

TextSynth Server

Version: 2024-01-20

Table of Contents

1	Introduction	1
2	Quick Start	2
2.1	Linux	2
2.1.1	First steps	2
2.1.2	GPU usage	2
2.2	Windows	3
2.2.1	First steps	3
2.2.2	GPU usage	4
3	Utilities	5
3.1	Text processing (<code>ts_test</code>)	5
3.1.1	Text generation	5
3.1.2	Translation	5
3.1.3	Short text compression and decompression	5
3.1.4	Perplexity computation	5
3.2	Text to image (<code>ts_sd</code>)	5
3.3	Chat (<code>ts_chat</code>)	6
3.4	Speech to text transcription (<code>ts_audiototext</code>)	6
3.5	Text compression (<code>ts_zip</code>)	6
3.6	Model Weight Conversion	6
3.7	Model Weight Quantization	7
4	TS Server Configuration	8
4.1	Syntax	8
4.2	Configuration parameters	8
4.3	JSON endpoints	9

1 Introduction

TextSynth Server is a web server proposing a REST API to large language models. They can be used for example for text completion, question answering, classification, chat, translation, image generation, ...

It has the following characteristics:

- All is included in a single binary. Very few external dependencies (Python is not needed) so installation is easy on Linux and Windows.
- Supports many Transformer variants (GPT-J, GPT-NeoX, OPT, Fairseq GPT, M2M100, CodeGen, GPT2, T5, RWKV, Llama 2, Falcon, MPT, Mistral, Whisper) and Stable Diffusion.
- Integrated REST JSON API for text completion, translation, image generation and audio transcription.
- Integrated HTML GUI for testing.
- Very high performance for small and large batches on CPU and GPU. Support of dynamic batching to handle a large number of simultaneous requests.
- Efficient custom 8, 4 and 3 bit quantization.
- Larger models work optimally on lower cost GPUs (e.g. RTX 3090, RTX A6000) thanks to efficient quantization.
- Support of speculative sampling for even faster inference.
- Support of grammar based sampling to constraint the model output according to a BNF grammar or a JSON schema.
- Uses the LibNC library for simple tensor manipulation using the C language.
- Simple command line tools (`ts_test`, `ts_sd`, `ts_chat`, `ts_audiototext` and `ts_zip`) are provided to test the various models.

The free version is available only for non commercial use. Commercial organizations must buy the commercial version. The commercial version adds the following features:

- High performance handling of simultaneous requests.
- Memory usage limitation.
- Multiple models can be served at the same time.
- SSL/TLS requests.

2 Quick Start

2.1 Linux

2.1.1 First steps

The TextSynth Server works only on x86 CPUs supporting AVX2 (all Intel CPUs since 2013 support it). The installation was tested on Fedora and CentOS/RockyLinux 8 distributions. Other distributions should work provided the `libjpeg` and `libmicrohttpd` libraries are installed.

1. Install the `libjpeg` and `libmicrohttpd` libraries. If you use Fedora, RHEL, CentOS or RockyLinux, you can type as root:

```
dnf install libjpeg libmicrohttpd
```

`ts_test` can be used without these libraries. `ts_sd` needs `libjpeg`. `ts_server` needs `libjpeg` and `libmicrohttpd`. Audio transcription requires the FFmpeg executable in order to convert the input audio file.

2. Extract the archive and go into its directory:

```
tar xtf ts_server-##version##.tar.gz
```

```
cd ts_server-##version##
```

when `##version##` is the version of the program.

3. Download one small example model such as `gpt2_117M.bin` from the `ts_server` web page.
4. Use it to generate text with the "ts_test" utility:

```
./ts_test -m gpt2-117M.bin g "The Linux kernel is"
```

The `-T` option can be used to use more or less CPU cores (the default is the number of physical cores).

5. Start the server:

```
./ts_server ts_server.cfg
```

You can edit the `ts_server.cfg` JSON configuration file if you want to use another model.

6. Try one request:

```
curl http://localhost:8080/v1/engines/gpt2_117M/completions \
-H "Content-Type: application/json" \
-d '{"prompt": "The Linux kernel is", "max_tokens": 100}'
```

The full request syntax is documented at <https://textsynth.com/documentation.html>.

7. You can use the integrated GUI by exploring with your browser:

```
http://localhost:8080
```

Now you are ready to load a larger model and to use it from your application.

2.1.2 GPU usage

You need an Nvidia Ampere, ADA or Hopper GPU (e.g. RTX 3090, RTX 4090, RTX A6000, A100 or H100) in order to use the server with cuda 11.x or 12.x installed. Enough memory must be available to load the model.

1. First ensure that it is working on CPU (See [First steps], page 2).
2. Ensure that you have a compatible cuda installation with cuda 11.x or 12.x. The software is preconfigured with cuda 11.x. If you want to use cuda 12.x, then change the link to the `libnc_cuda.so` library:

```
ln -sf libnc_cuda-12.so libnc_cuda.so
```

3. Then try to use the GPU with the `ts_test` utility:

```
./ts_test --cuda -m gpt2-117M.bin g "The Linux kernel is"
```

If you get an error such as:

```
Could not load: ./libnc_cuda.so
```

it means that cuda is not properly installed or that there is a mismatch between the installed cuda version and the one `ts_server` was compiled with. You can use:

```
ldd ./libnc_cuda.so
```

to check that all the required cuda libraries are present on your system.

4. Then edit the `ts_server.cfg` configuration to enable GPU support by uncommenting

```
cuda: true
```

and run the server:

```
./ts_server ts_server.cfg
```

5. Assuming you have curl, Try one request:

```
curl http://localhost:8080/v1/engines/gpt2_117M/completions \
-H "Content-Type: application/json" \
-d '{"prompt": "The Linux kernel is", "max_tokens": 100}'
```

6. You can use the integrated GUI by exploring with your browser:

```
http://localhost:8080
```

7. Depending on the amount of memory available on your GPU, you can set the memory parameter in `ts_server.cfg` to limit the amount of memory used by the server. It is usually necessary to use a few gigabytes less than maximum available amount of GPU memory.

2.2 Windows

2.2.1 First steps

The TextSynth Server works only on x86 CPUs supporting AVX2 (all Intel CPUs since 2013 support it).

1. Extract the ZIP archive, launch the shell and go into its directory:

```
cd ts_server-##version##
```

when `##version##` is the version of the program.

2. Download one small example model such as `gpt2_117M.bin` from the `ts_server` web page.
3. Use it to generate text with the "ts_test" utility:

```
ts_test -m gpt2-117M.bin g "The Linux kernel is"
```

The `-T` option can be used to use more or less CPU cores (the default is the number of physical cores).

4. Start the server:

```
ts_server ts_server.cfg
```

You can edit the `ts_server.cfg` JSON configuration file if you want to use another model.

5. Assuming you installed curl (you can download it from <https://curl.se/windows/>), try one request:

```
curl http://localhost:8080/v1/engines/gpt2_117M/completions \
-H "Content-Type: application/json" \
-d '{"prompt": "The Linux kernel is", "max_tokens": 100}'
```

The full request syntax is documented at <https://textsynth.com/documentation.html>.

6. You can use the integrated GUI by exploring with your browser:

```
http://localhost:8080
```

Now you are ready to load a larger model and to use it from your application.

2.2.2 GPU usage

You need an Nvidia Ampere, ADA or Hopper GPU (e.g. RTX 3090, RTX 4090, RTX A6000, A100 or H100) in order to use the server with cuda 11.x or 12.x installed. Enough memory must be available to load the model.

1. First ensure that it is working on CPU (see the previous section).
2. Ensure that you have a compatible cuda installation with cuda 11.x or 12.x. The software automatically detects the cuda version.
3. Then try to use the GPU with the `ts_test` utility:

```
./ts_test --cuda -m gpt2-117M.bin g "The Linux kernel is"
```

If you get an error such as:

```
Could not load: libnc_cuda-12.dll (error=126)
```

it means that cuda is not properly installed.

4. Then edit the `ts_server.cfg` configuration to enable GPU support by uncommenting

```
cuda: true
```

and run the server:

```
./ts_server ts_server.cfg
```

5. Assuming you have curl, Try one request:

```
curl http://localhost:8080/v1/engines/gpt2_117M/completions \  
-H "Content-Type: application/json" \  
-d '{"prompt": "The Linux kernel is", "max_tokens": 100}'
```

6. You can use the integrated GUI by exploring with your browser:

```
http://localhost:8080
```

7. Depending on the amount of memory available on your GPU, you can set the `memory` parameter in `ts_server.cfg` to limit the amount of memory used by the server. It is usually necessary to use a few gigabytes less than maximum available amount of GPU memory.

3 Utilities

3.1 Text processing (ts_test)

3.1.1 Text generation

```
./ts_test --cuda -m gpt2_117M.bin g "Hello, my name is"
```

When using a CPU, remove the `--cuda` option.

3.1.2 Translation

```
./ts_test --cuda -m m2m100_1_2B_q8.bin translate en fr "The dispute \
focuses on the width of seats provided on long-haul flights for \
economy passengers."
```

assuming you downloaded the `m2m100_1_2B_q8.bin` model.

3.1.3 Short text compression and decompression

```
./ts_test --cuda -m gpt2_117M.bin cs "Hello, how are you ?"
```

```
./ts_test --cuda ds "##msg##"
```

where `##msg##` is the compressed message.

3.1.4 Perplexity computation

The perplexity over a text file can be used to evaluate models. The text file is first tokenized, then cut in sequences of tokens. The default sequence length is the maximum context length of the model, use the `-l` option to change it. Then the log probabilities are averaged over a range of context positions and displayed as perplexity.

```
./ts_test --cuda -m mistral_7B.bin perplexity wiki.test.raw
ctx_len=8192, n_seq=40
START   END PERPLEXITY
    0   256     9.746
   256   512     5.758
   512  1024     5.072
  1024  2048     4.984
  2048  4096     4.934
  4096  8192     4.689
    0   8192     4.952
```

The `llama_perplexity` command evaluates the perplexity using the same algorithm as the `perplexity` utility in `llama.cpp` so that comparisons can be made. The default context length is 512.

```
./ts_test --cuda -m mistral_7B.bin llama_perplexity wiki.test.raw
ctx_len=512, start=256, n_seq=642
#SEQ PERPLEXITY
  641     5.6946
```

3.2 Text to image (ts_sd)

```
./ts_sd --cuda -m sd_v1.4.bin -o out.jpg "an astronaut riding a horse"
```

assuming you downloaded `sd_v1.4.bin`.

When using a CPU, remove the `--cuda` option.

3.3 Chat (ts_chat)

```
./ts_chat --cuda -m llama2_7B_chat_q4.bin
```

assuming you downloaded `llama2_7B_chat_q4.bin`.

When using a CPU, remove the `--cuda` option.

During the chat, some commands are available. Use `/h` during the chat to have some help. Type `Ctrl-C` once to stop the output and twice to quit.

3.4 Speech to text transcription (ts_audiototext)

```
./ts_audiototext --cuda -m whisper_large_v3_q8.bin -o out.json audiofile.mp3
```

assuming you downloaded `whisper_large_v3_q8.bin` and that `audiofile.mp3` is the audio file to be transcribed. `out.json` contains the transcribed text.

When using a CPU, remove the `--cuda` option.

3.5 Text compression (ts_zip)

To compress a text file (here `alice29.txt`), assuming you downloaded the `rwkv_169M.bin` model, use:

```
./ts_zip --cuda -m rwkv_169M.bin c alice29.txt /tmp/out.bin
```

To decompress it:

```
./ts_zip --cuda -m rwkv_169M.bin d /tmp/out.bin /tmp/out.txt
```

A checksum is included in the compressed file and it is automatically checked. It is essential to use the same software version, language model and GPU model when compressing and decompressing a file.

Large compression gains occur only if the input file is in a language that the language model has already seen.

The compression ratio, speed and memory usage depend on the language model but also on the selected context length (`-l` option) and batch size (`-b` option). They are both chosen automatically but can be overridden:

- The memory usage is proportional to the context length and batch size.
- The speed increases when the batch size increases.
- The compression ratio increases with larger context lengths and smaller batch sizes.

More information is available at https://bellard.org/ts_server/ts_zip.html.

3.6 Model Weight Conversion

TextSynth Server uses a specific file format to store the weights of the models. Python scripts are provided in `scripts/` to convert model checkpoints to the TextSynth format. The tokenizer is now included in the model file. For backward compatibility, tokenizer files are provided in the `tokenizer/` directory.

The script `hf_model_convert.py` should be used when converting from a Hugging Face model.

Example to convert Llama2 weights from Hugging Face to TextSynth:

```
python hf_model_convert.py --tokenizer llama_vocab.txt model_dir llama2.bin
```

where:

- `model_dir` is the directory containing the `config.json` and `pytorch_model*.bin` files.
- `llama_vocab.txt` is a `ts_server` tokenizer file from the `tokenizer/` directory.

3.7 Model Weight Quantization

With the `ncconvert` utility, it is possible to quantize the model weights to 8, 4 or 3 bits. Quantization reduces the GPU memory usage and increases the inference speed. 8 bit quantization yields a negligible loss. 4 bit quantization yields a very small loss. 3 bit quantization currently only works on a GPU.

Examples:

8 bit quantization:

```
./ncconvert -q bf8 pythia_deduped_160M.bin pythia_deduped_160M_q8.bin
```

4 bit quantization:

```
./ncconvert -q bf4 pythia_deduped_160M.bin pythia_deduped_160M_q4.bin
```

4 TS Server Configuration

The file `ts_server.cfg` provides an example of configuration.

4.1 Syntax

The syntax is similar to JSON with a few modifications:

- property names can be unquoted

```
{ property: 1 }
```
- Multi-line and single line C style comments are accepted

4.2 Configuration parameters

cuda Optional boolean (default = false). If true, CUDA (Nvidia GPU support) is enabled.

device_index

Optional integer (default = 0). Select the GPU device when using several GPUs. Use the `nvidia-smi` utility to list the available devices.

n_threads

Optional integer. When using a CPU, select the number of threads. It is set by default to the number of physical cores.

full_memory

Optional boolean (default = true). When using a GPU, `ts_server` reserves by default all the GPU memory for better efficiency. This parameter disables this behavior so that the GPU memory is allocated on demand.

max_memory

Optional integer (default = 0). If non zero, limit the consumed GPU memory to this value by pausing the HTTP requests until there is enough memory.

Since there is some overhead when handling the requests, it is better to set a value a few GB lower than the amount of total GPU memory.

kv_cache_max_count

Optional integer (default = 0). See the `kv_cache_size` parameter.

kv_cache_size

Optional integer (default = 0). The KV cache is stored in CPU memory and is only used by the `chat` endpoint to store the context of the conversation to accelerate the inference. It is disabled by default. `kv_cache_size` sets the maximum KV cache memory in bytes. `kv_cache_max_count` sets the maximum number of cached conversations.

models Array of objects. Each element defines a model that is served. The following parameters are defined:

name String. Name (ID) of the model in the HTTP requests.

filename String. Filename of the model. You can use the conversion scripts to create one from Pytorch checkpoints if necessary.

draft_model

Optional string. Filename of a smaller model used to accelerate inference (speculative sampling). The draft model must use the same tokenizer as the large model.

`sps_k_max`

Optional integer. When using speculative sampling, specify the maximum number of tokens that is predicted by the draft model. The optimal value needs to be determined by experimentation. It is usually 3 or 4.

Note: the free version only accepts one model definition.

`local_port`

Integer. TCP port on which the HTTP server listens to.

`bind_addr`

Optional string (default = "0.0.0.0"). Set the IP address on which the server listens to. Use "127.0.0.1" if you want to accept local connections only.

`tls`

Optional boolean (default = false). If true, HTTPS (TLS) connections are accepted instead of HTTP ones.

`tls_cert_file`

Optional string. If TLS is enabled, the certificate (PEM format) must be provided with this parameter.

`tls_key_file`

Optional string. If TLS is enabled, the private key of the certificate (PEM format) must be provided with this parameter.

`log_start`

Optional boolean (default = false). Print "Started." on the console when the server has loaded all the models and is ready to accept connections.

`gui`

Optional boolean (default = false). If true, enable a Graphical User Interface in addition to the remote API. It is available at the root URL, e.g. <http://127.0.0.1:8080>. The server just serves the files present in the `gui/` directory. You can modify or add new files if needed.

`log_filename`

String. Set the filename where the logs are written. There is one line per connection. The fields are:

- date and time (ISO format)
- source IP address
- HTTP method
- URI
- posted JSON

`from_proxy`

Optional boolean (default = true). If true, use the `X-Forwarded-For` header if available to determine the source IP address in the logs. It is useful to have the real IP address of a client when a proxy is used.

4.3 JSON endpoints

The server provides the following endpoints.

`v1/engines/{model_id}/completions`

Text completion.

Complete documentation at <https://textsynth.com/documentation.html>.

See `api_examples/completion.py` to have an example in Python.

`v1/engines/{model_id}/chat`

Chat based completion. Complete documentation at <https://textsynth.com/documentation.html>.

`v1/engines/{model_id}/translate`

Translation.

Complete documentation at <https://textsynth.com/documentation.html>.

See `api_examples/translate.py` to have an example in Python.

`v1/engines/{model_id}/logprob`

Log probability computation.

Complete documentation at <https://textsynth.com/documentation.html>.

`v1/engines/{model_id}/tokenize`

Tokenization.

Complete documentation at <https://textsynth.com/documentation.html>.

`v1/engines/{model_id}/text_to_image`

Text to image.

Complete documentation at <https://textsynth.com/documentation.html>.

See `api_examples/sd.py` to have an example in Python.

`v1/engines/{model_id}/transcript`

Speech to text transcription. See `api_examples/transcript.py` to have an example in Python.

The content type of the posted data should be `multipart/form-data` and should contain two files with the following names:

`json` contains the JSON request.

`file` contains the audio file to transcript. FFmpeg is invoked by `ts_server` to convert the audio file to raw samples.

The JSON request contains the following properties:

`language` String. The input ISO language code. The following languages are available: af, am, ar, as, az, ba, be, bg, bn, bo, br, bs, ca, cs, cy, da, de, el, en, es, et, eu, fa, fi, fo, fr, gl, gu, ha, haw, he, hi, hr, ht, hu, hy, id, is, it, ja, jw, ka, kk, km, kn, ko, la, lb, ln, lo, lt, lv, mg, mi, mk, ml, mn, mr, ms, mt, my, ne, nl, nn, no, oc, pa, pl, ps, pt, ro, ru, sa, sd, si, sk, sl, sn, so, sq, sr, su, sv, sw, ta, te, tg, th, tk, tl, tr, tt, uk, ur, uz, vi, yi, yo, yue, zh.

Additional parameters are available for testing or tuning:

`num_beams`

Optional integer, range: 2 to 5 (default = 5). Number of beams used for decoding.

`condition_on_previous_text`

Optional boolean (default = false). Condition the current frame on the previous text.

`logprob_threshold`

Option float (default = -1.0).

`no_speech_threshold`

Optional float (default = 0.6). Probability threshold of the `no_speech` token for no speech detection. The average log-probability of the generated tokens must also be below `logprob_threshold`.

A JSON object is returned containing the transcription. It contains the following properties:

text String. Transcribed text.

segments Array of objects containing the transcribed text segments with time-stamps. Each segment has the following properties:

- id** Integer. Segment ID.
- start** Float. Start time in seconds.
- end** Float. End time in seconds.
- text** String. Transcribed text for this segment.

language String. ISO language code.

duration Float. Transcription duration in seconds.

v1/memory_stats

Return a JSON object with the memory usage statistics. The following properties are available:

cur_memory
Integer. Current used memory in bytes (CPU or GPU memory).

max_memory
Integer. Maximum used memory in bytes since the last call (CPU or GPU memory).

kv_cache_count
Integer. Number of entries in the KV cache count.

kv_cache_size
Integer. CPU Memory in bytes used by the KV cache.

v1/models

Return the list of available models and their capabilities. It is used by the GUI.