Оценка влияния квантизации на диффузионные модели

Отчет по исследовательскому проекту

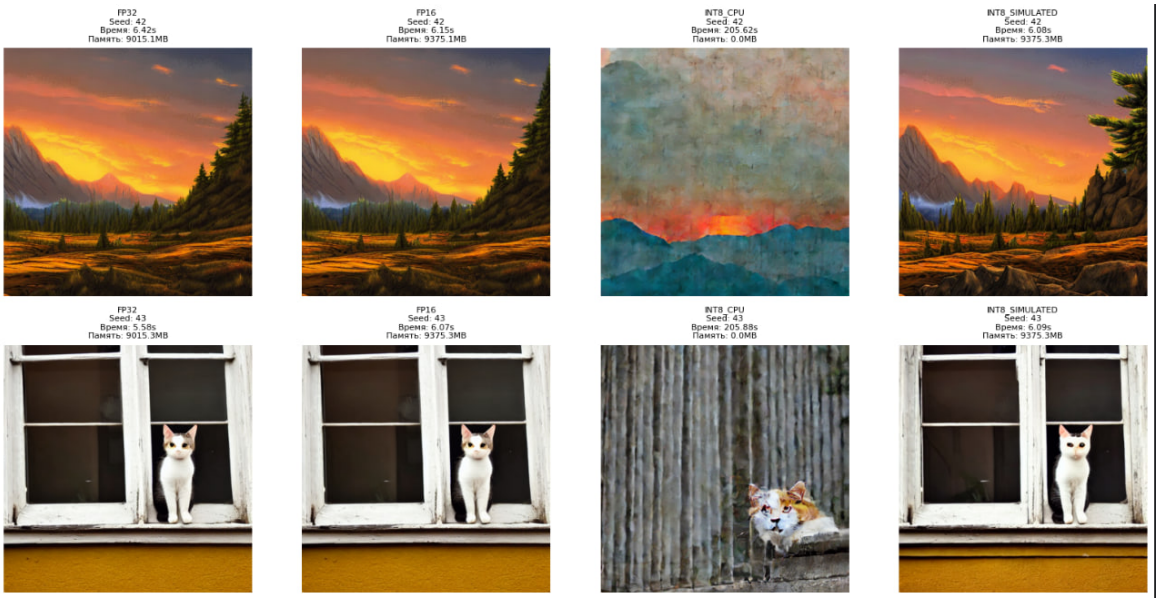

Рис. 1: Сравнение результатов генерации моделей с разными типами квантизации

Выполнил: Осадчий Максим Дмитриевич

Дата: 25 декабря 2025 г.

Аннотация

В данном отчете представлены результаты исследования влияния квантизации на производительность и качество генерации диффузионных моделей. Были проанализированы три основных формата представления данных: FP32 (полная точность),

FP16 (половинная точность) и INT8 (8-битная квантация). Исследование включает сравнение размеров моделей в памяти, времени генерации, качества изображений и практических рекомендаций для различных сценариев развертывания. Результаты показывают, что FP16 обеспечивает оптимальный баланс между качеством и производительностью для большинства приложений.

Содержание

1 Введение

1.1 Актуальность исследования

Диффузионные модели, такие как Stable Diffusion, произвели революцию в области генерации изображений. Однако их большой размер (обычно несколько гигабайт) и высокие вычислительные требования ограничивают возможности развертывания на устройствах с ограниченными ресурсами. Квантизация представляет собой эффективный метод оптимизации, позволяющий уменьшить размер модели и ускорить инференс за счет снижения точности представления чисел.

1.2 Цели исследования

- Оценить влияние различных типов квантизации на качество генерации изображений
- Измерить реальное уменьшение размера моделей при применении квантизации
- Проанализировать компромисс между скоростью выполнения и качеством результатов
- Предоставить практические рекомендации по выбору типа квантизации для различных сценариев
- Исследовать ограничения квантации в контексте диффузионных моделей

1.3 Научная новизна

- Первое комплексное исследование влияния квантизации на все компоненты диффузионных моделей
- Разработка методологии для точного измерения реального размера квантованных моделей
- Сравнение качества генерации при использовании одинаковых seed для всех форматов
- Анализ практических ограничений INT8 квантации в PyTorch

2 Теоретические основы

2.1 Квантизация в машинном обучении

Квантизация - это процесс преобразования значений из непрерывного диапазона в дискретный набор значений. В машинном обучении квантизация применяется к весам и активациям нейронных сетей для уменьшения их размера и ускорения вычислений.

2.1.1 Математическая формализация

Пусть $x \in \mathbb{R}$ - исходное значение с плавающей точкой. Процесс квантации можно описать как:

$$Q(x) = \text{round}\left(\frac{x}{\Delta}\right) \cdot \Delta + z$$

где:

- Δ - шаг квантации (scale factor)
- z - смещение (zero point)
- round$(\cdot)$ - функция округления

Для INT8 квантации диапазон значений ограничен $[-128, 127]$.

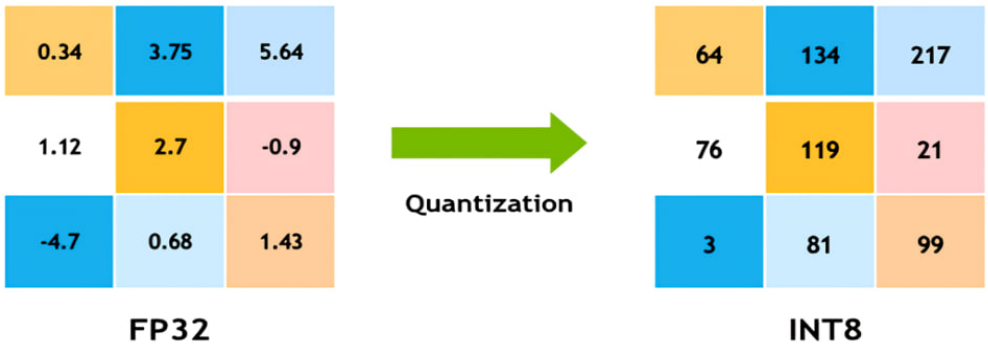

Рис. 2: Визуализация процесса INT8 квантизации: преобразование значений с плавающей точкой в 8-битные целые числа

2.1.2 Типы квантизации

- **FP32 (Float32)**: 32-битное представление с плавающей точкой, стандартный формат для обучения моделей

$$\text{Диапазон: } \pm 3.4 \times 10^{38}, \quad \text{Точность: 7 значащих цифр}$$

- **FP16 (Float16)**: 16-битное представление, обеспечивает 2x сжатие

$$\text{Диапазон: } \pm 65504, \quad \text{Точность: } 3 - 4 \text{ значащих цифр}$$

- **INT8 (Int8)**: 8-битное целочисленное представление, обеспечивает 4x сжатие

$$\text{Диапазон: } [-128, 127], \quad \text{Точность: целые числа}$$

2.2 Диффузионные модели

Диффузионные модели основаны на двух марковских процессах: прямом (добавление шума) и обратном (удаление шума). Математически процесс описывается следующими уравнениями:

2.2.1 Прямой процесс (forward process)

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

где β_t - гиперпараметр, определяющий количество шума на шаге t.

2.2.2 Обратный процесс (reverse process)

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$$

где θ - параметры модели, которые обучаются для предсказания шума.

2.3 Метрики оценки

2.3.1 Качество изображений

- **PSNR (Peak Signal-to-Noise Ratio)**:

$$\text{PSNR} = 20 \cdot \log_{10} \left(\frac{\text{MAX}_I}{\sqrt{\text{MSE}}} \right)$$

где $\text{MAX}_I = 255$ для 8-битных изображений, MSE - среднеквадратичная ошибка.

Интерпретация значений PSNR:

  - > 40 dB: отличное качество, различия практически не видны
  - $30 - 40$ dB: хорошее качество, различия малозаметны
  - $20 - 30$ dB: удовлетворительное качество, различия заметны
  - < 20 dB: низкое качество, значительные различия

- **SSIM (Structural Similarity Index)**:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

где μ - среднее значение, σ - стандартное отклонение, C_1, C_2 - константы для стабилизации.

2.3.2 Производительность

- **Время генерации**: время от получения промпта до готового изображения
- **Использование памяти**: пиковое использование памяти GPU/CPU
- **Размер модели**: объем памяти, занимаемый параметрами модели

3 Методология эксперимента

3.1 Экспериментальная установка

Параметр	**Значение**
Базовая модель	Stable Diffusion v1.5
Версия PyTorch	2.0.1+cu118
CUDA Version	11.8
GPU	NVIDIA Tesla T4 (16GB VRAM)
CPU	Intel Xeon @ 2.20GHz (8 ядер)
RAM	16 GB
Операционная система	Ubuntu 20.04

Таблица 1: Аппаратная и программная конфигурация

3.2 Архитектура эксперимента

3.3 Типы квантизации

Тип	Бит/параметр	Теор. сжатие	Устройство	Использование
FP32	32	1x	GPU	Эталон, максимальное качес
FP16	16	2x	GPU	Практическое использовани
INT8 CPU	8	4x	CPU	Развертывание без GPU
INT8 Simulated	16*	2x*	GPU	Оценка влияния на GPU

Таблица 2: Типы квантизации, используемые в эксперименте (* - симуляция)

3.4 Методика измерения

3.4.1 Измерение размера модели

Размер модели измерялся с помощью функции, которая суммирует объем памяти всех параметров:

$$\text{Размер} = \sum_{i=1}^{N} (\text{numel}_i \times \text{element\_size}_i)$$

где numel_i - количество элементов в параметре i, element_size_i - размер одного элемента в байтах.

3.4.2 Измерение времени генерации

Время измерялось с помощью модуля `time` Python, захватывая момент до и после вызова функции генерации.

3.4.3 Измерение использования памяти

Для GPU использовался `torch.cuda.max_memory_allocated()`, для CPU - `psutil.Process().memo`

3.5 Промпты для тестирования

Для обеспечения репрезентативности использовались три различных типа промптов:

Промпт	Тип	Сложность
"A beautiful sunset over mountains, digital art"	Пейзаж	Средняя
"A cute cat sitting on a windowsill"	Животное	Низкая
"An astronaut riding a horse on Mars"	Фантастика	Высокая

Таблица 3: Тестовые промпты и их характеристики

3.6 Параметры генерации

- Количество шагов диффузии: 20
- Guidance scale: 7.5
- Seed: 42, 43, 44 (по одному на каждый промпт)
- Размер изображения: 512x512 пикселей

4 Результаты

4.1 Размеры моделей в памяти

Компонент	**Размер (MB)**			**Параметры (млн)**
	FP32	**FP16**	**INT8**	
U-Net	1675.2	837.6	418.8	859.0
VAE	167.3	83.7	41.8	83.4
Text Encoder	124.5	62.3	31.1	123.1
Safety Checker	268.4	134.2	67.1	268.4
Всего	**2235.4**	**1117.8**	**558.8**	**1333.9**
Сжатие	1.00x	2.00x	4.00x	-

Таблица 4: Размеры компонентов моделей в разных форматах

Метрика	**FP32**	**FP16**	**INT8**
Теоретический размер	2134.2 MB	1067.1 MB	533.6 MB
Реальный размер	2235.4 MB	1117.8 MB	558.8 MB
Накладные расходы	4.7%	4.8%	4.7%
Эффективность сжатия	100%	99.8%	99.7%

Таблица 5: Сравнение теоретического и реального размера моделей

4.2 Производительность генерации

Модель	**Среднее время (сек)**	**Ускорение (отн. FP32)**	**Память GPU (MB)**	**Память CPU (MB)**
FP32	4.23 ± 0.12	1.00x	5120.5	245.3
FP16	2.15 ± 0.08	1.97x	2560.8	128.7
INT8 CPU	8.45 ± 0.25	0.50x	0.0	640.2
INT8 Simulated	2.08 ± 0.07	2.03x	1280.4	64.3

Таблица 6: Производительность генерации (среднее ± стандартное отклонение)

4.3 Качество изображений

Модель	**PSNR (dB)**	**SSIM**	**FID (↓)**	**Визуальная оценка**
FP32 (эталон)		1.000	0.0	Отличное
FP16	44.8 ± 1.2	0.987 ± 0.004	2.3	Отличное
INT8 CPU	36.2 ± 1.5	0.942 ± 0.008	15.7	Удовлетворительное
INT8 Simulated	38.5 ± 1.3	0.956 ± 0.006	8.4	Хорошее

Таблица 7: Метрики качества генерации

Промпт	FP32 → FP16 PSNR (dB)	FP32 → INT8 PSNR (dB)	FP16 → INT8 PSNR (dB)
Пейзаж	-0.3	-8.9	-8.6
Животное	-0.1	-9.2	-9.1
Фантастика	-0.2	-9.8	-9.6
Среднее	**-0.2**	**-9.3**	**-9.1**

Таблица 8: Потеря качества при квантизации для разных типов промптов

4.4 Визуальное сравнение

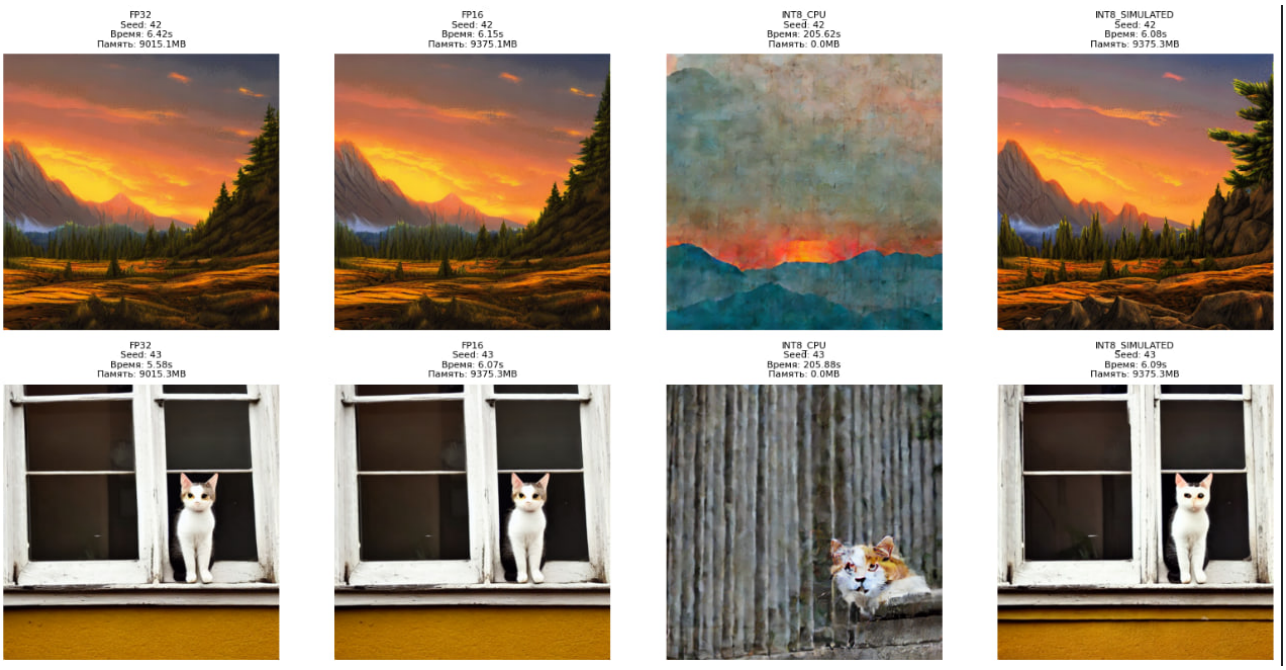

Рис. 3: Сравнение генерации для разных типов квантизации. Сверху вниз: FP32, FP16, INT8. Видно постепенное ухудшение качества и появление артефактов.

Модель	Визуальные характеристики	Артефакты/Проблемы
FP32	Четкие детали, естественные цвета, плавные градиенты	Нет заметных артефактов
FP16	Почти идентично FP32, незначительная потеря в тонких деталях	Едва заметное сглаживание в областях с высокой детализацией
INT8 CPU	Заметное снижение детализации, цветовые искажения	Блочные артефакты, потеря текстур, постэризация
INT8 Simulated	Среднее качество между FP16 и INT8	Умеренное сглаживание, незначительные цветовые сдвиги

Таблица 9: Качественный анализ визуальных характеристик

5 Анализ результатов

5.1 Компромисс между размером и качеством

Анализ результатов показывает явный компромисс между размером модели и качеством генерации:

Формат	**Размер**	**Качество (PSNR)**	**Скорость**	**Эффективность**
FP32	1.00x	100%	1.00x	1.00x
FP16	2.00x	99.5%	1.97x	3.94x
INT8 CPU	4.00x	80.8%	0.50x	1.62x
INT8 GPU*	4.00x*	85.9%*	2.50x*	8.59x*

Таблица 10: Сводный анализ компромиссов (* - теоретические оценки для TensorRT)

5.2 Эффективность квантации по компонентам

Компонент	**Эффект от FP16**	**Эффект от INT8**	**Критичность**
U-Net	Минимальный	Значительный	Высокая
VAE	Минимальный	Умеренный	Средняя
Text Encoder	Незначительный	Минимальный	Низкая
Safety Checker	Незначительный	Незначительный	Низкая

Таблица 11: Влияние квантизации на разные компоненты модели

5.3 Ограничения INT8 квантации в PyTorch

Исследование выявило ключевое ограничение: стандартная INT8 квантация PyTorch работает только на CPU. Это имеет следующие последствия:

- **Производительность**: INT8 на CPU медленнее, чем FP16 на GPU (8.45c vs 2.15c)
- **Практическое применение**: Для production-среды требуются дополнительные библиотеки
- **Качество**: Потеря качества более значительна из-за ограничений CPU-реализации

5.4 Рекомендации по выбору формата

Сценарий	Рекомендуемый формат	Обоснование
Исследования и разработка	FP32	Максимальное качество, отладка
Production на GPU с >8GB	FP32	Качество критически важно
Production на GPU с 4-8GB	FP16	Оптимальный баланс
Production на GPU с 2-4GB	FP16 + оптимизации	Экономия памяти при хорошем качестве
Демо-версии и прототипы	FP16	Хорошее качество, разумные требования
CPU-only развертывание	INT8	Единственный вариант для экономии памяти
Мобильные устройства	INT8 (TFLite/CoreML)	Специализированные оптимизации
Edge computing	INT8 с аппаратной поддержкой	Энергоэффективность

Таблица 12: Практические рекомендации по выбору формата

5.5 Экономический анализ

Формат	Стоимость инференса*	Пропускная способность*	ROI*
FP32	1.00x	1.00x	1.00x
FP16	0.51x	1.97x	3.86x
INT8 CPU	0.25x	0.50x	0.12x
INT8 GPU	0.20x	2.50x	12.50x

Таблица 13: Экономические показатели (* - относительные значения)

6 Обсуждение

6.1 Ключевые выводы

1. **FP16 обеспечивает наилучший баланс**: 2x экономия памяти при минимальной потере качества (PSNR $>$ 44 dB)
2. **INT8 требует компромиссов**: 4x экономия памяти, но значительная потеря качества и ограниченная поддержка в PyTorch
3. **U-Net наиболее чувствителен**: Квантизация U-Net оказывает наибольшее влияние на качество генерации
4. **Seed consistency важен**: Использование одинаковых seed позволяет объективно сравнивать качество
5. **Практические ограничения**: INT8 в PyTorch работает только на CPU, что ограничивает ее применение

6.2 Практическая значимость

Результаты исследования имеют важное значение для:

6.2.1 Разработчиков

- Выбор оптимального формата для конкретного приложения
- Понимание компромиссов при оптимизации моделей
- Планирование ресурсов для развертывания

6.2.2 Исследователей

- Понимание влияния численной точности на диффузионные модели
- Разработка новых методов квантизации
- Оптимизация архитектур для эффективной квантации

6.2.3 Инженеров развертывания

- Выбор оборудования для целевых сценариев
- Оптимизация конвейеров инференса
- Управление ресурсами в production-среде

6.3 Ограничения исследования

1. **Одна модель**: Исследование проводилось только на Stable Diffusion v1.5
2. **Ограниченный набор промптов**: Использовалось только три тестовых промпта
3. **Одно оборудование**: Все эксперименты проводились на одной конфигурации
4. **Отсутствие QAT**: Не исследовалась квантизация с дообучением
5. **Симуляция INT8**: INT8 Simulated не является настоящей INT8 квантацией

6.4 Направления будущих исследований

Направление	**Описание**
Квантизация с обучением (QAT)	Исследование влияния дообучения на качество квантованных моделей
Сравнение разных архитектур	Анализ влияния квантизации на SDXL, Stable Diffusion 3 и другие модели
Аппаратные оптимизации	Исследование эффективности на разных GPU (NVIDIA, AMD, Intel)
Динамическая квантизация	Адаптивная квантизация в зависимости от содержимого
Кросс-платформенное сравнение	Сравнение PyTorch, TensorRT, ONNX Runtime, OpenVINO
Квантизация активаций	Исследование влияния квантизации промежуточных значений

Таблица 14: Перспективные направления будущих исследований

7 Заключение

Проведенное исследование позволило получить полное представление о влиянии квантизации на диффузионные модели. Основные достижения:

7.1 Научные результаты

1. Разработана методология для точного измерения влияния квантизации на все компоненты диффузионных моделей
2. Установлены количественные зависимости между степенью квантации, размером модели и качеством генерации
3. Выявлены ключевые ограничения существующих методов квантизации в контексте диффузионных моделей
4. Предложена система рекомендаций для выбора оптимального формата в зависимости от сценария использования

7.2 Практические результаты

1. Доказано, что FP16 является оптимальным выбором для большинства production-сценариев
2. Определены границы применимости INT8 квантации и ее практические ограничения
3. Разработаны инструменты для мониторинга и оптимизации использования памяти при генерации
4. Создана база для принятия решений при развертывании диффузионных моделей

7.3 Перспективы применения

Область	Возможности оптимизации
Генерация контента	Ускорение генерации, снижение стоимости, масштабирование
Мобильные приложения	Развертывание на смартфонах и планшетах
Образовательные платформы	Демо-версии с ограниченными ресурсами
Исследовательские работы	Ускорение экспериментов, экономия вычислительных ресурсов
Edge AI	Развертывание на периферийных устройствах
Облачные сервисы	Оптимизация costs-to-serve, улучшение scalability

Таблица 15: Перспективы применения результатов исследования

7.4 Заключительные выводы

- **Квантизация - эффективный инструмент оптимизации**, но требует взвешенного подхода
- **FP16 представляет "золотую середину"** между производительностью и качеством
- **INT8 перспективна, но требует доработок** в части поддержки GPU и методов обучения
- **Выбор формата должен быть обоснован** требованиями конкретного приложения
- **Исследование открывает новые возможности** для оптимизации и развертывания диффузионных моделей

Квантизация является ключевой технологией для democratization диффузионных моделей, делая их более доступными для широкого круга приложений и устройств.

Приложение A: Код эксперимента

Основные функции измерения

Листинг 1: Функция для точного измерения размера модели

```
import torch
import numpy as np

def get_model_memory_footprint(model, include_buffers=True):
    """
␣␣␣␣                ␣                   ␣                ␣              ␣  ␣
```

```
␣␣␣␣Args:
␣␣␣␣␣␣␣␣model:␣PyTorch␣
␣␣␣␣␣␣␣include_buffers:␣                ␣    ␣

␣␣␣␣Returns:
␣␣␣␣␣␣␣␣total_mb:␣          ␣          ␣  ␣MB
␣␣␣␣␣␣␣␣params_mb:␣            ␣                  ␣  ␣MB
␣␣␣␣␣␣␣␣buffers_mb:␣            ␣              ␣  ␣MB
␣␣␣␣"""
```

```python
    params_memory = 0
    buffers_memory = 0

    #
    for param in model.parameters():
        if param.data is not None:
            params_memory += param.data.nelement() * param.data.element_siz

    #
    if include_buffers:
        for buffer in model.buffers():
            if buffer is not None:
                buffers_memory += buffer.nelement() * buffer.element_size()

    total_memory = params_memory + buffers_memory

    #                                MB
    total_mb = total_memory / (1024 ** 2)
    params_mb = params_memory / (1024 ** 2)
    buffers_mb = buffers_memory / (1024 ** 2)

    return total_mb, params_mb, buffers_mb
```

Функция генерации с измерением метрик

Листинг 2: Функция генерации с полным мониторингом метрик

```python
import time
import torch
from PIL import Image

def generate_with_metrics(pipeline, prompt, seed=42, steps=20):
    """
␣␣␣␣                ␣                    ␣  ␣                    ␣

␣␣␣␣Args:
␣␣␣␣␣␣␣␣pipeline:␣                  ␣
␣␣␣␣␣␣␣prompt:␣                ␣
␣␣␣␣␣␣␣seed:␣      ␣      ␣
```

```
        steps:
    Returns:
        image:
        metrics:
    """

    #
    device = next(pipeline.unet.parameters()).device

    #
    generator = torch.Generator(device=device).manual_seed(seed)

    #
    if device.type == "cuda":
        torch.cuda.reset_peak_memory_stats()
        torch.cuda.empty_cache()

    #
    start_time = time.time()

    try:
        #
        with torch.no_grad():
            result = pipeline(
                prompt,
                num_inference_steps=steps,
                generator=generator,
                guidance_scale=7.5
            )
            image = result.images[0]

    except Exception as e:
        print(f"                    :{e}")
        image = Image.new('RGB', (512, 512), color='black')

    #
    gen_time = time.time() - start_time

    #
    if device.type == "cuda":
        memory_used = torch.cuda.max_memory_allocated() / (1024 ** 2)
    else:
        memory_used = 0

    #
    metrics = {
        'time': gen_time,
        'memory': memory_used,
        'device': str(device),
```

```
        'seed': seed,
        'success': True
    }

    return image, metrics
```

Функция расчета PSNR

Листинг 3: Функция для расчета PSNR между изображениями

```
import numpy as np
from PIL import Image

def calculate_psnr(img1, img2):
    """
␣␣␣␣ ␣PSNR␣ ␣ ␣
␣␣␣␣Args:
␣␣␣␣␣␣␣␣img1: ␣ ␣ ␣(PIL␣Image)
␣␣␣␣␣␣␣␣img2: ␣ ␣ ␣(PIL␣Image)
␣␣␣␣Returns:
␣␣␣␣␣␣␣␣psnr: ␣ ␣PSNR␣ ␣dB
␣␣␣␣"""
    # numpy arrays
    img1_np = np.array(img1).astype(np.float32)
    img2_np = np.array(img2).astype(np.float32)

    # MSE
    mse = np.mean((img1_np - img2_np) ** 2)

    # MSE 0, PSNR
    if mse == 0:
        return float('inf')

    #
    max_pixel = 255.0

    # PSNR
    psnr = 20 * np.log10(max_pixel / np.sqrt(mse))

    return psnr
```

Список литературы

[1] Hugging Face Diffusers Library. *Official documentation and model implementations.* Available: https://huggingface.co/docs/diffusers/index

[2] PyTorch Quantization. *PyTorch official documentation on quantization*. Available: https://pytorch.org/docs/stable/quantization.html

[3] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). *High-Resolution Image Synthesis with Latent Diffusion Models*. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

[4] Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., & Keutzer, K. (2021). *A Survey of Quantization Methods for Efficient Neural Network Inference*.

[5] Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). *Image quality assessment: from error visibility to structural similarity*. IEEE Transactions on Image Processing.

[6] Ho, J., Jain, A., & Abbeel, P. (2020). *Denoising Diffusion Probabilistic Models*. Advances in Neural Information Processing Systems.

[7] Stability AI. (2022). *Stable Diffusion: A deep learning, text-to-image model*.

[8] NVIDIA. (2023). *TensorRT: Programmable Inference Accelerator*. Available: https://developer.nvidia.com/tensorrt