Инструкция по установке

Программного обеспечения

2026 г.

Содержание

1. Системные требования

Перед установкой ПО убедитесь, что сервер соответствует следующим минимальным системным требованиям.

Рекомендованная конфигурация: СЕРВЕР ГРАВИТОН С2122ИУ 2xS4516Y+-2xPH270W2-8x16GD5-1x16IRAID8-2x480GBDWPD1-1xRTXA4000-2x1600W-3YST

Примечание: Используемая модель Llama-3-8B-Instruct (формат GGUF, квантование Q8_0) имеет 8 млрд параметров и требует для запуска видеокарту с не менее 12 ГБ видеопамяти.

2. Предварительные требования

Перед началом установки убедитесь, что на целевом сервере установлены следующие компоненты:

2.1. Docker

Убедитесь, что Docker установлен и запущен. Для проверки выполните команду:

```
docker --version
```

Ожидаемый результат: вывод версии Docker (20.10 или выше).

2.2. Поддержка GPU в Docker

Для работы с GPU необходима установка Container Toolkit.

Ожидаемый результат: таблица с информацией о доступных GPU.

3. Состав поставки

Решение поставляется в виде сконфигурированного docker-контейнера в формате *.tar.

Контейнер	**Описание**
base_webui_v5	Решение на основе llama-server (llama.cpp). Включает Web UI (React), RAG-систему (FastAPI + ChromaDB + LlamaIndex) и llama-server для инференса модели. Поддерживает квантованные модели формата GGUF. Работа на GPU и/или CPU.

Состав контейнера (запускаемые сервисы):

Сервис	**Технология**	**Назначение**
gui	React + Vite (порт 5173)	Web-интерфейс пользователя (чат)

model	llama-server (порт 5000)	Инференс LLM-модели, OpenAI-совместимый API
rag	FastAPI + ChromaDB (порт 18000)	RAG-система: индексация и поиск по базе знаний (Wiki)

4. Установка на локальный сервер

4.1. Загрузка docker-контейнера

1. Скопируйте файл контейнера (*.tar) на сервер в удобный каталог.
2. Загрузите контейнер в Docker, выполнив команду:

```
docker load --input base_webui_v5.tar
```

Ожидаемый результат:

```
Loaded image: desm0nt/base_webui_v5:latest
```

3. Убедитесь, что образ загружен, выполнив команду:

```
docker images
```

В списке образов должен появиться образ *desm0nt/base_webui_v5* с тегом *latest*.

4.2. Запуск контейнера

Для запуска контейнера с поддержкой GPU выполните команду:

```
docker run -it --rm --gpus all -p 7860:7860 -p 5000:5000 -p 5005:5005 -p
5173:5173 -p 18000:18000 desm0nt/base_webui_v5:latest
```

Описание параметров:

- *-it* — запуск в интерактивном режиме с выводом логов;
- *--rm* — автоматическое удаление контейнера после остановки;
- *--gpus all* — предоставление доступа ко всем GPU;
- *-p 5173:5173* — порт Web UI приложения (основной интерфейс);
- *-p 5000:5000* — порт OpenAI-совместимого API (llama-server);
- *-p 18000:18000* — порт RAG-системы (FastAPI);
- *-p 5005:5005* — порт streaming;
- *-p 7860:7860* — служебный порт (зарезервирован).

4.3. Проверка успешного запуска

При первом запуске контейнер выполняет инициализацию: устанавливает зависимости (Node.js, Python-пакеты), собирает frontend и загружает модель из репозитория HuggingFace. Это может занять значительное время.

1. Дождитесь полного запуска контейнера. В процессе инициализации в логах будут отображаться этапы:

- установка системных пакетов и зависимостей;
- скачивание модели (при первом запуске);
- сборка frontend (npm install, npm run build-css);
- установка компонентов RAG-системы (chromadb, llama-index и др.).

2. По окончании инициализации в логах docker-контейнера появятся сообщения о запуске всех трёх сервисов (gui, model, rag).
3. Откройте браузер и перейдите по адресу:

```
http://<IP-адрес_сервера>:5173
```

Должен открыться Web UI приложения.

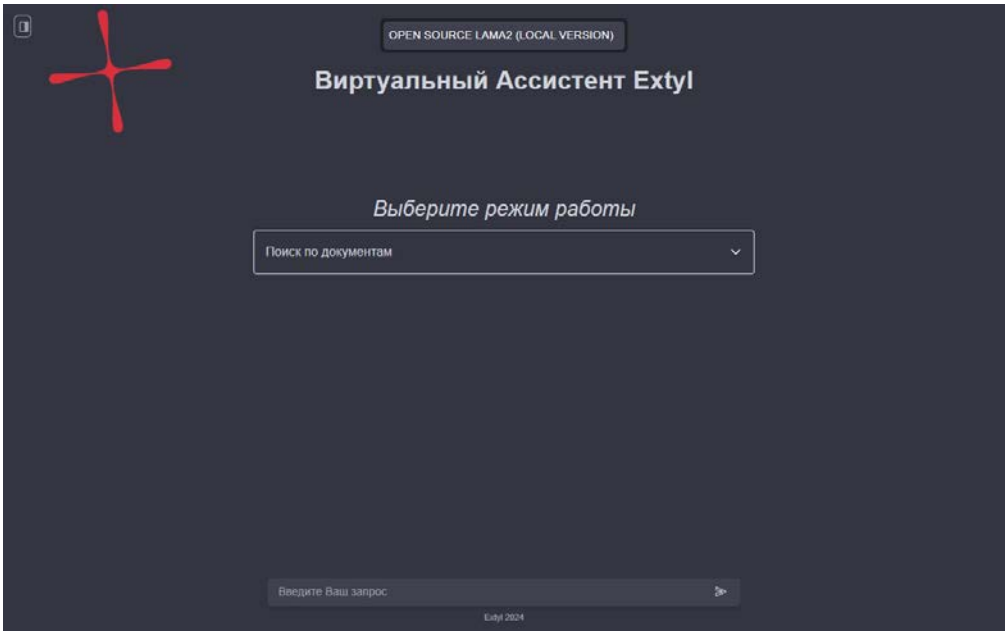

Рис. 1. Главное окно Web UI приложения

Назначение портов:

Порт	Назначение
5173	Web UI приложения (основной интерфейс пользователя)
5000	OpenAI-совместимый API (llama-server)
18000	RAG API (FastAPI — индексация и поиск по базе знаний)
5005	Streaming-порт

5. Развёртывание на сервере Vast.ai

Для развёртывания в облаке на базе сервиса Vast.ai выполните следующие шаги.

5.1. Подготовка образа

1. Загрузите образ контейнера в Docker (см. п. 4.1).

2. Опубликуйте образ в доступное через интернет хранилище — публичный Docker Hub или собственный Docker Registry:

```
docker tag desm0nt/base_webui_v5:latest
<ваш_репозиторий>/base_webui_v5:latest

docker push <ваш_репозиторий>/base_webui_v5:latest
```

5.2. Настройка шаблона на Vast.ai

1. Перейдите на страницу подбора сервера на Vast.ai.
2. Откройте редактирование шаблона образа (кнопка «Edit Image Config»).
3. Укажите наименование репозитория и версию окружения на Docker Hub.
4. Укажите команду запуска:

```
-e JUPYTER_DIR=/ -e OPEN_BUTTON_PORT=5173 -p 7860:7860 -p 5000:5000 -p
5005:5005 -p 5173:5173 -p 18000:18000
```

5. В поле On-start Script укажите entrypoint-скрипт и команду запуска:

```
env | grep _ >> /etc/environment; /scripts/docker-entrypoint.sh
/usr/bin/supervisord
```

6. Выберите и арендуйте сервер с подходящим GPU (не менее 12 ГБ видеопамяти).

5.3. Определение адресов и портов

ВАЖНО! При аренде на Vast.ai маппинг портов присваивает не указанные в команде порты, а произвольные из выделенного виртуальной машине диапазона.

Для определения фактического адреса и порта запущенного API:

1. Перейдите в панель управления арендованного сервера на Vast.ai.
2. Найдите раздел с информацией о проброшенных портах.
3. Зафиксируйте фактические адрес и порт для каждого из сервисов (Web UI — порт 5173, API — порт 5000, RAG — порт 18000).

6. Первоначальная настройка Web UI

После запуска контейнера и загрузки модели выполните первоначальную настройку Web UI.

1. Откройте браузер и перейдите по адресу Web UI.

Для локального сервера:

```
http://<IP-адрес_сервера>:5173
```

Для Vast.ai — используйте адрес и порт, определённые в п. 5.3.

2. Откройте боковую панель, нажав на иконку в левом верхнем углу экрана.

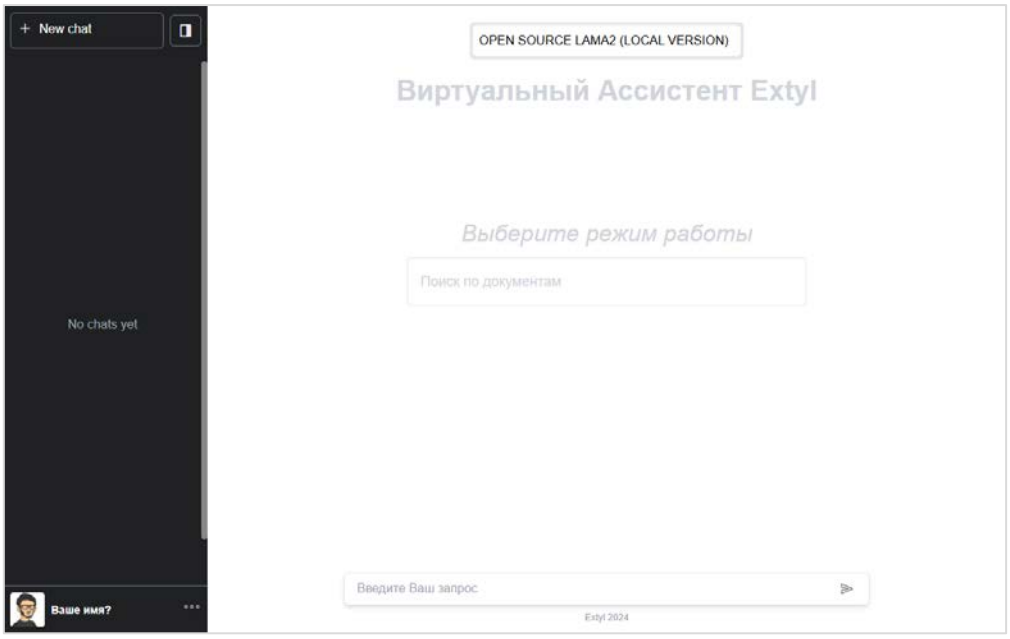

Рис. 2. Боковая панель с историей чатов и кнопкой Settings

3. Перейдите в раздел «Настройки».
4. Настройте параметры подключения:

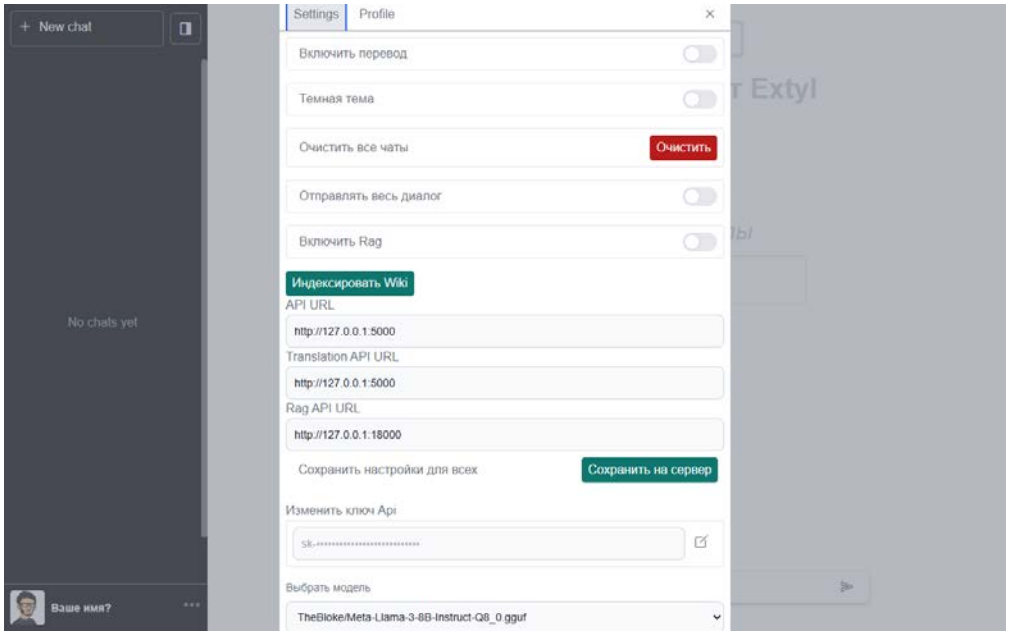

Рис. 3. Окно настроек Web UI

- **API URL** — адрес API основной модели (llama-server). Значение по умолчанию: *http://127.0.0.1:5000*. При работе на локальном сервере оставьте как есть. При удалённом доступе укажите IP-адрес сервера и порт 5000 (или фактический порт Vast.ai). Формат: *http://<IP-адрес>:<порт>*.
- **Translation API URL** — адрес API модели-переводчика. Значение по умолчанию: *http://127.0.0.1:5000* (используется та же модель). При отсутствии отдельной модели-переводчика оставьте значение по умолчанию.

- **RAG API URL** — адрес RAG-системы (FastAPI). Значение по умолчанию: *http://127.0.0.1:18000*. При работе на локальном сервере оставьте как есть. При удалённом доступе укажите фактический адрес и порт.
- **Отправлять весь диалог** — включите, если планируется задавать более одного вопроса в рамках одного диалога (сохраняет контекст беседы). Размер контекста модели: 16384 токена. По мере превышения объёма самые старые сообщения перестанут передаваться в модель.
- **Включить перевод** — включите при необходимости использования перевода запросов/ответов (по умолчанию выключен).
- **Включить RAG** — переключатель режима RAG. При включённом RAG запросы пользователя предварительно обогащаются релевантной информацией из проиндексированной базе знаний.

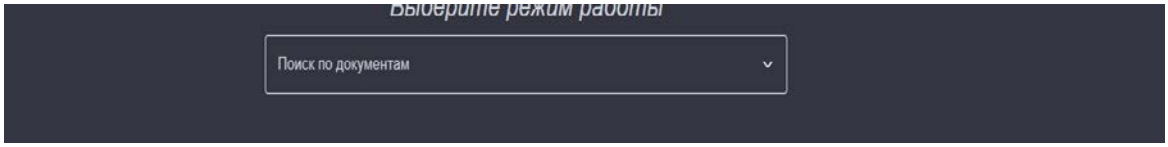

Рис. 4. Переключатель режима работы (RAG)

Примечание: Настройки «Ключ API» и «Выбрать модель» можно игнорировать — они предназначены для обеспечения совместимости с OpenAI API.

5. При необходимости сохранения настроек для всех пользователей нажмите кнопку «Сохранить на сервер». Это запишет текущие URL-адреса (API URL, Translation API URL, RAG API URL) в конфигурационный файл на сервере, и они станут значениями по умолчанию для всех новых подключений.
6. Закройте окно настроек.
7. **Обязательно обновите страницу** (F5) для применения настроек.
8. Введите сообщение в поле чата и отправьте его. Убедитесь, что модель генерирует ответ.

7. Индексация базы знаний (RAG)

Для работы RAG-системы необходимо предварительно проиндексировать базу знаний. Индексация создаёт векторное хранилище (ChromaDB) с содержимым источников данных, по которому осуществляется поиск при ответе на вопросы пользователей.

7.1. Индексация Wiki через Web UI

1. Откройте настройки Web UI (боковая панель).
2. Нажмите кнопку «Индексировать Wiki».
3. Дождитесь завершения индексации. Прогресс отображается в логах контейнера.

7.2. Индексация через API

Для программной индексации доступны следующие эндпоинты RAG API (порт 18000):

Эндпоинт	**Метод**	**Назначение**
/index_wiki_with_docs	POST	Индексация Wiki с документами
/index_wiki	POST	Индексация Wiki
/index_files	POST	Индексация файлов из указанной папки
/index_urls	POST	Индексация веб-страниц по списку URL

Пример вызова индексации Wiki:

```
curl -X POST http://<IP-адрес_сервера>:18000/index_wiki_with_docs
```

7.3. Проверка работы RAG

После индексации убедитесь, что RAG работает корректно:

1. Включите RAG в настройках Web UI (если не включён).
2. Задайте вопрос, ответ на который содержится в проиндексированной базе знаний.
3. Убедитесь, что ответ модели содержит информацию из базы знаний.

8. Проверка работоспособности

Для подтверждения корректности установки выполните следующие проверки.

№	**Проверка**	**Ожидаемый результат**
1	Docker-контейнер запущен (docker ps)	Контейнер в статусе Up
2	Web UI доступен по адресу :5173	Отображается интерфейс чата
3	Модель загружена (в логах docker появилось сообщение llama-server о запуске API)	Сообщение об успешном запуске llama-server в логах
4	Отправка тестового сообщения в чат	Модель генерирует корректный ответ
5	API доступен по :5000/v1/chat/completions (POST-запрос)	API возвращает JSON-ответ с результатом генерации
6	RAG API доступен по :18000 (POST /query)	RAG API возвращает JSON-ответ
7	Индексация Wiki выполнена (POST /index_wiki_with_docs)	Возвращается {"message": "Wiki indexed successfully"}

8	Вопрос с RAG возвращает ответ, обогащённый данными из базы знаний	Ответ содержит информацию из Wiki

9. Замена модели

Для замены модели, используемой llama-server, необходимо выполнить следующие действия.

1. Поместите новый файл модели в формате GGUF в директорию */app/bin/* внутри docker-контейнера. Для этого можно использовать docker cp:

   ```
   docker cp <путь_к_модели>.gguf <container_id>:/app/bin/
   ```

2. Отредактируйте файл конфигурации supervisord внутри контейнера:

   ```
   docker exec -it <container_id> vi /etc/supervisor/conf.d/supervisord.conf
   ```

3. В секции [program:model] измените параметр -m в строке command, указав имя нового файла модели:

   ```
   command=/app/bin/llama-server -m <имя_новой_модели>.gguf --port 5000 -c
   16384 -np 2 -ngl 20000 -fa --host 0.0.0.0
   ```

Описание ключевых параметров llama-server:

- *-m* — имя файла модели (в формате GGUF);
- *-c 16384* — размер контекста в токенах (каждые ~1000 токенов контекста потребляют дополнительно ~1 ГБ видеопамяти);
- *-np 2* — количество параллельных слотов обработки;
- *-ngl 20000* — количество слоёв, выгружаемых на GPU (большое значение = вся модель на GPU);
- *-fa* — включение flash attention (ускоряет работу);
- *--host 0.0.0.0* — прослушивание всех интерфейсов.

4. Перезапустите сервис модели:

   ```
   docker exec -it <container_id> supervisorctl restart model
   ```

   Примечание: Для постоянного применения изменений рекомендуется пересобрать docker-контейнер, изменив файл supervisord.conf и поместив новую модель в директорию llama/ перед сборкой.

10. Возможные проблемы и их решение

Проблема	**Решение**
Web UI не открывается	Убедитесь, что контейнер запущен (docker ps). Проверьте правильность указанного адреса и порта. Дождитесь полной инициализации контейнера (может занять несколько минут при первом запуске).

Модель не генерирует ответ	Проверьте логи контейнера (docker logs <container_id>). Убедитесь, что llama-server полностью загрузил модель — в логах должно быть сообщение о готовности API. Проверьте настройку API URL в Web UI.
RAG не возвращает релевантных данных	Убедитесь, что индексация базы знаний выполнена (кнопка «Индексировать Wiki» в настройках или POST-запрос на /index_wiki_with_docs). Проверьте логи RAG-сервиса в docker. Убедитесь, что RAG API URL в настройках указан верно.
Низкая скорость генерации ответов	Убедитесь, что модель загружена на GPU (параметр -ngl в supervisord.conf должен быть достаточно большим).
Контейнер останавливается сразу после запуска (OOM)	Недостаточно оперативной или видеопамяти. Проверьте соответствие системным требованиям (п. 1). Уменьшите параметр -ngl в supervisord.conf для частичной выгрузки модели на CPU. Уменьшите размер контекста (-c).
Ошибка npm install или npm run build-css при инициализации	Проверьте наличие доступа к интернету из контейнера. Node.js и пакеты скачиваются при первом запуске. Убедитесь, что DNS-резолвинг работает внутри контейнера.