LLaMA.cpp HTTP Server

Fast, lightweight, pure C/C++ HTTP server based on httplib, nlohmann::json and **llama.cpp**.

Set of LLM REST APIs and a simple web front end to interact with llama.cpp.

Features:

- LLM inference of F16 and quantized models on GPU and CPU
- OpenAI API compatible chat completions and embeddings routes
- Reranking endoint (WIP: #9510)
- Parallel decoding with multi-user support
- Continuous batching
- Multimodal (wip)
- Monitoring endpoints
- Schema-constrained JSON response format

The project is under active development, and we are looking for feedback and contributors.

Usage

Common params

Argument	Explanation
`-h, --help, --usage`	print usage and exit

Argument	Explanation
`--version`	show version and build info
`--verbose-prompt`	print a verbose prompt before generation (default: false)
`-t, --threads N`	number of threads to use during generation (default: -1) (env: LLAMA_ARG_THREADS)
`-tb, --threads-batch N`	number of threads to use during batch and prompt processing (default: same as --threads)
`-C, --cpu-mask M`	CPU affinity mask: arbitrarily long hex. Complements cpu-range (default: "")
`-Cr, --cpu-range lo-hi`	range of CPUs for affinity. Complements --cpu-mask
`--cpu-strict <0|1>`	use strict CPU placement (default: 0)
`--prio N`	set process/thread priority : 0-normal, 1-medium, 2-high, 3-realtime (default: 0)
`--poll <0...100>`	use polling level to wait for work (0 - no polling, default: 50)
`-Cb, --cpu-mask-batch M`	CPU affinity mask: arbitrarily long hex. Complements cpu-range-batch (default: same as --cpu-mask)
`-Crb, --cpu-range-batch lo-hi`	ranges of CPUs for affinity. Complements --cpu-mask-batch
`--cpu-strict-batch <0|1>`	use strict CPU placement (default: same as --cpu-strict)
`--prio-batch N`	set process/thread priority : 0-normal, 1-medium, 2-high, 3-realtime (default: 0)
`--poll-batch <0|1>`	use polling to wait for work (default: same as --poll)
`-c, --ctx-size N`	size of the prompt context (default: 4096, 0 = loaded from model) (env: LLAMA_ARG_CTX_SIZE)
`-n, --predict, --n-predict N`	number of tokens to predict (default: -1, -1 = infinity, -2 = until context filled) (env: LLAMA_ARG_N_PREDICT)
`-b, --batch-size N`	logical maximum batch size (default: 2048) (env: LLAMA_ARG_BATCH)

Argument	Explanation
`-ub, --ubatch-size N`	physical maximum batch size (default: 512) (env: LLAMA_ARG_UBATCH)
`--keep N`	number of tokens to keep from the initial prompt (default: 0, -1 = all)
`-fa, --flash-attn`	enable Flash Attention (default: disabled) (env: LLAMA_ARG_FLASH_ATTN)
`-p, --prompt PROMPT`	prompt to start generation with
`--no-perf`	disable internal libllama performance timings (default: false) (env: LLAMA_ARG_NO_PERF)
`-f, --file FNAME`	a file containing the prompt (default: none)
`-bf, --binary-file FNAME`	binary file containing the prompt (default: none)
`-e, --escape`	process escapes sequences (\n, \r, \t, ', ", \) (default: true)
`--no-escape`	do not process escape sequences
`--rope-scaling {none,linear,yarn}`	RoPE frequency scaling method, defaults to linear unless specified by the model (env: LLAMA_ARG_ROPE_SCALING_TYPE)
`--rope-scale N`	RoPE context scaling factor, expands context by a factor of N (env: LLAMA_ARG_ROPE_SCALE)
`--rope-freq-base N`	RoPE base frequency, used by NTK-aware scaling (default: loaded from model) (env: LLAMA_ARG_ROPE_FREQ_BASE)
`--rope-freq-scale N`	RoPE frequency scaling factor, expands context by a factor of 1/N (env: LLAMA_ARG_ROPE_FREQ_SCALE)
`--yarn-orig-ctx N`	YaRN: original context size of model (default: 0 = model training context size) (env: LLAMA_ARG_YARN_ORIG_CTX)
`--yarn-ext-factor N`	YaRN: extrapolation mix factor (default: -1.0, 0.0 = full interpolation) (env: LLAMA_ARG_YARN_EXT_FACTOR)

Argument	Explanation
`--yarn-attn-factor N`	YaRN: scale sqrt(t) or attention magnitude (default: 1.0) (env: LLAMA_ARG_YARN_ATTN_FACTOR)
`--yarn-beta-slow N`	YaRN: high correction dim or alpha (default: 1.0) (env: LLAMA_ARG_YARN_BETA_SLOW)
`--yarn-beta-fast N`	YaRN: low correction dim or beta (default: 32.0) (env: LLAMA_ARG_YARN_BETA_FAST)
`-dkvc, --dump-kv-cache`	verbose print of the KV cache
`-nkvo, --no-kv-offload`	disable KV offload (env: LLAMA_ARG_NO_KV_OFFLOAD)
`-ctk, --cache-type-k TYPE`	KV cache data type for K (default: f16) (env: LLAMA_ARG_CACHE_TYPE_K)
`-ctv, --cache-type-v TYPE`	KV cache data type for V (default: f16) (env: LLAMA_ARG_CACHE_TYPE_V)
`-dt, --defrag-thold N`	KV cache defragmentation threshold (default: 0.1, < 0 - disabled) (env: LLAMA_ARG_DEFRAG_THOLD)
`-np, --parallel N`	number of parallel sequences to decode (default: 1) (env: LLAMA_ARG_N_PARALLEL)
`--mlock`	force system to keep model in RAM rather than swapping or compressing (env: LLAMA_ARG_MLOCK)
`--no-mmap`	do not memory-map model (slower load but may reduce pageouts if not using mlock) (env: LLAMA_ARG_NO_MMAP)
`--numa TYPE`	attempt optimizations that help on some NUMA systems - distribute: spread execution evenly over all nodes - isolate: only spawn threads on CPUs on the node that execution started on - numactl: use the CPU map provided by numactl if run without this previously, it is recommended to drop the system page cache before using this see #1437 (env: LLAMA_ARG_NUMA)

Argument	Explanation
`-ngl, --gpu-layers, --n-gpu-layers N`	number of layers to store in VRAM (env: LLAMA_ARG_N_GPU_LAYERS)
`-sm, --split-mode {none,layer,row}`	how to split the model across multiple GPUs, one of: - none: use one GPU only - layer (default): split layers and KV across GPUs - row: split rows across GPUs (env: LLAMA_ARG_SPLIT_MODE)
`-ts, --tensor-split N0,N1,N2,...`	fraction of the model to offload to each GPU, comma-separated list of proportions, e.g. 3,1 (env: LLAMA_ARG_TENSOR_SPLIT)
`-mg, --main-gpu INDEX`	the GPU to use for the model (with split-mode = none), or for intermediate results and KV (with split-mode = row) (default: 0) (env: LLAMA_ARG_MAIN_GPU)
`--check-tensors`	check model tensor data for invalid values (default: false)
`--override-kv KEY=TYPE:VALUE`	advanced option to override model metadata by key. may be specified multiple times. types: int, float, bool, str. example: --override-kv tokenizer.ggml.add_bos_token=bool:false
`--lora FNAME`	path to LoRA adapter (can be repeated to use multiple adapters)
`--lora-scaled FNAME SCALE`	path to LoRA adapter with user defined scaling (can be repeated to use multiple adapters)
`--control-vector FNAME`	add a control vector note: this argument can be repeated to add multiple control vectors
`--control-vector-scaled FNAME SCALE`	add a control vector with user defined scaling SCALE note: this argument can be repeated to add multiple scaled control vectors
`--control-vector-layer-range START END`	layer range to apply the control vector(s) to, start and end inclusive
`-m, --model FNAME`	model path (default: `models/$filename` with filename from `--hf-file` or `--model-url` if set, otherwise models/7B/ggml-model-f16.gguf) (env: LLAMA_ARG_MODEL)

Argument	Explanation
`-mu, --model-url MODEL_URL`	model download url (default: unused) (env: LLAMA_ARG_MODEL_URL)
`-hfr, --hf-repo REPO`	Hugging Face model repository (default: unused) (env: LLAMA_ARG_HF_REPO)
`-hff, --hf-file FILE`	Hugging Face model file (default: unused) (env: LLAMA_ARG_HF_FILE)
`-hft, --hf-token TOKEN`	Hugging Face access token (default: value from HF_TOKEN environment variable) (env: HF_TOKEN)
`-ld, --logdir LOGDIR`	path under which to save YAML logs (no logging if unset)
`--log-disable`	Log disable
`--log-file FNAME`	Log to file
`--log-colors`	Enable colored logging (env: LLAMA_LOG_COLORS)
`-v, --verbose, --log-verbose`	Set verbosity level to infinity (i.e. log all messages, useful for debugging)
`-lv, --verbosity, --log-verbosity N`	Set the verbosity threshold. Messages with a higher verbosity will be ignored. (env: LLAMA_LOG_VERBOSITY)
`--log-prefix`	Enable prefx in log messages (env: LLAMA_LOG_PREFIX)
`--log-timestamps`	Enable timestamps in log messages (env: LLAMA_LOG_TIMESTAMPS)

Sampling params

Argument	Explanation
`--samplers SAMPLERS`	samplers that will be used for generation in the order, separated by ';' (default: dry;top_k;typ_p;top_p;min_p;xtc;temperature)
`-s, --seed SEED`	RNG seed (default: -1, use random seed for -1)
`--sampling-seq SEQUENCE`	simplified sequence for samplers that will be used (default: dkypmxt)

Argument	Explanation
`--ignore-eos`	ignore end of stream token and continue generating (implies --logit-bias EOS-inf)
`--penalize-nl`	penalize newline tokens (default: false)
`--temp N`	temperature (default: 0.8)
`--top-k N`	top-k sampling (default: 40, 0 = disabled)
`--top-p N`	top-p sampling (default: 0.9, 1.0 = disabled)
`--min-p N`	min-p sampling (default: 0.1, 0.0 = disabled)
`--xtc-probability N`	xtc probability (default: 0.0, 0.0 = disabled)
`--xtc-threshold N`	xtc threshold (default: 0.1, 1.0 = disabled)
`--typical N`	locally typical sampling, parameter p (default: 1.0, 1.0 = disabled)
`--repeat-last-n N`	last n tokens to consider for penalize (default: 64, 0 = disabled, -1 = ctx_size)
`--repeat-penalty N`	penalize repeat sequence of tokens (default: 1.0, 1.0 = disabled)
`--presence-penalty N`	repeat alpha presence penalty (default: 0.0, 0.0 = disabled)
`--frequency-penalty N`	repeat alpha frequency penalty (default: 0.0, 0.0 = disabled)
`--dry-multiplier N`	set DRY sampling multiplier (default: 0.0, 0.0 = disabled)
`--dry-base N`	set DRY sampling base value (default: 1.75)
`--dry-allowed-length N`	set allowed length for DRY sampling (default: 2)
`--dry-penalty-last-n N`	set DRY penalty for the last n tokens (default: -1, 0 = disable, -1 = context size)
`--dry-sequence-breaker STRING`	add sequence breaker for DRY sampling, clearing out default breakers ('\n', ':', '"', '*') in the process; use "none" to not use any sequence breakers
`--dynatemp-range N`	dynamic temperature range (default: 0.0, 0.0 = disabled)
`--dynatemp-exp N`	dynamic temperature exponent (default: 1.0)

Argument	Explanation
`--mirostat N`	use Mirostat sampling. Top K, Nucleus and Locally Typical samplers are ignored if used. (default: 0, 0 = disabled, 1 = Mirostat, 2 = Mirostat 2.0)
`--mirostat-lr N`	Mirostat learning rate, parameter eta (default: 0.1)
`--mirostat-ent N`	Mirostat target entropy, parameter tau (default: 5.0)
`-l, --logit-bias TOKEN_ID(+/-)BIAS`	modifies the likelihood of token appearing in the completion, i.e. `--logit-bias 15043+1` to increase likelihood of token ' Hello', or `--logit-bias 15043-1` to decrease likelihood of token ' Hello'
`--grammar GRAMMAR`	BNF-like grammar to constrain generations (see samples in grammars/ dir) (default: '')
`--grammar-file FNAME`	file to read grammar from
`-j, --json-schema SCHEMA`	JSON schema to constrain generations (https://json-schema.org/), e.g. `{}` for any JSON object For schemas w/ external $refs, use --grammar + example/json_schema_to_grammar.py instead

Example-specific params

Argument	Explanation
`--no-context-shift`	disables context shift on inifinite text generation (defau disabled) (env: LLAMA_ARG_NO_CONTEXT_SHIFT)
`-sp, --special`	special tokens output enabled (default: false)
`--spm-infill`	use Suffix/Prefix/Middle pattern for infill (instead of Prefix/Suffix/Middle) as some models prefer this. (defau disabled)
`--pooling {none,mean,cls,last,rank}`	pooling type for embeddings, use model default if unspecified (env: LLAMA_ARG_POOLING)
`-cb, --cont-batching`	enable continuous batching (a.k.a dynamic batching) (default: enabled)

Argument	Explanation
	(env: LLAMA_ARG_CONT_BATCHING)
`-nocb, --no-cont-batching`	disable continuous batching (env: LLAMA_ARG_NO_CONT_BATCHING)
`-a, --alias STRING`	set alias for model name (to be used by REST API) (env: LLAMA_ARG_ALIAS)
`--host HOST`	ip address to listen (default: 127.0.0.1) (env: LLAMA_ARG_HOST)
`--port PORT`	port to listen (default: 8080) (env: LLAMA_ARG_PORT)
`--path PATH`	path to serve static files from (default:) (env: LLAMA_ARG_STATIC_PATH)
`--embedding, --embeddings`	restrict to only support embedding use case; use only with dedicated embedding models (default: disabled) (env: LLAMA_ARG_EMBEDDINGS)
`--reranking, --rerank`	enable reranking endpoint on server (default: disabled) (env: LLAMA_ARG_RERANKING)
`--api-key KEY`	API key to use for authentication (default: none) (env: LLAMA_API_KEY)
`--api-key-file FNAME`	path to file containing API keys (default: none)
`--ssl-key-file FNAME`	path to file a PEM-encoded SSL private key (env: LLAMA_ARG_SSL_KEY_FILE)
`--ssl-cert-file FNAME`	path to file a PEM-encoded SSL certificate (env: LLAMA_ARG_SSL_CERT_FILE)
`-to, --timeout N`	server read/write timeout in seconds (default: 600) (env: LLAMA_ARG_TIMEOUT)
`--threads-http N`	number of threads used to process HTTP requests (default: -1) (env: LLAMA_ARG_THREADS_HTTP)
`--cache-reuse N`	min chunk size to attempt reusing from the cache via K shifting (default: 0) (env: LLAMA_ARG_CACHE_REUSE)
`--metrics`	enable prometheus compatible metrics endpoint (default: disabled)

Argument	Explanation
	(env: LLAMA_ARG_ENDPOINT_METRICS)
`--slots`	enable slots monitoring endpoint (default: disabled) (env: LLAMA_ARG_ENDPOINT_SLOTS)
`--props`	enable changing global properties via POST /props (default: disabled) (env: LLAMA_ARG_ENDPOINT_PROPS)
`--no-slots`	disables slots monitoring endpoint (env: LLAMA_ARG_NO_ENDPOINT_SLOTS)
`--slot-save-path PATH`	path to save slot kv cache (default: disabled)
`--chat-template JINJA_TEMPLATE`	set custom jinja chat template (default: template taken from model's metadata) if suffix/prefix are specified, template will be disabled only commonly used templates are accepted: https://github.com/ggerganov/llama.cpp/wiki/Template supported-by-llama_chat_apply_template (env: LLAMA_ARG_CHAT_TEMPLATE)
`-sps, --slot-prompt-similarity SIMILARITY`	how much the prompt of a request must match the prompt of a slot in order to use that slot (default: 0.50, 0.0 = disabled)
`--lora-init-without-apply`	load LoRA adapters without applying them (apply later via POST /lora-adapters) (default: disabled)

Note: If both command line argument and environment variable are both set for the same param, the argument will take precedence over env var.

Example usage of docker compose with environment variables:

```yaml
services:
  llamacpp-server:
    image: ghcr.io/ggerganov/llama.cpp:server
    ports:
      - 8080:8080
    volumes:
      - ./models:/models
    environment:
      # alternatively, you can use "LLAMA_ARG_MODEL_URL" to download the mo
      LLAMA_ARG_MODEL: /models/my_model.gguf
      LLAMA_ARG_CTX_SIZE: 4096
      LLAMA_ARG_N_PARALLEL: 2
```

```
LLAMA_ARG_ENDPOINT_METRICS: 1
LLAMA_ARG_PORT: 8080
```

Build

`llama-server` is built alongside everything else from the root of the project

- Using `make`:

```
make llama-server
```

- Using `CMake`:

```
cmake -B build
cmake --build build --config Release -t llama-server
```

Binary is at `./build/bin/llama-server`

Build with SSL

`llama-server` can also be built with SSL support using OpenSSL 3

- Using `make`:

```
# NOTE: For non-system openssl, use the following:
#   CXXFLAGS="-I /path/to/openssl/include"
#   LDFLAGS="-L /path/to/openssl/lib"
make LLAMA_SERVER_SSL=true llama-server
```

- Using `CMake`:

```
cmake -B build -DLLAMA_SERVER_SSL=ON
cmake --build build --config Release -t llama-server
```

Quick Start

To get started right away, run the following command, making sure to use the correct path for the model you have:

Unix-based systems (Linux, macOS, etc.)

```
./llama-server -m models/7B/ggml-model.gguf -c 2048
```

Windows

```
llama-server.exe -m models\7B\ggml-model.gguf -c 2048
```

The above command will start a server that by default listens on `127.0.0.1:8080`. You can consume the endpoints with Postman or NodeJS with axios library. You can visit the web front end at the same url.

Docker

```
docker run -p 8080:8080 -v /path/to/models:/models ghcr.io/ggerganov/llama.

# or, with CUDA:
docker run -p 8080:8080 -v /path/to/models:/models --gpus all ghcr.io/ggerg
```

Testing with CURL

Using curl. On Windows, `curl.exe` should be available in the base OS.

```
curl --request POST \
    --url http://localhost:8080/completion \
    --header "Content-Type: application/json" \
    --data '{"prompt": "Building a website can be done in 10 simple steps:"
```

Advanced testing

We implemented a server test framework using human-readable scenario.

Before submitting an issue, please try to reproduce it with this format.

Node JS Test

You need to have Node.js installed.

```
mkdir llama-client
```

```
cd llama-client
```

Create a index.js file and put this inside:

```
const prompt = `Building a website can be done in 10 simple steps:`;

async function Test() {
    let response = await fetch("http://127.0.0.1:8080/completion", {
        method: 'POST',
        body: JSON.stringify({
            prompt,
            n_predict: 512,
        })
    })
    console.log((await response.json()).content)
}

Test()
```

And run it:

```
node index.js
```

API Endpoints

GET `/health`: Returns heath check result

Response format

- HTTP status code 503
  - Body: `{"error": {"code": 503, "message": "Loading model", "type": "unavailable_error"}}`
  - Explanation: the model is still being loaded.
- HTTP status code 200
  - Body: `{"status": "ok" }`
  - Explanation: the model is successfully loaded and the server is ready.

POST `/completion`: Given a `prompt`, it returns the predicted completion.

```
*Options:*

`prompt`: Provide the prompt for this completion as a string or as an
array of strings or numbers representing tokens. Internally, if
```

`cache_prompt` is `true`, the prompt is compared to the previous completion and only the "unseen" suffix is evaluated. A `BOS` token is inserted at the start, if all of the following conditions are true:

  - The prompt is a string or an array with the first element given as a string
  - The model's `tokenizer.ggml.add_bos_token` metadata is `true`

These input shapes and data type are allowed for `prompt`:

  - Single string: `"string"`
  - Single sequence of tokens: `[12, 34, 56]`
  - Mixed tokens and strings: `[12, 34, "string", 56, 78]`

Multiple prompts are also supported. In this case, the completion result will be an array.

  - Only strings: `["string1", "string2"]`
  - Strings and sequences of tokens: `["string1", [12, 34, 56]]`
  - Mixed types: `[[12, 34, "string", 56, 78], [12, 34, 56], "string"]`

`temperature`: Adjust the randomness of the generated text. Default: `0.8`

`dynatemp_range`: Dynamic temperature range. The final temperature will be in the range of `[temperature - dynatemp_range; temperature + dynatemp_range]` Default: `0.0`, which is disabled.

`dynatemp_exponent`: Dynamic temperature exponent. Default: `1.0`

`top_k`: Limit the next token selection to the K most probable tokens. Default: `40`

`top_p`: Limit the next token selection to a subset of tokens with a cumulative probability above a threshold P. Default: `0.95`

`min_p`: The minimum probability for a token to be considered, relative to the probability of the most likely token. Default: `0.05`

`n_predict`: Set the maximum number of tokens to predict when generating text. **Note:** May exceed the set limit slightly if the last token is a partial multibyte character. When 0, no tokens will be generated but the prompt is evaluated into the cache. Default: `-1`, where `-1` is infinity.

`n_indent`: Specify the minimum line indentation for the generated text in number of whitespace characters. Useful for code completion tasks. Default: `0`

`n_keep`: Specify the number of tokens from the prompt to retain when the context size is exceeded and tokens need to be discarded. The number excludes the BOS token.
By default, this value is set to `0`, meaning no tokens are kept. Use

`-1` to retain all tokens from the prompt.

`stream`: It allows receiving each predicted token in real-time instead of waiting for the completion to finish. To enable this, set to `true`.

`stop`: Specify a JSON array of stopping strings.
These words will not be included in the completion, so make sure to add them to the prompt for the next iteration. Default: `[]`

`typical_p`: Enable locally typical sampling with parameter p. Default: `1.0`, which is disabled.

`repeat_penalty`: Control the repetition of token sequences in the generated text. Default: `1.1`

`repeat_last_n`: Last n tokens to consider for penalizing repetition. Default: `64`, where `0` is disabled and `-1` is ctx-size.

`penalize_nl`: Penalize newline tokens when applying the repeat penalty. Default: `true`

`presence_penalty`: Repeat alpha presence penalty. Default: `0.0`, which is disabled.

`frequency_penalty`: Repeat alpha frequency penalty. Default: `0.0`, which is disabled.

`dry_multiplier`: Set the DRY (Don't Repeat Yourself) repetition penalty multiplier. Default: `0.0`, which is disabled.

`dry_base`: Set the DRY repetition penalty base value. Default: `1.75`

`dry_allowed_length`: Tokens that extend repetition beyond this receive exponentially increasing penalty: multiplier * base ^ (length of repeating sequence before token - allowed length). Default: `2`

`dry_penalty_last_n`: How many tokens to scan for repetitions. Default: `-1`, where `0` is disabled and `-1` is context size.

`dry_sequence_breakers`: Specify an array of sequence breakers for DRY sampling. Only a JSON array of strings is accepted. Default: `['\n', ':', '"', '*']`

`xtc_probability`: Set the chance for token removal via XTC sampler. Default: `0.0`, which is disabled.

`xtc_threshold`: Set a minimum probability threshold for tokens to be removed via XTC sampler. Default: `0.1` (> `0.5` disables XTC)

`mirostat`: Enable Mirostat sampling, controlling perplexity during text generation. Default: `0`, where `0` is disabled, `1` is Mirostat, and `2` is Mirostat 2.0.

`mirostat_tau`: Set the Mirostat target entropy, parameter tau. Default: `5.0`

`mirostat_eta`: Set the Mirostat learning rate, parameter eta. Default: `0.1`

`grammar`: Set grammar for grammar-based sampling. Default: no grammar

`json_schema`: Set a JSON schema for grammar-based sampling (e.g. `{"items": {"type": "string"}, "minItems": 10, "maxItems": 100}` of a list of strings, or `{}` for any JSON). See [tests](../../tests/test-json-schema-to-grammar.cpp) for supported features. Default: no JSON schema.

`seed`: Set the random number generator (RNG) seed. Default: `-1`, which is a random seed.

`ignore_eos`: Ignore end of stream token and continue generating. Default: `false`

`logit_bias`: Modify the likelihood of a token appearing in the generated text completion. For example, use `"logit_bias": [[15043,1.0]]` to increase the likelihood of the token 'Hello', or `"logit_bias": [[15043,-1.0]]` to decrease its likelihood. Setting the value to false, `"logit_bias": [[15043,false]]` ensures that the token `Hello` is never produced. The tokens can also be represented as strings, e.g. `[["Hello, World!",-0.5]]` will reduce the likelihood of all the individual tokens that represent the string `Hello, World!`, just like the `presence_penalty` does. Default: `[]`

`n_probs`: If greater than 0, the response also contains the probabilities of top N tokens for each generated token given the sampling settings. Note that for temperature < 0 the tokens are sampled greedily but token probabilities are still being calculated via a simple softmax of the logits without considering any other sampler settings. Default: `0`

`min_keep`: If greater than 0, force samplers to return N possible tokens at minimum. Default: `0`

`t_max_predict_ms`: Set a time limit in milliseconds for the prediction (a.k.a. text-generation) phase. The timeout will trigger if the generation takes more than the specified time (measured since the first token was generated) and if a new-line character has already been generated. Useful for FIM applications. Default: `0`, which is disabled.

`image_data`: An array of objects to hold base64-encoded image `data` and its `id`s to be reference in `prompt`. You can determine the place of the image in the prompt as in the following: `USER:[img-12]Describe the image in detail.\nASSISTANT:`. In this case, `[img-12]` will be replaced by the embeddings of the image with id `12` in the following `image_data` array: `{..., "image_data": [{"data": "<BASE64_STRING>", "id": 12}]}`. Use `image_data` only with multimodal models, e.g., LLaVA.

`id_slot`: Assign the completion task to an specific slot. If is -1 the task will be assigned to a Idle slot. Default: `-1`

`cache_prompt`: Re-use KV cache from a previous request if possible. This way the common prefix does not have to be re-processed, only the suffix that differs between the requests. Because (depending on the backend) the logits are **not** guaranteed to be bit-for-bit identical for different batch sizes (prompt processing vs. token generation) enabling this option can cause nondeterministic results. Default: `false`

`samplers`: The order the samplers should be applied in. An array of strings representing sampler type names. If a sampler is not set, it will not be used. If a sampler is specified more than once, it will be applied multiple times. Default: `["dry", "top_k", "typ_p", "top_p", "min_p", "xtc", "temperature"]` - these are all the available values.

Response format

- Note: When using streaming mode (`stream`), only `content` and `stop` will be returned until end of completion.
- `completion_probabilities` : An array of token probabilities for each completion. The array's length is `n_predict` . Each item in the array has the following structure:

```
{
  "content": "<the token selected by the model>",
  "probs": [
    {
      "prob": float,
      "tok_str": "<most likely token>"
    },
    {
      "prob": float,
      "tok_str": "<second most likely token>"
    },
    ...
  ]
},
```

Notice that each `probs` is an array of length `n_probs` .

- `content` : Completion result as a string (excluding `stopping_word` if any). In case of streaming mode, will contain the next token as a string.
- `stop` : Boolean for use with `stream` to check whether the generation has stopped (Note: This is not related to stopping words array `stop` from input options)

- `generation_settings` : The provided options above excluding `prompt` but including `n_ctx` , `model` . These options may differ from the original ones in some way (e.g. bad values filtered out, strings converted to tokens, etc.).
- `model` : The path to the model loaded with `-m`
- `prompt` : The provided `prompt`
- `stopped_eos` : Indicating whether the completion has stopped because it encountered the EOS token
- `stopped_limit` : Indicating whether the completion stopped because `n_predict` tokens were generated before stop words or EOS was encountered
- `stopped_word` : Indicating whether the completion stopped due to encountering a stopping word from `stop` JSON array provided
- `stopping_word` : The stopping word encountered which stopped the generation (or "" if not stopped due to a stopping word)
- `timings` : Hash of timing information about the completion such as the number of tokens `predicted_per_second`
- `tokens_cached` : Number of tokens from the prompt which could be re-used from previous completion (`n_past`)
- `tokens_evaluated` : Number of tokens evaluated in total from the prompt
- `truncated` : Boolean indicating if the context size was exceeded during generation, i.e. the number of tokens provided in the prompt (`tokens_evaluated`) plus tokens generated (`tokens predicted`) exceeded the context size (`n_ctx`)

POST `/tokenize` : Tokenize a given text

```
*Options:*

`content`: (Required) The text to tokenize.

`add_special`: (Optional) Boolean indicating if special tokens, i.e.
`BOS`, should be inserted.  Default: `false`

`with_pieces`: (Optional) Boolean indicating whether to return token
pieces along with IDs.  Default: `false`
```

Response:

Returns a JSON object with a `tokens` field containing the tokenization result. The `tokens` array contains either just token IDs or objects with `id` and `piece` fields, depending on the `with_pieces` parameter. The piece field is a string if the piece is valid unicode or a list of bytes otherwise.

If `with_pieces` is `false` :

```
{
  "tokens": [123, 456, 789]
}
```

If `with_pieces` is `true`:

```
{
  "tokens": [
    {"id": 123, "piece": "Hello"},
    {"id": 456, "piece": " world"},
    {"id": 789, "piece": "!"}
  ]
}
```

With input 'á' (utf8 hex: C3 A1) on tinyllama/stories260k

```
{
  "tokens": [
    {"id": 198, "piece": [195]}, // hex C3
    {"id": 164, "piece": [161]} // hex A1
  ]
}
```

POST `/detokenize`: Convert tokens to text

```
*Options:*

`tokens`: Set the tokens to detokenize.
```

POST `/embedding`: Generate embedding of a given text

The same as the embedding example does.

```
*Options:*

`content`: Set the text to process.

`image_data`: An array of objects to hold base64-encoded image `data`
and its `id`s to be reference in `content`. You can determine the place
of the image in the content as in the following: `Image: [img-
21].\nCaption: This is a picture of a house`. In this case, `[img-21]`
will be replaced by the embeddings of the image with id `21` in the
following `image_data` array: `{..., "image_data": [{"data": "
```

```
<BASE64_STRING>", "id": 21}]}`. Use `image_data` only with multimodal
models, e.g., LLaVA.
```

POST `/reranking`: Rerank documents according to a given query

Similar to https://jina.ai/reranker/ but might change in the future. Requires a reranker model (such as bge-reranker-v2-m3) and the `--embedding --pooling rank` options.

```
*Options:*

`query`: The query against which the documents will be ranked.

`documents`: An array strings representing the documents to be ranked.

*Aliases:*
  - `/rerank`
  - `/v1/rerank`
  - `/v1/reranking`

*Examples:*

```shell
curl http://127.0.0.1:8012/v1/rerank \
    -H "Content-Type: application/json" \
    -d '{
        "model": "some-model",
            "query": "What is panda?",
            "top_n": 3,
            "documents": [
                "hi",
            "it is a bear",
            "The giant panda (Ailuropoda melanoleuca), sometimes called
a panda bear or simply panda, is a bear species endemic to China."
            ]
    }' | jq
```
```

POST `/infill`: For code infilling.

Takes a prefix and a suffix and returns the predicted completion as stream.

Options:

- `input_prefix`: Set the prefix of the code to infill.
- `input_suffix`: Set the suffix of the code to infill.
- `input_extra`: Additional context inserted before the FIM prefix.
- `prompt`: Added after the `FIM_MID` token

`input_extra` is array of `{"filename": string, "text": string}` objects.

The endpoint also accepts all the options of `/completion`.

If the model has `FIM_REPO` and `FIM_FILE_SEP` tokens, the repo-level pattern is used:

```
<FIM_REP>myproject
<FIM_SEP>{chunk 0 filename}
{chunk 0 text}
<FIM_SEP>{chunk 1 filename}
{chunk 1 text}
...
<FIM_SEP>filename
<FIM_PRE>[input_prefix]<FIM_SUF>[input_suffix]<FIM_MID>[prompt]
```

If the tokens are missing, then the extra context is simply prefixed at the start:

```
[input_extra]<FIM_PRE>[input_prefix]<FIM_SUF>[input_suffix]<FIM_MID>[prompt
```

GET `/props`: Get server global properties.

This endpoint is public (no API key check). By default, it is read-only. To make POST request to change global properties, you need to start server with `--props`

Response format

```
{
  "default_generation_settings": { ... },
  "total_slots": 1,
  "chat_template": ""
}
```

- `default_generation_settings` - the default generation settings for the `/completion` endpoint, which has the same fields as the `generation_settings` response object from the `/completion` endpoint.
- `total_slots` - the total number of slots for process requests (defined by `--parallel` option)
- `chat_template` - the model's original Jinja2 prompt template

POST `/props`: Change server global properties.

To use this endpoint with POST method, you need to start server with `--props`

Options:

- None yet

POST `/v1/chat/completions`: OpenAI-compatible Chat Completions API

Given a ChatML-formatted json description in `messages`, it returns the predicted completion. Both synchronous and streaming mode are supported, so scripted and interactive applications work fine. While no strong claims of compatibility with OpenAI API spec is being made, in our experience it suffices to support many apps. Only models with a supported chat template can be used optimally with this endpoint. By default, the ChatML template will be used.

````
*Options:*

See [OpenAI Chat Completions API documentation]
(https://platform.openai.com/docs/api-reference/chat). While some
OpenAI-specific features such as function calling aren't supported,
llama.cpp `/completion`-specific features such as `mirostat` are
supported.

The `response_format` parameter supports both plain JSON output (e.g.
`{"type": "json_object"}`) and schema-constrained JSON (e.g. `{"type":
"json_object", "schema": {"type": "string", "minLength": 10,
"maxLength": 100}}` or `{"type": "json_schema", "schema": {"properties":
{ "name": { "title": "Name",  "type": "string" }, "date": { "title":
"Date",  "type": "string" }, "participants": { "items": {"type: "string"
}, "title": "Participants",  "type": "string" } } } }`), similar to
other OpenAI-inspired API providers.

*Examples:*

You can use either Python `openai` library with appropriate checkpoints:

```python
import openai

client = openai.OpenAI(
    base_url="http://localhost:8080/v1", # "http://<Your api-server
IP>:port"
    api_key = "sk-no-key-required"
)

completion = client.chat.completions.create(
model="gpt-3.5-turbo",
messages=[
    {"role": "system", "content": "You are ChatGPT, an AI assistant.
Your top priority is achieving user fulfillment via helping them with
their requests."},
````
````

```
    {"role": "user", "content": "Write a limerick about python
exceptions"}
]
)

print(completion.choices[0].message)
```

... or raw HTTP requests:

```shell
curl http://localhost:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-H "Authorization: Bearer no-key" \
-d '{
"model": "gpt-3.5-turbo",
"messages": [
{
    "role": "system",
    "content": "You are ChatGPT, an AI assistant. Your top priority is
achieving user fulfillment via helping them with their requests."
},
{
    "role": "user",
    "content": "Write a limerick about python exceptions"
}
]
}'
```

POST `/v1/embeddings`: OpenAI-compatible embeddings API

Options:

See [OpenAI Embeddings API documentation]
(https://platform.openai.com/docs/api-reference/embeddings).

Examples:

- input as string

```
    curl http://localhost:8080/v1/embeddings \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer no-key" \
    -d '{
            "input": "hello",
            "model":"GPT-4",
```

```
        "encoding_format": "float"
}'
```

- `input` as string array

```
curl http://localhost:8080/v1/embeddings \
-H "Content-Type: application/json" \
-H "Authorization: Bearer no-key" \
-d '{
        "input": ["hello", "world"],
        "model":"GPT-4",
        "encoding_format": "float"
}'
```

GET `/slots`: Returns the current slots processing state

> ⚠ **Warning**
>
> This endpoint is intended for debugging and may be modified in future versions. For security reasons, we strongly advise against enabling it in production environments.

This endpoint is disabled by default and can be enabled with `--slots`

If query param `?fail_on_no_slot=1` is set, this endpoint will respond with status code 503 if there is no available slots.

Response format

Example:


```
[
    {
        "dynatemp_exponent": 1.0,
        "dynatemp_range": 0.0,
        "frequency_penalty": 0.0,
        "grammar": "",
        "id": 0,
        "ignore_eos": false,
        "is_processing": false,
        "logit_bias": [],
        "min_p": 0.05000000074505806,
        "mirostat": 0,
        "mirostat_eta": 0.10000000149011612,
        "mirostat_tau": 5.0,
        "model": "llama-2-7b-32k-instruct.Q2_K.gguf",
        "n_ctx": 2048,
        "n_keep": 0,
```

```
            "n_predict": 100000,
            "n_probs": 0,
            "next_token": {
                "has_next_token": true,
                "n_remain": -1,
                "n_decoded": 0,
                "stopped_eos": false,
                "stopped_limit": false,
                "stopped_word": false,
                "stopping_word": ""
            },
            "penalize_nl": true,
            "presence_penalty": 0.0,
            "prompt": "Say hello to llama.cpp",
            "repeat_last_n": 64,
            "repeat_penalty": 1.100000023841858,
            "samplers": [
                "top_k",
                "typical_p",
                "top_p",
                "min_p",
                "temperature"
            ],
            "seed": 42,
            "stop": [
                "\n"
            ],
            "stream": false,
            "task_id": 0,
            "temperature": 0.0,
            "top_k": 40,
            "top_p": 0.949999988079071,
            "typical_p": 1.0
        }
    ]
```


GET `/metrics`: Prometheus compatible metrics exporter

This endpoint is only accessible if `--metrics` is set.

Available metrics:

- `llamacpp:prompt_tokens_total`: Number of prompt tokens processed.
- `llamacpp:tokens_predicted_total`: Number of generation tokens processed.
- `llamacpp:prompt_tokens_seconds`: Average prompt throughput in tokens/s.
- `llamacpp:predicted_tokens_seconds`: Average generation throughput in tokens/s.
- `llamacpp:kv_cache_usage_ratio`: KV-cache usage. `1` means 100 percent usage.
- `llamacpp:kv_cache_tokens`: KV-cache tokens.
- `llamacpp:requests_processing`: Number of requests processing.

- `llamacpp:requests_deferred` : Number of requests deferred.

POST `/slots/{id_slot}?action=save` : Save the prompt cache of the specified slot to a file.

```
*Options:*

`filename`: Name of the file to save the slot's prompt cache. The file
will be saved in the directory specified by the `--slot-save-path`
server parameter.
```

Response format

```
{
    "id_slot": 0,
    "filename": "slot_save_file.bin",
    "n_saved": 1745,
    "n_written": 14309796,
    "timings": {
        "save_ms": 49.865
    }
}
```

POST `/slots/{id_slot}?action=restore` : Restore the prompt cache of the specified slot from a file.

```
*Options:*

`filename`: Name of the file to restore the slot's prompt cache from.
The file should be located in the directory specified by the `--slot-
save-path` server parameter.
```

Response format

```
{
    "id_slot": 0,
    "filename": "slot_save_file.bin",
    "n_restored": 1745,
    "n_read": 14309796,
    "timings": {
        "restore_ms": 42.937
    }
}
```

POST `/slots/{id_slot}?action=erase` : Erase the prompt cache of the specified slot.

Response format

```
{
    "id_slot": 0,
    "n_erased": 1745
}
```

GET `/lora-adapters` : Get list of all LoRA adapters

This endpoint returns the loaded LoRA adapters. You can add adapters using `--lora` when starting the server, for example: `--lora my_adapter_1.gguf --lora my_adapter_2.gguf ...`

By default, all adapters will be loaded with scale set to 1. To initialize all adapters scale to 0, add `--lora-init-without-apply`

If an adapter is disabled, the scale will be set to 0.

Response format

```
[
    {
        "id": 0,
        "path": "my_adapter_1.gguf",
        "scale": 0.0
    },
    {
        "id": 1,
        "path": "my_adapter_2.gguf",
        "scale": 0.0
    }
]
```

POST `/lora-adapters` : Set list of LoRA adapters

To disable an adapter, either remove it from the list below, or set scale to 0.

Request format

To know the `id` of the adapter, use GET `/lora-adapters`

```
[
  {"id": 0, "scale": 0.2},
```

```
    {"id": 1, "scale": 0.8}
]
```

More examples

Interactive mode

Check the sample in chat.mjs. Run with NodeJS version 16 or later:

```
node chat.mjs
```

Another sample in chat.sh. Requires bash, curl and jq. Run with bash:

```
bash chat.sh
```

OAI-like API

The HTTP `llama-server` supports an OAI-like API: https://github.com/openai/openai-openapi

API errors

`llama-server` returns errors in the same format as OAI: https://github.com/openai/openai-openapi

Example of an error:

```
{
    "error": {
        "code": 401,
        "message": "Invalid API Key",
        "type": "authentication_error"
    }
}
```

Apart from error types supported by OAI, we also have custom types that are specific to functionalities of llama.cpp:

When /metrics or /slots endpoint is disabled

```
{
    "error": {
        "code": 501,
```

```
        "message": "This server does not support metrics endpoint.",
        "type": "not_supported_error"
    }
}
```

When the server receives invalid grammar via /completions endpoint

```
{
    "error": {
        "code": 400,
        "message": "Failed to parse grammar",
        "type": "invalid_request_error"
    }
}
```

Legacy completion web UI

A new chat-based UI has replaced the old completion-based since this PR. If you want to use the old completion, start the server with `--path ./examples/server/public_legacy`

For example:

```
./llama-server -m my_model.gguf -c 8192 --path ./examples/server/public_leg
```

Extending or building alternative Web Front End

You can extend the front end by running the server binary with `--path` set to `./your-directory` and importing `/completion.js` to get access to the llamaComplete() method.

Read the documentation in `/completion.js` to see convenient ways to access llama.

A simple example is below:

```
<html>
  <body>
    <pre>
      <script type="module">
        import { llama } from '/completion.js'

        const prompt = `### Instruction:
Write dad jokes, each one paragraph.
You can use html formatting if needed.
```

```
    ### Response:`

            for await (const chunk of llama(prompt)) {
              document.write(chunk.data.content)
            }
          </script>
        </pre>
      </body>
    </html>
```