**## Introduction**

DocuChat provides an **\*\*API\*\*** containing all the building blocks required to build

**\*\*private, context-aware AI applications\*\***. The API follows and extends OpenAI API standard, and supports

both normal and streaming responses.

The API is divided in logical blocks:

- High-level API, abstracting all the complexity of a RAG (Retrieval Augmented Generation) pipeline implementation:

    - Ingestion of documents: internally managing document parsing, splitting, metadata extraction,

      embedding generation and storage.

    - Chat & Completions using context from ingested documents: abstracting the retrieval of context, the prompt

      engineering and the response generation.

> A working **\*\*Gradio UI client\*\*** is provided to test the API.

**## Quick Local Installation steps**

The steps in `Installation and Settings` section are better explained and cover more

setup scenarios. But if you are looking for a quick setup guide, here it is:

```

# Clone the repo

git clone https://github.com/safhash/DocuChat

cd DocuChat

# Install Python 3.11

pyenv install 3.11

pyenv local 3.11

# Install dependencies

poetry install --with ui,local

# Download Embedding and LLM models

poetry run python scripts/setup

# Navigate to the UI and try it out!

http://localhost:8001/

```

**## Installation and Settings**

**### Base requirements to run DocuChat**

\* Git clone DocuChat repository, and navigate to it:

```

  git clone https://github.com/safhash/DocuChat

  cd DocuChat

```

\* Install Python 3.11. Ideally through a python version manager like `pyenv`.

  Python 3.12

  should work too. Earlier python versions are not supported.

    \* osx/linux: [pyenv](https://github.com/pyenv/pyenv)

    \* windows: [pyenv-win](https://github.com/pyenv-win/pyenv-win)

```

pyenv install 3.11

pyenv local 3.11

```

\* Install [Poetry](https://python-poetry.org/docs/#installing-with-the-official-installer) for dependency management:

\* Have a valid C++ compiler like gcc. See [Troubleshooting: C++ Compiler](#troubleshooting-c-compiler) for more details.

\* Install `make` for scripts:

    \* osx: (Using homebrew): `brew install make`

    \* windows: (Using chocolatey) `choco install make`

**### Install dependencies**

Install the dependencies:

```bash

poetry install --with ui

```

Verify everything is working by running `make run` (or `poetry run python -m docu\_chat`) and navigate to

http://localhost:8001. You should see a [Gradio UI](https://gradio.app/) **\*\*configured with a mock LLM\*\*** that will

echo back the input. Later we'll see how to configure a real LLM.

**### Settings**

> Note: the default settings of DocuChat work out-of-the-box for a 100% local setup. Skip this section if you just

> want to test DocuChat locally, and come back later to learn about more configuration options.

DocuChat is configured through *\*profiles\** that are defined using yaml files, and selected through env variables.

The full list of properties configurable can be found in `settings.yaml`

**#### env var `PGPT\_SETTINGS\_FOLDER`**

The location of the settings folder. Defaults to the root of the project.

Should contain the default `settings.yaml` and any other `settings-{profile}.yaml`.

**#### env var `PGPT\_PROFILES`**

By default, the profile definition in `settings.yaml` is loaded.

Using this env var you can load additional profiles; format is a comma separated list of profile names.

This will merge `settings-{profile}.yaml` on top of the base settings file.

For example:

`PGPT\_PROFILES=local,cuda` will load `settings-local.yaml`

and `settings-cuda.yaml`, their contents will be merged with

later profiles properties overriding values of earlier ones like `settings.yaml`.

During testing, the `test` profile will be active along with the default, therefore `settings-test.yaml`

file is required.

**#### Environment variables expansion**

Configuration files can contain environment variables,

they will be expanded at runtime.

Expansion must follow the pattern `${VARIABLE\_NAME:default\_value}`.

For example, the following configuration will use the value of the `PORT`

environment variable or `8001` if it's not set.

Missing variables with no default will produce an error.

```yaml

server:

  port: ${PORT:8001}

```

**### Local LLM requirements**

Install extra dependencies for local execution:

```bash

poetry install --with local

```

For DocuChat to run fully locally GPU acceleration is required

(CPU execution is possible, but very slow), however,

typical Macbook laptops or window desktops with mid-range GPUs lack VRAM to run

even the smallest LLMs. For that reason

**\*\*local execution is only supported for models compatible with [llama.cpp](**https://github.com/ggerganov/llama.cpp**)\*\***

These two models are known to work well:

\* https://huggingface.co/TheBloke/Llama-2-7B-chat-GGUF

\* https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.1-GGUF (recommended)

To ease the installation process, use the `setup` script that will download both

the embedding and the LLM model and place them in the correct location (under `models` folder):

```bash

poetry run python scripts/setup

```

If you are ok with CPU execution, you can skip the rest of this section.

As stated before, llama.cpp is required and in

particular [llama-cpp-python](https://github.com/abetlen/llama-cpp-python)

is used.

> It's highly encouraged that you fully read llama-cpp and llama-cpp-python documentation relevant to your platform.

> Running into installation issues is very likely, and you'll need to troubleshoot them yourself.

**#### Windows NVIDIA GPU support**

Windows GPU support is done through CUDA.

Follow the instructions on the original [llama.cpp](https://github.com/ggerganov/llama.cpp) repo to install the required

dependencies.

Some tips to get it working with an NVIDIA card and CUDA (Tested on Windows 10 with CUDA 11.5 RTX 3070):

\* Install latest VS2022 (and build tools) https://visualstudio.microsoft.com/vs/community/

\* Install CUDA toolkit https://developer.nvidia.com/cuda-downloads

\* Verify your installation is correct by running `nvcc --version` and `nvidia-smi`, ensure your CUDA version is up to

  date and your GPU is detected.

\* [Optional] Install CMake to troubleshoot building issues by compiling llama.cpp directly https://cmake.org/download/

If you have all required dependencies properly configured running the

following powershell command should succeed.

```powershell

$env:CMAKE\_ARGS='-DLLAMA\_CUBLAS=on'; poetry run pip install --force-reinstall --no-cache-dir llama-cpp-python

```

If your installation was correct, you should see a message similar to the following next

time you start the server `BLAS = 1`.

```

llama\_new\_context\_with\_model: total VRAM used: 4857.93 MB (model: 4095.05 MB, context: 762.87 MB)

AVX = 1 | AVX2 = 1 | AVX512 = 0 | AVX512\_VBMI = 0 | AVX512\_VNNI = 0 | FMA = 1 | NEON = 0 | ARM\_FMA = 0 | F16C = 1 | FP16\_VA = 0 | WASM\_SIMD = 0 | BLAS = 1 | SSE3 = 1 | SSSE3 = 0 | VSX = 0 |

```

Note that llama.cpp offloads matrix calculations to the GPU but the performance is

still hit heavily due to latency between CPU and GPU communication. You might need to tweak

batch sizes and other parameters to get the best performance for your particular system.

```

**#### Known issues and Troubleshooting**

Execution of LLMs locally still has a lot of sharp edges, specially when running on non Linux platforms.

You might encounter several issues:

\* Performance: RAM or VRAM usage is very high, your computer might experience slowdowns or even crashes.

\* GPU Virtualization on Windows and OSX: Simply not possible with docker desktop, you have to run the server directly on

  the host.

\* Building errors: Some of DocuChat dependencies need to build native code, and they might fail on some platforms.

  Most likely you are missing some dev tools in your machine (updated C++ compiler, CUDA is not on PATH, etc.).

  If you encounter any of these issues, please open an issue and we'll try to help.

**#### Troubleshooting: C++ Compiler**

If you encounter an error while building a wheel during the `pip install` process, you may need to install a C++

compiler on your computer.

**\*\*For Windows 10/11\*\***

To install a C++ compiler on Windows 10/11, follow these steps:

1. Install Visual Studio 2022.

2. Make sure the following components are selected:

    \* Universal Windows Platform development

    \* C++ CMake tools for Windows

3. Download the MinGW installer from the [MinGW website](https://sourceforge.net/projects/mingw/).

4. Run the installer and select the `gcc` component.

\*\* For OSX \*\*

1. Check if you have a C++ compiler installed, Xcode might have done it for you. for example running `gcc`.

2. If not, you can install clang or gcc with homebrew `brew install gcc`

**#### Troubleshooting: Mac Running Intel**

When running a Mac with Intel hardware (not M1), you may run into \_clang: error: the clang compiler does not support '

-march=native'\_ during pip install.

If so set your archflags during pip install. eg: \_ARCHFLAGS="-arch x86\_64" pip3 install -r requirements.txt\_

**## Running the Server**

After following the installation steps you should be ready to go. Here are some common run setups:

**### Running 100% locally**

Make sure you have followed the *\*Local LLM requirements\** section before moving on.

This command will start DocuChat using the `settings.yaml` (default profile) together with the `settings-local.yaml`

configuration files. By default, it will enable both the API and the Gradio UI. Run:

```

PGPT\_PROFILES=local make run

```

or

```

PGPT\_PROFILES=local poetry run python -m docu\_chat

```

When the server is started it will print a log *\*Application startup complete\**.

Navigate to http://localhost:8001 to use the Gradio UI or to http://localhost:8001/docs (API section) to try the API

using Swagger UI.

**## Gradio UI user manual**

Gradio UI is a ready to use way of testing most of DocuChat API functionalities.

![Gradio DocuChat](https://lh3.googleusercontent.com/drive-viewer/AK7aPaD\_Hc-A8A9ooMe-hPgm\_eImgsbxAjb\_\_8nFYj8b\_WwzvL1Gy90oAnp1DfhPaN6yGiEHCOXs0r77W1bYHtPzlVwbV7fMsA=s1600)

**### Execution Modes**

It has 3 modes of execution (you can select in the top-left):

\* Query Docs: uses the context from the

  ingested documents to answer the questions posted in the chat. It also takes

  into account previous chat messages as context.

    \* Makes use of `/chat/completions` API with `use\_context=true` and no

      `context\_filter`.

\* Search in Docs: fast search that returns the 4 most related text

  chunks, together with their source document and page.

    \* Makes use of `/chunks` API with no `context\_filter`, `limit=4` and

      `prev\_next\_chunks=0`.

\* LLM Chat: simple, non-contextual chat with the LLM. The ingested documents won't

  be taken into account, only the previous messages.

    \* Makes use of `/chat/completions` API with `use\_context=false`.

**### Document Ingestion**

Ingest documents by using the `Upload a File` button. You can check the progress of

the ingestion in the console logs of the server.

The list of ingested files is shown below the button.

If you want to delete the ingested documents, refer to \*Reset Local documents

database\* section in the documentation.

**### Chat**

Normal chat interface, self-explanatory ;)

You can check the actual prompt being passed to the LLM by looking at the logs of

the server. We'll add better observability in future releases.

**## Ingesting & Managing Documents**

🚧 Document Update and Delete are still WIP. 🚧

The ingestion of documents can be done in different ways:

\* Using the `/ingest` API

\* Using the Gradio UI

\* Using the Bulk Local Ingestion functionality (check next section)

**### Reset Local documents database**

When running in a local setup, you can remove all ingested documents by simply

deleting all contents of `local\_data` folder (except .gitignore).

To simplify this process, you can use the command:

```bash

make wipe

```