

# Руководство по локальному запуску VLM на ноутбуках с GPU.

Данное руководство нацелено по шагам показать как запустить VLM на ноутбуке с GPU на примере модели Qwen3-VL-8B-Instruct-FP8/Qwen3-VL-8B-Thinking-FP8.

Гарантирована работа на ноутбуке STEALTH 18 HX AI A2XW:

Процессор Intel Core Ultra 9 - 32Гб.

GPU RTX5090 - 24Гб, CUDA 12.8, Драйвер 573.22.

## 1. Общая архитектура решения

Что мы в итоге получим:

1. WSL2 (Ubuntu 24.04) - удобная Linux-среда внутри Windows.
2. Docker + NVIDIA Container Toolkit - чтобы запускать модели в контейнере с доступом к GPU.
3. Контейнер с vLLM, в котором загружена VLM Qwen3-VL-8B-Instruct-FP8 и поднят OpenAI-совместимый HTTP API на <http://localhost:8000/v1>.
4. Open WebUI - веб-интерфейс чата с моделью на <http://localhost:8080/>.
5. Примеры интеграции:
  - через LangChain (текстовый чат и обработка изображения);
  - через Open WebUI (чат, настройки моделей, изображения и аудио).

Кратко, почему выбираю vLLM движок для инференса:

- vLLM всегда быстрее на GPU за счёт PagedAttention и продвинутого управления памятью.
- Лучше масштабируется: удобен для серверных нагрузок, множества одновременных запросов, батчинга.
- Гибче в продакшне: нормальный Python API, интеграция с FastAPI, тонкая настройка генерации.
- Поддерживает больше режимов и оптимизаций (tp, kv-cache, сложные пайплайны).

Ollama (рассматривается [здесь](#)) удобнее для десктопа и простого запуска для одного пользователя, vLLM - для производительного GPU-сервера и бэкенда.

Таким образом, с учетом основной цели - развертывание сервисов в продакшне, выбор за vLLM.

Кратко, почему выбираю Qwen3-VL-8B-Instruct-FP8/Qwen3-VL-8B-Thinking-FP8:

- Поддерживается работа с 32 языками.
- Доступен контекст объёмом до 256k токенов.
- Квантизация FP8;
- Модель обладает возможностью обработки изображений:
  - пользователь может отправить изображение и задать вопросы по его содержимому;
  - пользователь может отправить фотографию страницы, нарисованной от руки (например, карандашом), и запросить генерацию HTML-страницы с интерфейсом, соответствующим изображённому макету;
  - пользователь может отправить отсканированные или сфотографированные страницы и запросить распознавание текста либо подготовку конспекта.
  - Модель интерпретирует математические выражения и может давать разъяснения по математике, химии, электронике и смежным дисциплинам.
- Предусмотрены две конфигурации модели:
  - с использованием механизма рассуждений (THINKING);
  - без механизма рассуждений (INSTRUCT);
- SOTA в своей весовой категории.

## 2. Установка и настройка WSL2 (Ubuntu 24.04)

Этот блок следует выполнять в Windows PowerShell последовательно:

### WSL

```
wsl --install #      WSL (    WSL    )
wsl.exe --list --online #
wsl.exe --install Ubuntu-24.04 #  Ubuntu-24.04 (    22.04)
wsl --terminate Ubuntu-24.04 #  (    )
wsl --unregister Ubuntu-24.04 #  (    )
```

**Дальнейшая работа должна проходить в терминале Ubuntu-24.04.**

## 3. Базовая подготовка Ubuntu

Обновляем пакеты и ставим необходимые утилиты (curl и gnupg2):

### Ubuntu-24.04

```
sudo apt-get update && sudo apt-get install -y --no-install-recommends \
  curl \
  gnupg2
```

Доустанавливаем Python (dev, venv, pip) и инструменты сборки (по умолчанию в Ubuntu-24.04 уже установлен Python 3.12, который является последней стабильной версией. Версии начиная с 3.13 все еще на bugfix этапе, так что не советую их выбирать для текущих задач, так как не все бинарные пакеты (.whl) могут быть доступны):

### Python

```
sudo apt install -y python3-dev python3-venv python3-pip build-essential
```

## 4. Установка Docker в Ubuntu

Добавляем официальный репозиторий докер:

### Docker

```
sudo apt update
sudo apt install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

sudo tee /etc/apt/sources.list.d/docker.sources <<EOF
Types: deb
URIs: https://download.docker.com/linux/ubuntu
Suites: $(. /etc/os-release && echo "${UBUNTU_CODENAME}:-$VERSION_CODENAME}")
Components: stable
Signed-By: /etc/apt/keyrings/docker.asc
EOF

sudo apt update
```

Устанавливаем сам docker:

## Docker

```
sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Устанавливаем docker compose:

## Docker compose

```
sudo apt install docker-compose
```

Добавляем пользователя в группу docker (чтобы не нужно было постоянно выполнять команды docker через sudo)

## Docker

```
sudo usermod -aG docker $USER  
newgrp docker
```

Проверяем:

## Docker

```
docker ps
```

Если список контейнеров выводится без ошибок, docker работает.

## 5. Настройка NVIDIA Container Toolkit (Пробрасываем GPU в контейнеры)

Добавляем репозиторий NVIDIA Container Toolkit:

## NVIDIA Container Toolkit

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \  
&& curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | \  
sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' | \  
sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

Включаем experimental-репозиторий и обновляем пакеты:

## NVIDIA Container Toolkit

```
sudo sed -i -e '/experimental/ s/^#/g' /etc/apt/sources.list.d/nvidia-container-toolkit.list  
sudo apt-get update
```

Устанавливаем нужную версию NVIDIA Container Toolkit. На текущий момент 1.18.0-1

## NVIDIA Container Toolkit

```
export NVIDIA_CONTAINER_TOOLKIT_VERSION=1.18.0-1
sudo apt-get install -y \
    nvidia-container-toolkit=${NVIDIA_CONTAINER_TOOLKIT_VERSION} \
    nvidia-container-toolkit-base=${NVIDIA_CONTAINER_TOOLKIT_VERSION} \
    libnvidia-container-tools=${NVIDIA_CONTAINER_TOOLKIT_VERSION} \
    libnvidia-container1=${NVIDIA_CONTAINER_TOOLKIT_VERSION}

sudo systemctl restart docker
```

Проверяем:

## Docker

```
docker run --rm --gpus all nvidia/cuda:12.8.0-runtime-ubuntu24.04 nvidia-smi
```

Если выводится информация о вашей видеокарте и драйвере - всё настроено корректно. Пример:

```
NVIDIA-SMI 570.151          Driver Version: 573.22        CUDA Version: 12.8
+-----+-----+-----+
| GPU  Name           Persistence-M | Bus-Id     Disp.A  | Volatile Uncorr. ECC
| Fan  Temp   Perf   Pwr:Usage/Cap |          Memory-Usage | GPU-Util Compute M.
|          |          |              |                |          | MIG M.
+-----+-----+-----+
| 0  NVIDIA GeForce RTX 5090 ...  On   00000000:01:00.0 Off  N/A
| N/A  50C   P2       18W / 110W | 0MiB / 24463MiB | 0%     Default N/A
+-----+-----+-----+
Processes:
GPU  GI  CI          PID  Type  Process name          CPU Memory Usage
ID   ID
+-----+
No running processes found
```

## 6. Загрузка модели Qwen3-VL-8B-Instruct-FP8/Qwen3-VL-8B-Thinking-FP8 с Hugging Face

Модели находятся в репозитории:

- Qwen/Qwen3-VL-8B-Instruct-FP8
- Qwen/Qwen3-VL-8B-Thinking-FP8

Для скачивания VLM используется Git LFS (либо можно скачать вручную и перенести в Ubuntu через Проводник ->необходимы: веса модели в формате safetensors, config.json, tokenizer.json, tokenizer\_config.json, generation\_config.json.

## Git LFC

```
curl -s https://packagecloud.io/install/repositories/github/git-lfs/script.deb.sh | sudo bash
sudo apt-get install -y git-lfs
git lfs install
```

В домашней директории создаем каталог проектов (опционально):

## Ubuntu-24.04

```
mkdir -p ~/projects
cd ~/projects
```

Клонируем модель (занимает 20-30 минут):

#### Git LFC

```
git clone https://huggingface.co/Qwen/Qwen3-VL-8B-Instruct-FP8  
git clone https://huggingface.co/Qwen/Qwen3-VL-8B-Thinking-FP8
```

Если вы клонируете приватные/ограниченные модели, то будет нужен токен доступа:

#### Git LFC

```
git clone https://<YOUR_HF_TOKEN>@huggingface.co/<MODEL_PATH>
```

## 7. Запуск vLLM с моделью Qwen3-VL-8B-Instruct-FP8

### 7.1. Используя Docker (рекомендованный)

Официально поддержка архитектуры Qwen3-VL в vLLM через transformers добавлена в версии `vllm>=0.11.0`.

Однако данная версия требует `CUDA>=12.9`. Так как мы имеем CUDA 12.8 и не можем обновить ее через бинарник (ограничения кибербезопасности), то пишем кастомный Dockerfile. Это минимальный образ с CUDA, Python и vLLM:

#### Dockerfile

```
FROM nvidia/cuda:12.8.0-devel-ubuntu24.04  
  
RUN apt-get update && apt-get install -y \  
    python3 python3-pip build-essential \  
    && rm -rf /var/lib/apt/lists/*  
  
RUN pip3 install --break-system-packages "vllm>=0.11.0"  
  
RUN pip3 install --break-system-packages "accelerate>=0.34.0"  
  
RUN pip3 install --break-system-packages nvidia-ml-py  
  
EXPOSE 8000  
CMD ["python3", "-m", "vllm.entrypoints.openai.api_server"]
```

Рядом создаем docker-compose.yaml для запуска сервиса модели:

## Docker compose

```
docker-compose.yaml
services:
  model:
    build: .
    container_name: qwen3vl-8b-instruct-fp8
    entrypoint: []
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              capabilities: [gpu]
              device_ids: ['0']
    ports:
      - "8000:8000"
    environment:
      TORCH_CUDA_ARCH_LIST: "8.9"
    volumes:
      - ${HOME}/projects/Qwen3-VL-8B-Instruct-FP8:/model:ro
      - ${HOME}/.cache:/root/.cache
    ipc: host
    command:
      - python3
      - -m
      - vllm.entrypoints.openai.api_server
      - --model
      - /model
      - --max-model-len
      - "8192"
      - --served-model-name
      - "qwen3vl-8b-instruct-fp8"
      - --dtype
      - "auto"
      - --quantization
      - "fp8"
      - --gpu-memory-utilization
      - "0.93"
      - --max-num-seqs
      - "1"
```

Важно:

- volumes:
  - \${HOME}/projects/Qwen3-VL-8B-Instruct-FP8:/model:ro - путь до директории с моделью, которую вы склонировали с Hugging Face.
- --served-model-name "qwen3vl-8b-instruct-fp8" - именно это имя вы будете указывать в клиентах (Open WebUI, LangChain и т.д.).

Если запускаем Thinking версию, то заменяем везде Instruct и добавляем параметр:

```
--reasoning-parser
- "deepseek_r1"
```

Значение поля "reasoning-parser" именно "deepseek\_r1". Без этого параметра рассуждения попадают в сам ответ, а не в специальную часть для reasoning.

Сборка и запуск контейнера с моделью:

## Docker

```
docker compose build
docker compose up
```

При успешном запуске в логах увидите, что vLLM загружает модель и поднимает OpenAI-совместимый сервер на порту 8000.  
Можно проверить:

#### Ubuntu-24.04

```
curl http://localhost:8000/v1/models
```

Должен вернуться JSON со списком моделей, среди которых - qwen3vl-8b-instruct-fp8.

## 7.2. Используя Python

Создаем изолированное окружение:

```
python3 -m venv env  
source env/bin/activate
```

Устанавливаем vLLM и библиотеки huggingface (могут понадобиться если скачивать модели напрямую):

```
python -m pip install vllm _hub huggingface_hub[cli]
```

Запускаем vLLM сервер с моделью:

#### vLLM

```
vllm serve /model \  
--served-model-name qwen3vl-8b-instruct-fp8 \  
--max-model-len 8192 \  
--dtype auto \  
--quantization fp8 \  
--gpu-memory-utilization 0.93 \  
--max-num-seqs 1 \  
--host 0.0.0.0 \  
--port 8000
```

Проверка работоспособности аналогично вышеннаписанному.

## 8. Установка и запуск Open WebUI

Open WebUI - это удобный веб-интерфейс, который будет обращаться к нашему vLLM-серверу как к OpenAI-совместимому API.

Запускаем отдельный контейнер (можно добавить в docker-compose.yaml как один из сервисов, здесь разделено для лучшего восприятия):

#### Open WebUI

```
docker run -d \  
-p 8080:8080 \  
--add-host host.docker.internal:host-gateway \  
-v open-webui:/app/backend/data \  
--name open-webui \  
--restart always \  
ghcr.io/open-webui/open-webui:main
```

После запуска откройте в браузере <http://localhost:8080/>:

Настройка подключения к vLLM в Open WebUI:

- Зайдите под админом (при первом запуске создаётся первый пользователь - он же админ).
- Перейдите в Панель администратора Настройки Подключения.
- Добавьте новое подключение с URL <http://host.docker.internal:8000/v1>

The screenshot shows two main sections of the HR-assistant application:

- User Management (Left Side):** Shows a sidebar with "Настройки", "Архив чатов", "Лесенница", "Панель администратора" (highlighted with a red arrow), and "Выход". Below it, "Активные пользователи: 1" and "admin" (highlighted with a red arrow) are listed.
- Connection Configuration (Right Side):** Shows the "Настройки" tab selected. Under "Подключения", the "API OpenAI" section is highlighted with a red arrow. It shows the URL "http://host.docker.internal:8000/v1" which is also highlighted with a red arrow. Other sections like "Ollama API" and "Прямые подключения" are visible.

Перейдите в раздел Модели:

- Вы увидите модель `qwen3v1-8b-instruct-fp8`.
- Можно настроить кастомизацию: системный промпт, лого, параметры генерации. Рекомендую:
  - `top_k=20`,
  - `top_p=0.95`,
  - `temperature=0.5`.
  - `frequency_penalty=1.5`.

The screenshot shows the "Настройки" tab selected. Under the "Модели" section, the "HR-assistant" model is listed. A red arrow points from the left margin to the "HR-assistant" entry. At the bottom right, there are buttons for "Импортировать Пресеты" and "Экспорт Пресетов (1)".

Перейдите в раздел Аудио (голосовой ввод):

- Выберите Whisper (локально).
- Замените название модели base на turbo.

Пользователи Оценки Функции Настройки

Общее Речь в текст

Поддерживаемые MIME типы напр., audio/wav,audio/mpeg,video/\* (оставьте пустым для значений по умолчанию)

Система распознавания речи

Модель распознавания речи

turbo

Open WebUI использует более быстрый внутренний интерфейс whisper. Нажмите здесь, чтобы узнать больше о faster-whisper и ознакомиться с доступными моделями.

Текст в речь

Система синтеза речи

Голос TTS

По умолчанию

Разделение ответов

Web API

Paragraphs

Перейдите в раздел Изображения. Опционально. Так как для разворачивания параллельно с LLM модели для генерации изображений не хватает VRAM, то я использую OpenAI API через прокси. При достаточном количестве свободной VRAM (минимум 14Гб для Kandinsky 5.0 Image Lite) можно развернуть модель через ComfyUI и добавить ее в Open WebUI (<https://habr.com/ru/companies/hostkey/articles/884664/>):

- Endpoint: <https://api.proxyapi.ru/openai/v1>
- Модель: gpt-image-1

Функции Настройки

Общее

Генерация изображений

Create Image

Модель gpt-image-1

Image Size 1024x1024

Генерация промпта к изображению

Механизм генерации изображений

OpenAI API Base URL

OpenAI API Key

OpenAI API Version

Additional Parameters

⋮

Edit Image

Image Edit

Модель gpt-image-1

Image Size 1024x1024

Image Edit Engine

OpenAI API Base URL

OpenAI API Key

OpenAI API Version

## Результаты:

HR-ассистент

Здравствуйте! Я — виртуальный HR-ассистент ЗАО «Альфа-Банк» Республики Беларусь. Моя задача — стать вашим надежным помощником в повседневной работе с кандидатами и подборе персонала, чтобы процесс был максимально эффективным, профессиональным и дружелобным.

Что я умею:

- Генерировать профессиональные тексты для коммуникаций

Я помогу вам составить:

- Приглашения на собеседование;
- Отклики на резюме (включая благодарности и отказы);
- Напоминания кандидатам;
- Внутренние комментарии по профилям;
- Корпоративные посты/объявления в соцсетях;
- Тексты для email рассылок.

Все тексты будут соответствовать стилю «Альфа-Банка» — деловому, но дружелобному, с акцентом на ценности компании.

Советовать по формулировке вакансий

Если вы пишете описание должности — я помогу:

- Структурировать его правильно (заголовок → обязанности → требования → условия → ценности);
- Использовать инкогнитный и привлекательный язык;
- Соответствовать бренду банка и HR-политике.

Анализировать резюме

Вы загрузите скан или фото резюме? Я:

- Выделю ключевые навыки, опыт и достижения;
- Укажу несоответствия требованиям.

HR-ассистент

```
class ChatModel:
    def __init__(self):
        from langchain_core.messages import HumanMessage, SystemMessage
        self.llm = ChatOpenAI(model="qwenv1-8b-instruct-fss", temperature=0.3, max_tokens=500)

    def generate(self, message):
        base_url = "http://localhost:8000/v1"
        headers = {"Content-Type": "application/json", "Authorization": f"Bearer {self.llm._api_key}"}
        data = {
            "model": "qwenv1-8b-instruct-fss",
            "temperature": 0.3,
            "max_tokens": 500,
            "streaming": True
        }
        response = requests.post(f"{base_url}/chat", json={"message": message}, headers=headers)
        return response.json()["text"]
```

Что ты видишь на изображении?

HR-ассистент

На изображении — фрагмент кода на Python, который инициализирует и запускает локальный LLM (Large Language Model) через API vLLM, а затем отправляет ему запрос с помощью LangChain.

Что я вижу:

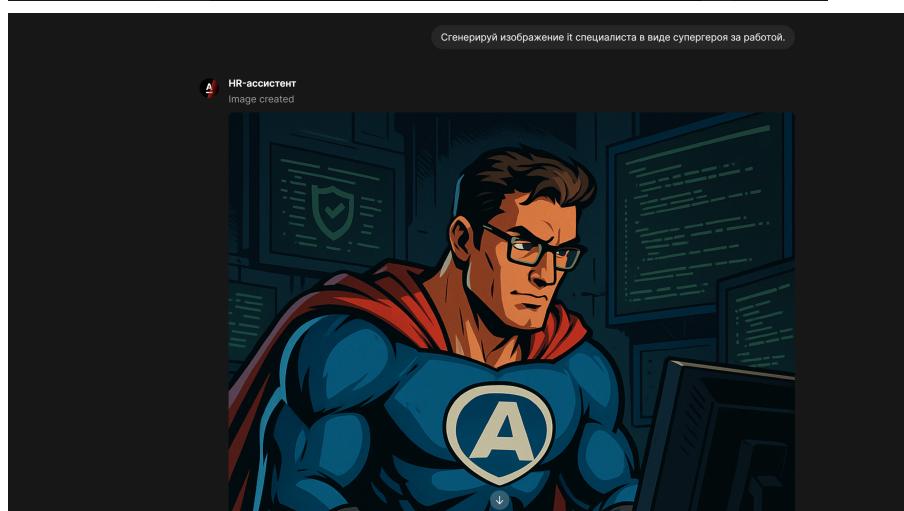
- Импорт библиотек:

```
from langchain_core.messages import HumanMessage, SystemMessage
```

Подключаются классы для работы с сообщениями: системное (`SystemMessage`) и пользовательское (`HumanMessage`).

- Инициализация модели:

```
llm = ChatModel()
llm.generate("Привет")
```



## 9. Использование модели через LangChain

Текстовый запрос:

```
from langchain_openai import ChatOpenAI
from langchain_core.messages import HumanMessage, SystemMessage
```

```
#     OpenAI- API vLLM
llm = ChatOpenAI(
    base_url="http://localhost:8000/v1",           # URL vLLM
    api_key="token-abc",                           # vLLM ,
    model="qwen3v1-8b-instruct-fp8",               # ,      --served-model-name
    temperature=0.3,                             #
    max_tokens=6000,                            #
    streaming=True,                            #
)
messages = [
    SystemMessage(content=" - AI ."),
    HumanMessage(content="! . ?")
]
print("\n :")
for chunk in llm.stream(messages):
    print(chunk.content, end="", flush=True)
print()
```

Если всё настроено корректно, вы увидите потоковый ответ от модели.

## Анализ изображений и OCR:

```
import base64
from PIL import Image
from io import BytesIO
from langchain_openai import ChatOpenAI
from langchain_core.messages import SystemMessage, HumanMessage

def image_to_base64(image_path: str, max_size: tuple = (512, 512)) -> str:
    """ , base64 """
    with Image.open(image_path) as img:
        if img.mode != "RGB":
            img = img.convert("RGB")
        img.thumbnail(max_size)
        buffered = BytesIO()
        img.save(buffered, format="JPEG")
    return base64.b64encode(buffered.getvalue()).decode("utf-8")

llm = ChatOpenAI(
    base_url="http://localhost:8000/v1",
    api_key="token-abc",
    model="qwen3v1-8b-instruct-fp8",
    temperature=0.0,
)

image_path = "data/295914_original.jpg"
image_b64 = image_to_base64(image_path)

messages = [
    SystemMessage(content=" - AI .      OCR. "),
    HumanMessage(
        content=[
            {"type": "text", "text": " . . . "},
            {
                "type": "image_url",
                "image_url": {"url": f"data:image/jpeg;base64,{image_b64}"}
            }
        ]
    ),
]
print("Output:")
```

```
for chunk in llm.stream(messages):
    print(chunk.content, end="", flush=True)
print()
```

Модель должна выдать распознанный текст с изображения (OCR).

## 10. Вместо вывода:

Всегда существует вероятность возникновения непредвиденных ошибок. В случае обнаружения проблем или некорректной работы в процессе настройки, пожалуйста, сообщите об этом. Совместно проанализируем ситуацию и устраним проблему.