

```
bash

pip install google-genai pillow
```

8.2 환경 변수 설정

```
bash
```

```
export GEMINI_API_KEY="your-api-key-here"
```

8.3 실행

```
bash

python script_to_image.py
```

---

9. 폴더 구조

```
project/
├── script_to_image.py      # 메인 프로그램
├── input/
│   └── script.txt          # 입력 대본
├── output/
│   ├── scene_001.png       # 생성된 이미지
│   ├── scene_002.png
│   ├── ...
│   └── result.json         # 결과 리포트
└── requirements.txt        # 의존성
```

---

10. 체크리스트

구현 완료 확인

- [ ] 대본 분석 (장면 분할)
- [ ] 캐릭터 프로필 자동 생성
- [ ] 화면 비율 선택 (16:9, 1:1, 9:16)
- [ ] 화풍 선택 (실사, 3D, 애니메이션)
- [ ] 한국인 캐릭터 + 한복 금지 적용
- [ ] 캐릭터 일관성 유지
- [ ] 100장 생성 시 안정성 (재시도 로직)
- [ ] Rate Limit 대응 (요청 간 딜레이)
- [ ] 결과 리포트 생성

---

11. 참고 문서

문서	URL
Rate Limits	https://ai.google.dev/gemini-api/docs/rate-limits
Image Generation	https://ai.google.dev/gemini-api/docs/image-generation
Pricing	https://ai.google.dev/gemini-api/docs/pricing
Models	https://ai.google.dev/gemini-api/docs/models
프롬프트 가이드	https://developers.googleblog.com/en/how-to-prompt-gemini-2-5-flash-image-generation-for-the-best-results/