Nice Diffusion 인페인팅 통합 가이드

스테이블 디퓨전과 컴피유아이의 inpaint 기능을 Nice Diffusion 프로젝트에 완전히 통합하는 실용적인 구현 가이드입니다. 실시간 프리뷰를 최우선으로 하는 단계별 접근법을 제시합니다.

1. 마스크 그리기 시스템: HTML5 Canvas 기반 실시간 편집

핵심 아키텍처: 멀티 레이어 캔버스 시스템

실시간 성능과 메모리 효율성을 위해 **3개의 분리된 캔버스 레이어**를 사용합니다: GitHub +7

javascript

```javascript
// canvas_manager.js - 확장된 캔버스 관리자
class MaskEditorCanvas {
  constructor(containerId, options = {}) {
    this.container = document.getElementById(containerId);
    this.options = {
      width: options.width || 768,
      height: options.height || 768,
      modelType: options.modelType || 'SD15'
    };

    this.setupCanvasLayers();
    this.initializeDrawingSystem();
    this.setupEventHandlers();
  }

  setupCanvasLayers() {
    // 배경 레이어 (원본 이미지)
    this.backgroundCanvas = this.createCanvas('background-layer');
    this.backgroundCtx = this.backgroundCanvas.getContext('2d');

    // 마스크 레이어 (편집 가능한 마스크)
    this.maskCanvas = this.createCanvas('mask-layer');
    this.maskCtx = this.maskCanvas.getContext('2d', { alpha: false });

    // 프리뷰 레이어 (실시간 미리보기)
    this.previewCanvas = this.createCanvas('preview-layer');
    this.previewCtx = this.previewCanvas.getContext('2d');

    // 레이어 스타일 설정
    this.maskCanvas.style.opacity = '0.7';
    this.previewCanvas.style.pointerEvents = 'auto';
  }

  createCanvas(id) {
    const canvas = document.createElement('canvas');
    canvas.id = id;
    canvas.width = this.options.width;
    canvas.height = this.options.height;
    canvas.style.position = 'absolute';
    canvas.style.top = '0';
    canvas.style.left = '0';
    this.container.appendChild(canvas);
    return canvas;
  }
}
```

고급 브러시 도구 시스템

압력 감지와 스무딩을 지원하는 브러시 구현:

javascript

```javascript
class AdvancedBrushSystem {
  constructor(canvas) {
    this.canvas = canvas;
    this.ctx = canvas.getContext('2d');
    this.isDrawing = false;
    this.lastPoint = null;
    this.brushSettings = {
      size: 20,
      opacity: 0.8,
      hardness: 0.5,
      flow: 1.0
    };
    this.undoStack = [];
    this.redoStack = [];
  }

  // 스무딩된 브러시 스트로크
  drawSmoothStroke(currentPoint, pressure = 1.0) {
    if (!this.lastPoint) {
      this.lastPoint = currentPoint;
      return;
    }

    const distance = this.calculateDistance(this.lastPoint, currentPoint);
    const steps = Math.max(1, Math.floor(distance / 2));

    for (let i = 0; i <= steps; i++) {
      const t = i / steps;
      const interpolated = this.interpolatePoints(this.lastPoint, currentPoint, t);

      this.drawBrushDab(
        interpolated.x,
        interpolated.y,
        this.brushSettings.size * pressure,
        this.brushSettings.opacity * pressure
      );
    }

    this.lastPoint = currentPoint;
  }

  // 압력 감지 브러시 점
  drawBrushDab(x, y, size, opacity) {
    this.ctx.save();
    this.ctx.globalAlpha = opacity;
    this.ctx.globalCompositeOperation = 'source-over';

    // 부드러운 가장자리를 위한 radial gradient
```

```
    // 부드러운 가장자리를 위한 radial gradient
    const gradient = this.ctx.createRadialGradient(
      x, y, 0,
      x, y, size / 2
    );
    gradient.addColorStop(0, `rgba(255, 255, 255, ${this.brushSettings.hardness})`);
    gradient.addColorStop(1, 'rgba(255, 255, 255, 0)');

    this.ctx.fillStyle = gradient;
    this.ctx.beginPath();
    this.ctx.arc(x, y, size / 2, 0, Math.PI * 2);
    this.ctx.fill();
    this.ctx.restore();
  }

  // 실시간 브러시 프리뷰
  showBrushPreview(x, y) {
    const previewCanvas = document.getElementById('brush-preview');
    const previewCtx = previewCanvas.getContext('2d');

    previewCtx.clearRect(0, 0, previewCanvas.width, previewCanvas.height);
    previewCtx.strokeStyle = 'rgba(255, 255, 255, 0.8)';
    previewCtx.lineWidth = 2;
    previewCtx.beginPath();
    previewCtx.arc(x, y, this.brushSettings.size / 2, 0, Math.PI * 2);
    previewCtx.stroke();
  }
}
```

실시간 마스크 불투명도 및 시각화

javascript

```javascript
class MaskVisualizationSystem {
  constructor(maskCanvas) {
    this.maskCanvas = maskCanvas;
    this.visualizationMode = 'red-overlay'; // 'red-overlay', 'checkerboard', 'outline'
    this.opacity = 0.7;
  }

  updateMaskVisualization(opacity, mode) {
    this.opacity = opacity;
    this.visualizationMode = mode;

    switch(mode) {
      case 'red-overlay':
        this.maskCanvas.style.opacity = opacity;
        this.maskCanvas.style.mixBlendMode = 'multiply';
        break;
      case 'checkerboard':
        this.applyCheckerboardPattern();
        break;
      case 'outline':
        this.showMaskOutline();
        break;
    }
  }

  applyCheckerboardPattern() {
    const ctx = this.maskCanvas.getContext('2d');
    const imageData = ctx.getImageData(0, 0, this.maskCanvas.width, this.maskCanvas.height);
    const data = imageData.data;

    for (let i = 0; i < data.length; i += 4) {
      const x = (i / 4) % this.maskCanvas.width;
      const y = Math.floor((i / 4) / this.maskCanvas.width);
      const checker = ((Math.floor(x / 8) + Math.floor(y / 8)) % 2) * 255;

      if (data[i + 3] > 0) { // 마스크 영역
        data[i] = checker;     // R
        data[i + 1] = checker; // G
        data[i + 2] = checker; // B
      }
    }

    ctx.putImageData(imageData, 0, 0);
  }
}
```

2. Python-JavaScript 브릿지: 효율적인 데이터 전송

Base64 인코딩 최적화 시스템

압축과 배치 처리를 지원하는 마스크 전송:

javascript

```javascript
// 프론트엔드: 마스크 데이터 추출 및 전송
class MaskDataTransmitter {
  constructor(canvas, compressionLevel = 0.8) {
    this.canvas = canvas;
    this.compressionLevel = compressionLevel;
    this.transmissionQueue = [];
    this.isTransmitting = false;
  }

  // 압축된 마스크 데이터 추출
  extractMaskData(format = 'png') {
    try {
      // 큰 마스크의 경우 JPEG 압축 사용
      if (this.canvas.width * this.canvas.height > 1000000) {
        format = 'jpeg';
      }

      const dataURL = this.canvas.toDataURL(`image/${format}`, this.compressionLevel);
      const base64String = dataURL.replace(/^data:image\/[a-z]+;base64,/, '');

      return {
        data: base64String,
        format: format,
        width: this.canvas.width,
        height: this.canvas.height,
        timestamp: Date.now(),
        compressed: format === 'jpeg'
      };
    } catch (error) {
      console.error('마스크 데이터 추출 실패:', error);
      return null;
    }
  }

  // 실시간 전송 관리
  async sendMaskToBackend(callback = null) {
    if (this.isTransmitting) return;

    this.isTransmitting = true;
    const maskData = this.extractMaskData();

    if (!maskData) {
      this.isTransmitting = false;
      return;
    }

    try {
      // AI 모델을 통한 백엔드 전송
```

```
            // NiceGUI를 통한 백엔드 전송
            const response = await window.pywebview.api.process_mask_data(maskData);

            if (callback) callback(response);

        } catch (error) {
            console.error('마스크 전송 실패:', error);
            // 실패한 데이터를 큐에 저장
            this.transmissionQueue.push(maskData);
        } finally {
            this.isTransmitting = false;
        }
    }
}
```

Python 백엔드: 견고한 데이터 수신 시스템

python

```python
# canvas_editor.py - 확장된 마스크 처리
import base64
import io
import asyncio
from PIL import Image
import numpy as np
from typing import Dict, Any, Optional

class MaskProcessor:
    def __init__(self):
        self.current_mask = None
        self.mask_history = []
        self.processing_queue = asyncio.Queue()

    async def process_mask_data(self, mask_payload: Dict[str, Any]) -> Dict[str, Any]:
        """마스크 데이터를 처리하고 결과 반환"""
        try:
            # Base64 디코딩
            mask_data = base64.b64decode(mask_payload['data'])
            mask_image = Image.open(io.BytesIO(mask_data))

            # 모델 타입에 따른 크기 조정
            target_size = self.get_target_size(mask_payload.get('model_type', 'SD15'))
            if mask_image.size != target_size:
                mask_image = mask_image.resize(target_size, Image.LANCZOS)

            # 그레이스케일 변환
            if mask_image.mode != 'L':
                mask_image = mask_image.convert('L')

            # 마스크 전처리
            processed_mask = self.preprocess_mask(
                mask_image,
                blur_factor=mask_payload.get('blur_factor', 0),
                expand_pixels=mask_payload.get('expand_pixels', 0)
            )

            # 현재 마스크 저장
            self.current_mask = processed_mask
            self.mask_history.append(processed_mask)

            # 히스토리 크기 제한
            if len(self.mask_history) > 20:
                self.mask_history.pop(0)

            return {
                'success': True,
```

```
            'mask_id': len(self.mask_history) - 1,
            'dimensions': processed_mask.size,
            'timestamp': mask_payload['timestamp']
        }

    except Exception as e:
        return {
            'success': False,
            'error': str(e),
            'timestamp': mask_payload.get('timestamp')
        }

def get_target_size(self, model_type: str) -> tuple:
    """모델 타입에 따른 타겟 크기 반환"""
    if model_type.upper() == 'SDXL':
        return (1024, 1024)
    return (512, 512)

def preprocess_mask(self, mask: Image.Image, blur_factor: int = 0,
                    expand_pixels: int = 0) -> Image.Image:
    """마스크 전처리"""
    mask_array = np.array(mask)

    # 마스크 확장
    if expand_pixels > 0:
        import cv2
        kernel = np.ones((expand_pixels*2+1, expand_pixels*2+1), np.uint8)
        mask_array = cv2.dilate(mask_array, kernel, iterations=1)

    # 블러 적용
    if blur_factor > 0:
        import cv2
        mask_array = cv2.GaussianBlur(mask_array, (blur_factor*2+1, blur_factor*2+1), 0)

    return Image.fromarray(mask_array)
```

NiceGUI 통합을 위한 실시간 통신

python

```python
# 메인 애플리케이션 통합
from nicegui import ui
import asyncio

class InpaintUI:
    def __init__(self):
        self.mask_processor = MaskProcessor()
        self.current_image = None
        self.current_mask = None
        self.setup_ui()

    def setup_ui(self):
        with ui.row().classes('w-full'):
            # 캔버스 영역
            with ui.column().classes('flex-1'):
                ui.label('마스크 편집기').classes('text-xl font-bold')

                # 캔버스 컨테이너
                self.canvas_container = ui.html('''
                    <div id="mask-editor-container" style="position: relative; width: 768px; height: 768px;">
                        <canvas id="background-layer"></canvas>
                        <canvas id="mask-layer"></canvas>
                        <canvas id="preview-layer"></canvas>
                        <canvas id="brush-preview"></canvas>
                    </div>
                ''')

                # 컨트롤 버튼들
                with ui.row():
                    ui.button('마스크 지우기', on_click=self.clear_mask)
                    ui.button('실행 취소', on_click=self.undo_mask)
                    ui.button('인페인트 실행', on_click=self.execute_inpainting)

            # 파라미터 패널
            with ui.column().classes('w-80'):
                ui.label('인페인트 설정').classes('text-lg font-bold')

                self.brush_size = ui.slider(
                    min=1, max=100, value=20,
                    on_change=self.update_brush_size
                ).props('label="브러시 크기"')

                self.mask_opacity = ui.slider(
                    min=0, max=1, value=0.7, step=0.1,
                    on_change=self.update_mask_opacity
                ).props('label="마스크 불투명도"')
```

```
            self.strength = ui.slider(
                min=0, max=1, value=0.8, step=0.1
            ).props('label="인페인트 강도"')

            self.guidance_scale = ui.slider(
                min=1, max=20, value=7.5, step=0.5
            ).props('label="가이던스 스케일"')

    async def process_mask_update(self, mask_data):
        """JavaScript에서 호출되는 마스크 업데이트 처리"""
        result = await self.mask_processor.process_mask_data(mask_data)

        if result['success']:
            # 실시간 프리뷰 업데이트
            await self.update_preview()

        return result

    async def update_preview(self):
        """실시간 프리뷰 업데이트"""
        if self.current_image and self.mask_processor.current_mask:
            # 빠른 프리뷰 생성 (실제 인페인트 아님)
            await ui.run_javascript('''
                updateMaskPreview();
            ''')
```

3. 백엔드 인페인트 파이프라인: 고성능 처리 시스템

메모리 최적화된 파이프라인 매니저

python

```python
import torch
from diffusers import AutoPipelineForInpainting
from typing import Dict, Any, Optional
import gc

class OptimizedInpaintPipeline:
    def __init__(self, model_path: str, device: str = "cuda"):
        self.device = device
        self.model_path = model_path
        self.pipeline = None
        self.current_model_type = None
        self.memory_optimization_enabled = True

    def load_pipeline(self, model_type: str = "SD15"):
        """모델 타입에 따른 파이프라인 로드"""
        if self.pipeline is None or self.current_model_type != model_type:
            # 기존 파이프라인 정리
            if self.pipeline is not None:
                del self.pipeline
                torch.cuda.empty_cache()
                gc.collect()

            # 모델 경로 결정
            if model_type.upper() == "SDXL":
                model_path = "diffusers/stable-diffusion-xl-1.0-inpainting-0.1"
            else:
                model_path = "runwayml/stable-diffusion-inpainting"

            # 파이프라인 생성
            self.pipeline = AutoPipelineForInpainting.from_pretrained(
                model_path,
                torch_dtype=torch.float16,
                variant="fp16",
                use_safetensors=True
            )

            # 메모리 최적화 적용
            self.apply_memory_optimizations()
            self.current_model_type = model_type

    def apply_memory_optimizations(self):
        """메모리 최적화 기법 적용"""
        if self.memory_optimization_enabled:
            # CPU 오프로딩 활성화
            self.pipeline.enable_model_cpu_offload()

            # xFormers 메모리 효율적 attention
```

```
        if torch.cuda.is_available():
            self.pipeline.enable_xformers_memory_efficient_attention()

        # VAE 슬라이싱 (대용량 배치용)
        self.pipeline.enable_vae_slicing()

        # VAE 타일링 (대용량 이미지용)
        self.pipeline.enable_vae_tiling()

    async def inpaint_async(self, image: Image.Image, mask: Image.Image,
                            prompt: str, **kwargs) -> Image.Image:
        """비동기 인페인트 처리"""
        try:
            # 모델 타입 감지 및 로드
            model_type = kwargs.get('model_type', 'SD15')
            self.load_pipeline(model_type)

            # 입력 전처리
            processed_image, processed_mask = self.preprocess_inputs(
                image, mask, model_type
            )

            # 파라미터 최적화
            optimized_kwargs = self.optimize_parameters(kwargs, model_type)

            # 인페인트 실행
            with torch.cuda.device(self.device):
                result = self.pipeline(
                    prompt=prompt,
                    image=processed_image,
                    mask_image=processed_mask,
                    **optimized_kwargs
                )

            return result.images[0]

        except torch.cuda.OutOfMemoryError:
            # OOM 처리
            torch.cuda.empty_cache()
            # 파라미터 축소 후 재시도
            kwargs['num_inference_steps'] = max(kwargs.get('num_inference_steps', 20) // 2, 10)
            return await self.inpaint_async(image, mask, prompt, **kwargs)

    def preprocess_inputs(self, image: Image.Image, mask: Image.Image,
                          model_type: str) -> tuple:
        """입력 이미지 전처리"""
        target_size = (1024, 1024) if model_type.upper() == 'SDXL' else (512, 512)

        # 비율 유지하면서 크기 조정
        image = image.resize(target_size, Image.LANCZOS)
```

```
        mask = mask.resize(target_size, Image.LANCZOS)

        return image, mask

    def optimize_parameters(self, kwargs: Dict[str, Any], model_type: str) -> Dict[str, Any]:
        """모델 타입에 따른 파라미터 최적화"""
        optimized = kwargs.copy()

        if model_type.upper() == 'SDXL':
            # SDXL 최적화
            optimized['guidance_scale'] = max(optimized.get('guidance_scale', 7.5), 8.0)
            optimized['strength'] = min(optimized.get('strength', 0.8), 0.99)
        else:
            # SD 1.5 최적화
            optimized['guidance_scale'] = optimized.get('guidance_scale', 7.5)
            optimized['strength'] = optimized.get('strength', 0.8)

        return optimized
```

StateManager 통합

python

```python
# 기존 StateManager 확장
class EnhancedStateManager:
    def __init__(self):
        self.inpaint_pipeline = None
        self.mask_processor = MaskProcessor()
        self.current_mode = 'generate'  # 'generate' 또는 'inpaint'
        self.inpaint_history = []

    def switch_to_inpaint_mode(self, image: Image.Image):
        """인페인트 모드로 전환"""
        self.current_mode = 'inpaint'
        self.current_image = image

        # 인페인트 파이프라인 초기화
        if self.inpaint_pipeline is None:
            self.inpaint_pipeline = OptimizedInpaintPipeline(
                model_path=self.get_current_model_path()
            )

    async def process_inpaint_request(self, prompt: str, mask_data: Dict[str, Any],
                                      parameters: Dict[str, Any]) -> Image.Image:
        """인페인트 요청 처리"""
        if self.current_mode != 'inpaint':
            raise ValueError("인페인트 모드가 아닙니다")

        # 마스크 처리
        mask_result = await self.mask_processor.process_mask_data(mask_data)
        if not mask_result['success']:
            raise ValueError(f"마스크 처리 실패: {mask_result['error']}")

        # 인페인트 실행
        result = await self.inpaint_pipeline.inpaint_async(
            image=self.current_image,
            mask=self.mask_processor.current_mask,
            prompt=prompt,
            **parameters
        )

        # 결과 저장
        self.inpaint_history.append({
            'original': self.current_image,
            'mask': self.mask_processor.current_mask,
            'result': result,
            'prompt': prompt,
            'parameters': parameters,
            'timestamp': time.time()
        })
```

4. UI/UX 자연스러운 통합: 반응형 인터페이스

모드 전환 시스템

python

```python
# 향상된 UI 컴포넌트
class ResponsiveInpaintUI:
    def __init__(self):
        self.state_manager = EnhancedStateManager()
        self.current_mode = 'generate'
        self.setup_responsive_ui()

    def setup_responsive_ui(self):
        with ui.column().classes('w-full min-h-screen'):
            # 헤더 및 모드 전환
            with ui.row().classes('w-full justify-between items-center p-4'):
                ui.label('Nice Diffusion').classes('text-2xl font-bold')

                # 모드 전환 토글
                self.mode_toggle = ui.toggle(
                    ['생성', '인페인트'],
                    value='생성',
                    on_change=self.handle_mode_change
                ).classes('bg-blue-500')

            # 메인 콘텐츠 영역
            with ui.row().classes('flex-1 gap-4 p-4'):
                # 이미지 편집 영역
                with ui.column().classes('flex-1'):
                    self.setup_image_area()

                # 파라미터 패널
                with ui.column().classes('w-80'):
                    self.setup_parameter_panel()

    def setup_image_area(self):
        """이미지 편집 영역 설정"""
        with ui.card().classes('w-full'):
            # 모드별 헤더
            self.image_header = ui.label('이미지 생성').classes('text-lg font-bold')

            # 캔버스 컨테이너 (반응형)
            self.canvas_container = ui.html('''
                <div id="responsive-canvas-container" class="w-full flex justify-center">
                    <div id="canvas-wrapper" style="position: relative; max-width: 100%;">
                        <canvas id="main-canvas" style="max-width: 100%; height: auto;"></canvas>
                    </div>
                </div>
            ''')

            # 도구 모음
            with ui.row().classes('justify-center gap-2 mt-4'):
```

```
            self.setup_tools()

    def setup_tools(self):
        """도구 모음 설정"""
        # 공통 도구
        ui.button('지우기', on_click=self.clear_canvas).classes('btn-secondary')
        ui.button('실행 취소', on_click=self.undo_action).classes('btn-secondary')

        # 인페인트 전용 도구
        self.inpaint_tools = ui.row().classes('gap-2')
        with self.inpaint_tools:
            ui.button('마스크 반전', on_click=self.invert_mask).classes('btn-secondary')
            ui.button('마스크 저장', on_click=self.save_mask).classes('btn-secondary')

        # 초기에는 숨김
        self.inpaint_tools.visible = False

    def handle_mode_change(self, e):
        """모드 전환 처리"""
        new_mode = 'inpaint' if e.value == '인페인트' else 'generate'

        if new_mode != self.current_mode:
            self.current_mode = new_mode
            self.update_ui_for_mode(new_mode)

    def update_ui_for_mode(self, mode):
        """모드에 따른 UI 업데이트"""
        if mode == 'inpaint':
            self.image_header.text = '인페인트 편집'
            self.inpaint_tools.visible = True

            # 캔버스를 인페인트 모드로 전환
            ui.run_javascript('''
                initializeInpaintMode();
            ''')
        else:
            self.image_header.text = '이미지 생성'
            self.inpaint_tools.visible = False

            # 캔버스를 생성 모드로 전환
            ui.run_javascript('''
                initializeGenerateMode();
            ''')
```

반응형 캔버스 시스템

javascript

```javascript
// 반응형 캔버스 관리
class ResponsiveCanvasManager {
  constructor() {
    this.currentMode = 'generate';
    this.breakpoints = {
      mobile: 768,
      tablet: 1024,
      desktop: 1200
    };

    this.setupResponsiveCanvas();
    this.setupEventListeners();
  }

  setupResponsiveCanvas() {
    this.updateCanvasSize();
    window.addEventListener('resize', () => {
      this.updateCanvasSize();
    });
  }

  updateCanvasSize() {
    const container = document.getElementById('responsive-canvas-container');
    const canvas = document.getElementById('main-canvas');

    if (!container || !canvas) return;

    const containerWidth = container.clientWidth;
    const deviceType = this.getDeviceType(containerWidth);

    // 디바이스 타입에 따른 캔버스 크기 설정
    const canvasSize = this.getCanvasSizeForDevice(deviceType);

    canvas.width = canvasSize.width;
    canvas.height = canvasSize.height;

    // 모바일에서는 터치 이벤트 최적화
    if (deviceType === 'mobile') {
      this.optimizeForTouch();
    }
  }

  getDeviceType(width) {
    if (width < this.breakpoints.mobile) return 'mobile';
    if (width < this.breakpoints.tablet) return 'tablet';
    return 'desktop';
  }
```

```
  getCanvasSizeForDevice(deviceType) {
    switch (deviceType) {
      case 'mobile':
        return { width: 320, height: 320 };
      case 'tablet':
        return { width: 512, height: 512 };
      case 'desktop':
        return { width: 768, height: 768 };
      default:
        return { width: 512, height: 512 };
    }
  }

  optimizeForTouch() {
    const canvas = document.getElementById('main-canvas');

    // 터치 이벤트 최적화
    canvas.addEventListener('touchstart', this.handleTouchStart.bind(this), { passive: false });
    canvas.addEventListener('touchmove', this.handleTouchMove.bind(this), { passive: false });
    canvas.addEventListener('touchend', this.handleTouchEnd.bind(this), { passive: false });
  }

  handleTouchStart(e) {
    e.preventDefault();
    const touch = e.touches[0];
    const rect = e.target.getBoundingClientRect();
    const x = touch.clientX - rect.left;
    const y = touch.clientY - rect.top;

    this.startDrawing(x, y);
  }

  handleTouchMove(e) {
    e.preventDefault();
    const touch = e.touches[0];
    const rect = e.target.getBoundingClientRect();
    const x = touch.clientX - rect.left;
    const y = touch.clientY - rect.top;

    this.continuDrawing(x, y);
  }
}
```

5. 실시간 프리뷰: 최적화된 성능 구현

성능 최적화 핵심 전략

javascript

```javascript
class OptimizedPreviewSystem {
  constructor() {
    this.frameRate = 60;
    this.frameInterval = 1000 / this.frameRate;
    this.lastFrameTime = 0;
    this.animationId = null;

    this.dirtyRegions = [];
    this.needsUpdate = false;

    this.setupPerformanceMonitoring();
    this.startRenderLoop();
  }

  startRenderLoop() {
    const render = (currentTime) => {
      if (currentTime - this.lastFrameTime >= this.frameInterval) {
        if (this.needsUpdate) {
          this.renderFrame();
          this.needsUpdate = false;
        }
        this.lastFrameTime = currentTime;
      }

      this.animationId = requestAnimationFrame(render);
    };

    this.animationId = requestAnimationFrame(render);
  }

  renderFrame() {
    // 더티 영역만 다시 그리기
    if (this.dirtyRegions.length > 0) {
      this.renderDirtyRegions();
      this.dirtyRegions = [];
    }

    // 성능 모니터링
    this.updatePerformanceMetrics();
  }

  renderDirtyRegions() {
    const canvas = document.getElementById('preview-layer');
    const ctx = canvas.getContext('2d');

    this.dirtyRegions.forEach(region => {
      ctx.clearRect(region.x, region.y, region.width, region.height);
```

```
      this.renderRegion(ctx, region);
    });
  }

  markDirty(x, y, width, height) {
    this.dirtyRegions.push({ x, y, width, height });
    this.needsUpdate = true;
  }

  // 배치 업데이트를 위한 디바운싱
  debouncedUpdate = this.debounce(() => {
    this.needsUpdate = true;
  }, 16); // ~60fps

  debounce(func, wait) {
    let timeout;
    return function executedFunction(...args) {
      const later = () => {
        clearTimeout(timeout);
        func(...args);
      };
      clearTimeout(timeout);
      timeout = setTimeout(later, wait);
    };
  }

  setupPerformanceMonitoring() {
    this.performanceMetrics = {
      frameCount: 0,
      averageFrameTime: 0,
      lastPerformanceLog: performance.now()
    };

    // 60프레임마다 성능 로깅
    setInterval(() => {
      this.logPerformanceMetrics();
    }, 1000);
  }

  logPerformanceMetrics() {
    const now = performance.now();
    const timeDiff = now - this.performanceMetrics.lastPerformanceLog;
    const fps = (this.performanceMetrics.frameCount / timeDiff) * 1000;

    if (fps < 30) {
      console.warn(`낮은 FPS 감지: ${fps.toFixed(2)}fps`);
      this.adjustPerformanceSettings();
    }

    this.performanceMetrics.frameCount = 0;
```

```
        this.performanceMetrics.lastPerformanceLog = now;
    }

    adjustPerformanceSettings() {
        // 성능이 낮을 때 자동 조정
        this.frameRate = Math.max(this.frameRate - 5, 30);
        this.frameInterval = 1000 / this.frameRate;

        // 렌더링 품질 조정
        this.enablePerformanceMode();
    }

    enablePerformanceMode() {
        // 성능 모드 활성화
        document.documentElement.style.setProperty('--canvas-quality', 'low');

        // 불필요한 시각 효과 비활성화
        const canvas = document.getElementById('preview-layer');
        if (canvas) {
            canvas.style.imageRendering = 'pixelated';
        }
    }
}
```

메모리 효율적인 마스크 관리

javascript

```javascript
class EfficientMaskManager {
  constructor() {
    this.maskPool = [];
    this.activePool = [];
    this.poolSize = 10;

    this.initializePool();
  }

  initializePool() {
    for (let i = 0; i < this.poolSize; i++) {
      const canvas = document.createElement('canvas');
      canvas.width = 768;
      canvas.height = 768;
      this.maskPool.push(canvas);
    }
  }

  acquireMask() {
    let mask = this.maskPool.pop();
    if (!mask) {
      mask = document.createElement('canvas');
      mask.width = 768;
      mask.height = 768;
    }

    this.activePool.push(mask);
    return mask;
  }

  releaseMask(mask) {
    const index = this.activePool.indexOf(mask);
    if (index > -1) {
      this.activePool.splice(index, 1);

      // 캔버스 초기화
      const ctx = mask.getContext('2d');
      ctx.clearRect(0, 0, mask.width, mask.height);

      // 풀로 반환
      this.maskPool.push(mask);
    }
  }

  // 메모리 사용량 모니터링
  getMemoryUsage() {
    return {
```

```
            poolSize: this.maskPool.length,
            activeCount: this.activePool.length,
            estimatedMemory: (this.maskPool.length + this.activePool.length) * 768 * 768 * 4 // RGBA
        };
    }
}
```

6. 완전한 통합 예제

최종 통합 컴포넌트

python

```python
# main_inpaint_system.py - 완전한 통합 시스템
class CompleteInpaintSystem:
    def __init__(self):
        self.ui = ResponsiveInpaintUI()
        self.state_manager = EnhancedStateManager()
        self.mask_processor = MaskProcessor()
        self.inpaint_pipeline = OptimizedInpaintPipeline("runwayml/stable-diffusion-inpainting")

        self.setup_complete_system()

    def setup_complete_system(self):
        """완전한 시스템 설정"""
        # JavaScript 통합
        ui.add_head_html('''
            <script>
                // 글로벌 시스템 초기화
                window.inpaintSystem = {
                    maskManager: null,
                    previewSystem: null,
                    canvasManager: null
                };

                document.addEventListener('DOMContentLoaded', function() {
                    // 시스템 컴포넌트 초기화
                    window.inpaintSystem.maskManager = new EfficientMaskManager();
                    window.inpaintSystem.previewSystem = new OptimizedPreviewSystem();
                    window.inpaintSystem.canvasManager = new ResponsiveCanvasManager();

                    console.log('Nice Diffusion 인페인트 시스템 초기화 완료');
                });

                // Python 백엔드와의 통신 함수
                async function sendMaskToBackend(maskData) {
                    try {
                        const response = await pywebview.api.process_mask_data(maskData);
                        return response;
                    } catch (error) {
                        console.error('백엔드 통신 오류:', error);
                        return { success: false, error: error.message };
                    }
                }

                // 실시간 인페인트 프리뷰
                async function generateInpaintPreview(prompt, maskData) {
                    const result = await pywebview.api.generate_preview(prompt, maskData);
                    if (result.success) {
                        updatePreviewDisplay(result.preview_url);
                    }
```

```
            }
            return result;
        }
    </script>
''')

# CSS 스타일링
ui.add_head_html('''
    <style>
        #mask-editor-container {
            border: 2px solid #e2e8f0;
            border-radius: 8px;
            background: #f8fafc;
        }

        #mask-editor-container canvas {
            border-radius: 6px;
        }

        .inpaint-tools {
            background: #ffffff;
            padding: 1rem;
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
        }

        .parameter-panel {
            background: #f8fafc;
            padding: 1.5rem;
            border-radius: 8px;
        }

        @media (max-width: 768px) {
            #mask-editor-container {
                width: 100% !important;
                height: auto !important;
            }

            .parameter-panel {
                width: 100% !important;
                margin-top: 1rem;
            }
        }
    </style>
''')

async def process_complete_inpaint(self, prompt: str, negative_prompt: str,
                                   mask_data: dict, parameters: dict):
    """완전한 인페인트 처리 워크플로우"""
    try:
```

```
        # 1. 마스크 처리
        mask_result = await self.mask_processor.process_mask_data(mask_data)
        if not mask_result['success']:
            return mask_result

        # 2. 모드 전환
        if self.state_manager.current_mode != 'inpaint':
            self.state_manager.switch_to_inpaint_mode(self.state_manager.current_image)

        # 3. 인페인트 실행
        result_image = await self.inpaint_pipeline.inpaint_async(
            image=self.state_manager.current_image,
            mask=self.mask_processor.current_mask,
            prompt=prompt,
            negative_prompt=negative_prompt,
            **parameters
        )

        # 4. 결과 저장 및 반환
        result_url = self.save_result_image(result_image)

        return {
            'success': True,
            'result_url': result_url,
            'mask_id': mask_result['mask_id'],
            'parameters': parameters
        }

    except Exception as e:
        return {
            'success': False,
            'error': str(e)
        }

def save_result_image(self, image: Image.Image) -> str:
    """결과 이미지 저장"""
    import tempfile
    import os

    # 임시 파일로 저장
    temp_dir = tempfile.mkdtemp()
    filename = f"inpaint_result_{int(time.time())}.png"
    filepath = os.path.join(temp_dir, filename)

    image.save(filepath)

    # 웹 접근 가능한 URL 반환
    return f"/temp/{filename}"
```

```python
# 메인 애플리케이션 실행
if __name__ == "__main__":
    inpaint_system = CompleteInpaintSystem()
    ui.run(port=8080, title="Nice Diffusion - 인페인트 에디터")
```

핵심 구현 포인트

1. 실시간 프리뷰 최적화

- **멀티레이어 캔버스**: 배경, 마스크, 프리뷰 분리
- **더티 영역 추적**: 변경된 부분만 다시 그리기
- **60fps 렌더링**: requestAnimationFrame 활용
- **메모리 풀링**: 캔버스 객체 재사용

2. 효율적인 데이터 전송

- **압축된 Base64**: 큰 이미지에 JPEG 압축 적용
- **배치 처리**: 여러 마스크 업데이트 묶어서 전송
- **에러 복구**: 실패한 전송 재시도 메커니즘

3. 메모리 최적화

- **모델 CPU 오프로딩**: VRAM 사용량 감소
- **VAE 슬라이싱**: 대용량 이미지 처리
- **동적 파라미터 조정**: OOM 발생 시 자동 조정

4. 반응형 UI

- **디바이스 감지**: 화면 크기에 따른 UI 조정
- **터치 최적화**: 모바일 디바이스 지원
- **모드 전환**: 부드러운 인터페이스 전환

이 가이드는 Nice Diffusion 프로젝트에 완전히 통합 가능한 인페인트 시스템을 제공하며, 실시간 프리뷰를 최우선으로 하는 고성능 구현을 보장합니다.