Getting Started

The ***EvidenceSeeker Boilerplate*** is in an early stage of development. Currently, we offer:

1. The ***EvidenceSeeker* Demoapp:** An *EvidenceSeeker* instance based on all 2024 APuZ editions as knowledge base with a minimalistic user interface.
2. The **`evidence-seeker` Python package:** The *EvidenceSeeker Boilerplate* is a Python package, which can be used to build your own *EvidenceSeeker* instance.

For subsequent releases, we are working on (hosted) versions of the boilerplate, which should make the setup and integration of *EvidenceSeeker* instances even easier.

EvidenceSeeker Demoapp

We have set up a small *EvidenceSeeker* with all 2024 APuZ editions as knowledge base and provide access to this *EvidenceSeeker* via a small Gradio app, which provides a minimal user interface. Links:

- Our APuZ-EvidenceSeeker Demoapp: If you are interested in experimenting with this Demapp, contact us for access information!
- EvidenceSeeker Demoapp Logs: This is a collection of fact-checks and their results performed by our Demoapp. You can use this to get an idea of the capabilities of the *EvidenceSeeker* Demoapp.

Side note: You can also run the *EvidenceSeeker* Demoapp locally if you want to experiment with your knowledge base or if you want to use a different language model.

The `evidence-seeker` Python package

The `evidence-seeker` Python package is available on PyPI. Follow the subsequent steps to set up your own *EvidenceSeeker* instance.

1. Prerequisites

You need an installed Python (3.11 or 3.12) and `pip`.

1. *Installing Python:* You can choose between different ways to install Python, depending on your operating system. You can find instructions for installing Python, for instance,
   - on the Python wiki or
   - in the Real-Python installation guide.
2. *Installing `pip`:* If you have installed Python, you should also have `pip` installed. You can check this by running `pip --version` in your terminal. If you do not have `pip` installed, you can find instructions on installing it here.

2. Installation of the `evidence-seeker` package

Open a terminal and use `pip` to install the `evidence-seeker` package from PyPI:

```
pip install evidence-seeker
```

3. Preparation

Generating a directory structure might be helpful before you begin configuring your *Evidence-Seeker* instance. There, you can put your configuration files, the knowledge base, the index, and possibly other resources. This is not strictly necessary as long as you specify the different locations in the configuration files.

The *EvidenceSeeker* boilerplate comes with a command line interface—the `evse` cli—that can create a directory structure for your *EvidenceSeeker* instance. Calling the `evse` cli with:

```
evse init <name_of_your_evidence_seeker>
```

will create the following directory structure in the current working directory, and will contain configuration files with default values.

```
name_of_your_evidence_seeker/
   config/ # Directory for configuration files
      preprocessor.yaml
      retriever.yaml
      confirmation_analysis.yaml
      api_keys.txt # File for API keys
   knowledge_base/
```

```
      metadata.json # Metadata for the knowledge base files
      data_files/ # Directory for the knowledge base files (e.g., PDF files)
          file1.pdf
          file2.pdf
          ...
  logs/ # Directory for logging
  embeddings/ # Directory for the index
```

4. Configuration

There are various ways to configure your *EvidenceSeeker* instance to your needs—either in your Python code or via YAML configuration files. At least, you have to specify the language models used during the different steps of the *EvidenceSeeker* pipeline and the indexed knowledge base used for fact-checking. For all other settings, sensible defaults allow starting with a minimal configuration.

For details, see the Configuration section.

5. Building the index

EvidenceSeeker instances fact-check statements relative to a specified knowledge base. The knowledge base can, for instance, comprise a set of PDF files. For an *EvidenceSeeker* instance to work, the knowledge base must be represented by a created index of the knowledge. More technically, an *EvidenceSeeker* works on created embeddings (a vector-based representation) of the knowledge base. Accordingly, you have to create such an index using an embedding model.

If you used `evse init` to create the directory structure, you can use the `evse` CLI to create an index in the following way:

1. Copy all PDF files you want to use as knowledge base into the `knowledge_base/data_files` directory of your *EvidenceSeeker* instance.
2. If you want to provide the fact checker with metadata for the files in your knowledge base, create a file `meta_data.json` in the `knowledge_base/` directory that contains metadata for each PDF file. The metadata should be in JSON format and can include fields like `title`, `author`, `date`, etc. For example:

```
{
    "file1.pdf": {
        "title": "Title of File 1",
        "author": "Author of File 1",
        "date": "2024-01-01"
```

```
 },
    "file2.pdf": {
        "title": "Title of File 2",
        "author": "Author of File 2",
        "date": "2024-02-01"
 }
}
```

3. In your terminal, change into the directory of your *EvidenceSeeker* instance, e.g., `cd name_of_your_evidence_seeker/`.
4. Ensure you have set the environment variables for your API keys in the file `config/api_keys.txt` of your *EvidenceSeeker* instance.
5. Now you can build the index using `evse build-index`. This will create an index of the PDF files in the `knowledge_base/data_files` directory and store it in the `embeddings/` directory. The index will be created using the embedding model specified in the configuration file.

Alternatively, you can use and adapt the following Python snippet to create an index:

```
from evidence_seeker import RetrievalConfig, IndexBuilder

config = RetrievalConfig(
    ### Using local embedding model (via Huggingface API)
    embed_backend_type="huggingface",
    embed_model_name="sentence-transformers/paraphrase-multilingual-mpnet-base-v2",
    # Path to the directory containing your PDF files
    document_input_dir="path/to/your/pdf_files",
    # Path where the index will be stored
    index_persist_path="path/to/your/index",
)
index_builder = IndexBuilder(config=config)
index_builder.build_index()
```

For further details on configuring the EvidenceSeeker Retriever component, see here and here.

6. Executing the Pipeline

Once you built the index, you can run the *EvidenceSeeker* pipeline to fact-check statements against the knowledge base.

With the `evse` cli, you can run the pipeline with:

```
evse run --input "you statement to fact-check"
```

The output will be written as a Markdown file to the `logs/` directory of your *EvidenceSeeker* instance.

You can also specify the ouput file with the `--output` option:

```
evse run --input "you statement to fact-check" --output "path/to/output/file.md"
```

Alternatively, you can run the pipeline with the following Python snippet:

```
from evidence_seeker import EvidenceSeeker
import asyncio

pipeline = EvidenceSeeker(
    retrieval_config_file="path/to/retrieval_config.yaml",
    confirmation_analysis_config_file="path/to/confirmation_analysis_config.yaml",
    preprocessing_config_file="path/to/preprocessing_config.yaml",
)
# run the pipeline
results = asyncio.run(pipeline("your statement to fact-check"))
```

7. Integration into Existing Workflows

Currently, there are two ways to integrate *EvidenceSeeker* into your existing workflows:

1. Programmatically integrating your EvidenceSeeker instance by using the `evidence-seeker` package, or
2. by exposing the *EvidenceSeeker Demoapp* as an MCP server (for details, see this HF blog post).

Development Version

If you want to have more control over your *EvidenceSeeker* instance or if you want to implement an additional feature, you can use the development version of the ***EvidenceSeeker Boilerplate*** by git-cloning the EvidenceSeeker repository:

```
git clone git@github.com:debatelab/evidence-seeker.git
```

By default, we use hatch for Python package and environment management. The repository contains the description of a development environment with pinned dependencies. You can create and spawn a corresponding Python environment with

```
hatch -v shell evse-dev_env.py3.11
```

or

```
hatch -v shell evse-dev_env.py3.12
```