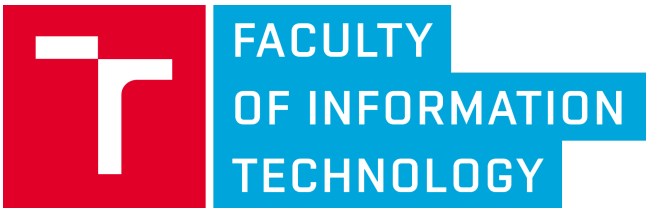

Technická Dokumentace k Software Textjuicer - Software pro generování zkrácených popisů textů

Martin Dočekal, Martin Fajčík, Michal Hradiš

Vysoké učení technické v Brně Brno
2024

Tento dokument byl vytvořen s finanční podporou MK ČR v rámci programu **NAKI III program na podporu aplikovaného výzkumu v oblasti národní a kulturní identity na léta 2023 až 2030**

v projektu semANT - Sémantický průzkumník textového kulturního dědictví.

Číslo a název projektu:

DH23P03OVV060	semANT - Sémantický průzkumník textového kulturního dědictví

Název a popis dílčího výstupu:

Textjuicer - Software pro generování zkrácených popisů textů
Tento dokument popisuje funkčnost a použití software Textjuicer, který umožňuje sumarizovat texty pomocí speciálně natrénovaných velkých jazykových modelů, volně dostupných obecných jazykových modelů i pomocí modelů dostupných přes komerční API.

Jazyk dokumentu

Angličtina

Organizace a řešitel

Vysoké učení technické v Brně	Ing. Michal Hradiš Ph.D.

Usage overview

This software focuses on text summarization with the emphasis on summarization of query-based retrieval results. Whereas standard summarization condenses the most important information from a text, this software includes an option to efficiently summarize a number text fragments and to extract information relevant to a user query. These summarization strategies are targeted specifically to document retrieval applications.

The software provides a unified API for summarization in the form of a python package and a REST API. These interfaces hide several summarization options: either on OpenAI models, general-purpose open source models or task specific smaller models.

The software includes a number of summarization configurations with optimized prompts and smaller models specifically trained for summarization.

Availability

The software is available from https://github.com/DCGM/semant-summarization and summarization models (https://huggingface.co/BUT-FIT/csmpt-7B-RAGsum, https://huggingface.co/BUT-FIT/CSTinyLLama-1.2B-RAGsum) are available from Huggingface repository. The git repository includes:

- Python package semantsum
- REST API server
- Docker and Docker compose files for easy deployment

Python distribution package can be installed directly from the git repository by running:

```
pip install -e
git+https://github.com/DCGM/semant-summarization.git#egg=semantsum
```

The summarization models are available in two versions:

- Compatible with library Transformers (Hugging Face)
- Quantized version for efficient and fast inference in Ollama inference server (https://ollama.com/download)

Links to all available models are available in the README.md file.

License

Software: BSD 3-Clause License

Custom models: Various permissive licenses

Requirements overview

Python in version 3.10 or higher is required alongside a number of standard packages. These dependencies are listed in requirements.txt files in the git repository.

API deployment using Docker requires standard Docker Engine and Docker Compose installation https://docs.docker.com/engine/install/ubuntu/.

Using OpenAI models requires a valid OpenAI API key https://platform.openai.com/

Local model inference using Transformers library requires this library to be installed together with PyTorch, supported inference hardware (most likely nVidia GPU) and relevant drivers. For installation instructions see the official documentation (https://huggingface.co/docs/transformers/installation, https://pytorch.org/get-started/locally/)

Ollama provides much faster and more memory efficient model inference. Ollama can be installed on Windows, iOS and Linux. List of compatible hardware accelerators can be found in https://github.com/ollama/ollama/blob/main/docs/gpu.md

Hardware requirements for local model inference depends on the size of the models. In general, 1 billion and 3 billion models can use any current GPU. 7 billion models require high-end GPU with 16 GB RAM or more. Larger models may require multiple GPUs.

Summarization Types

This section describes the types of summaries that are implemented in TextJuicer. We distinguish summary types according to the information given at input and the form of summary at the output.

All described types were selected to enrich the historical document search. Even though the form of these summary types is not strictly limited to this usage, our fine-tuned models were trained on document search data, and thus, using those models in different domains is discouraged.

Single document summarization

The simplest type of summary that summarizes one input document into a couple of sentences. In the domain of document search, the input document is a single search result.

Query-Based Multi-Document Summarization

This type of summarization creates a summary from multiple documents relevant to a given query. Thus e.g., for document search, we have top-k search results and the user query used to find those documents, and we provide a summary of those documents with respect to the search query.

This form helps the user to search for relevant information without the need to go through search results.

We distinguish three forms of this type of summary according to output format. The first one is in the form of unformatted plain text. We also use summaries in the form of timelines and topics. Those two are described in the following sections.

Timeline

The output is in the form of a list of time events and a summary of each event.

Example output:

```
- **19. dubna 1945**: MUDr. Vladimír Tůma, docent Karlovy university a České techniky v Praze, byl v Praze na Pankráci ubit gestapáky. Jeho aktivity v odboji zahrnovaly uskutečňování spojení se zahraničím a pomoc pronásledovaným a důstojníkům československé armády [3].

- **8. května 1945**: V den kapitulace Německa ustupovala německá vojska z Brna směrem na jih a situace se stala chaotickou, kdy vojska zapalovala vozy s municí, což ohrožovalo blízké domy v Přibyslavi [7].

- **13. ledna 1946**: V Rakousku bylo nalezeno dalších pět československých dětí, které byly během války odvlečeny. Tím se zvýšil jejich celkový počet na 45. Dvě z dětí budou brzy repatriovány, včetně pětileté Helgy Seibtové [12].
```

Topics

When using this type of summary, the output is in the form of a list of topics in input documents and a brief text summarizing content for a given topic.

Example:

```
1. **Odpory a hrdinství jednotlivců**

  Tento tematický okruh se zaměřuje na jednotlivce, jako je Fr. Katzer a MUDr. Vladimír Tůma, kteří se během okupace angažovali v obraně českých občanů a v odboji, přičemž využívali různých způsobů, jak pomoci pronásledovaným.

2. **Konflikty a situace na konci války**

  Dalším tématem jsou události v České republice v závěru druhé světové války, konkrétně chaos a násilí, které provázely ústup německých vojsk až k jejich kapitulaci.

3. **Osvobozování Československa**

   Tento okruh se zabývá účastí sovětských a rumunských vojsk na osvobozování Moravy a přítomností československých sil v SSSR, které hrály klíčovou roli v boji proti nacismu.
```

Models

To perform summarization, we use our trained models and also the third-party OpenAI model GPT-4o mini.

In contrast to the general purpose GPT-4o mini model, we have trained smaller models dedicated to summarization that could be used locally.

Fine-tuned Models

We have trained the 7 billion parameters MPT model and the 1.2 billion parameters TinyLLama model for plain text query-based multi-document summarization.

For fine-tuning, we used our distillation dataset created from GPT-4o mini summaries based on the results of historical document searches.

Our fine-tuned models are available under the BUT-FIT organization at https://huggingface.co/BUT-FIT. For example, our summarization MPT model is available at https://huggingface.co/BUT-FIT/csmpt-7B-RAGsum, and the TinyLLama is available at https://huggingface.co/BUT-FIT/CSTinyLlama-1.2B. More information about models can be found at README.md.

Python Package

We created a Python package for summarization that is available at https://github.com/DCGM/semant-summarization. It was designed to allow the easy addition of new models, and it also allows the use of OpenAI-based APIs. Therefore, it is also possible to use self-hosted APIs like Ollama.

Configuration

The package is built in a way that makes every model fully loadable from a configuration file. The configuration file is a YAML file that contains all the necessary information to load the model.

You can use *create_config* argument to create a configuration file using the configuration builder:

```
./run.sh create_config path_to_config.yaml
```

How to add a new summarization model

For adding an API based summarization model the addition of a new model is as easy as creating a new configuration file and saving it in the *config* directory.

However, if you need to create a brand-new class for your model, read further.

The module *summarizer.py* contains abstract base classes for several types of summaries. If there is a new summarization model that does not fit into any of the existing classes, a new class can be added to the module. The new class should inherit from the *Summarizer* class.

To see an example of how to add a new model, you can inspect the *openai.py* module.

Also, check the *create_config* method in the *__main__.py* and *init_summarizer* in the *summarizator_factory.py* module to make your new model compatible with the configuration builder and loader.

Demonstrator

There is a simple command line demonstrator showing how to use API based summarization models.

To run the OpenAI demonstrator, use the *run_summarizer* argument. You can use one of the configuration files from the *config* directory.

For these, you also need to configure the API. The API configuration could be provided in the model configuration or using *--api_config* parameter to specify a separate configuration file for the API.

```
./run.py run_summarizer semantsum/config/openai_query.yaml --api_config
openai_api.yaml
```

To see an example of a structured (non-plaintext) summary, you can try:

```
./run.py   run_summarizer   semantsum/config/openai_query_timeline.yaml
--api_config openai_api.yaml
```

List available models

There is an argument for listing available models:

```
./run.py available
```

An example output:

```
openai_query (openai)
openai_query_timeline (openai)
openai_query_topic (openai)
openai_single (openai)
```

Requirements

The package requires Python 3.10 and newer. All necessary software dependencies are listed in the requirements.txt file. There is no need for GPU accelerators for third-party APIs. However, when using self-hosted APIs like Ollama, it is advised to use GPUs.

REST API

The REST API provides simple encapsulation of the python package. It provides three endpoints `/api/summary/summarizers` which lists the available summarization types and two summarization endpoints `/api/summary/string` and `/api/summary/struct`. The source code is located in directory /api in the git repository.

Run API server

1. Create python virtual environment with python >3.10
2. Install dependencies by `pip install -r ./api/requirements.txt`
3. Set environment variables
    a. For the use of OpenAI models `OPENAI_KEY`. If you need non-standard url for OpenAI API, define `OPENAI_URL`.
    b. For Ollama inference, specify `OLLAMA_URL` and `OLLAMA_KEY`.
4. Run the server from ./api: `python run.py`
    a. By default, the server listens on all network interfaces on port 8000.
    b. You can change the network interfaces and port by editing the `./api/run.py` file.
5. The REST API endpoint documentation is provided by the running server at `/docs`. The documentation is automatically generated from the source code.

Docker deployment

Docker provides the most convenient way of deployment. All relevant files are located in the directory `./api.deploy`. Run the docker image by:

1. Install Docker Engine and Docker Compose
2. Change directory to `./api.deploy`
3. Copy the provides example of environment file `.env.example` to `.env` and fill the relevant variable values:
    a. For the use of OpenAI models `OPENAI_KEY`. If you need a non-standard url for OpenAI API, define `OPENAI_URL`.
    b. For Ollama inference, specify `OLLAMA_URL` and `OLLAMA_KEY`.
4. Build and run the docker compose service by: `./update.sh build`