Local LLM Inference

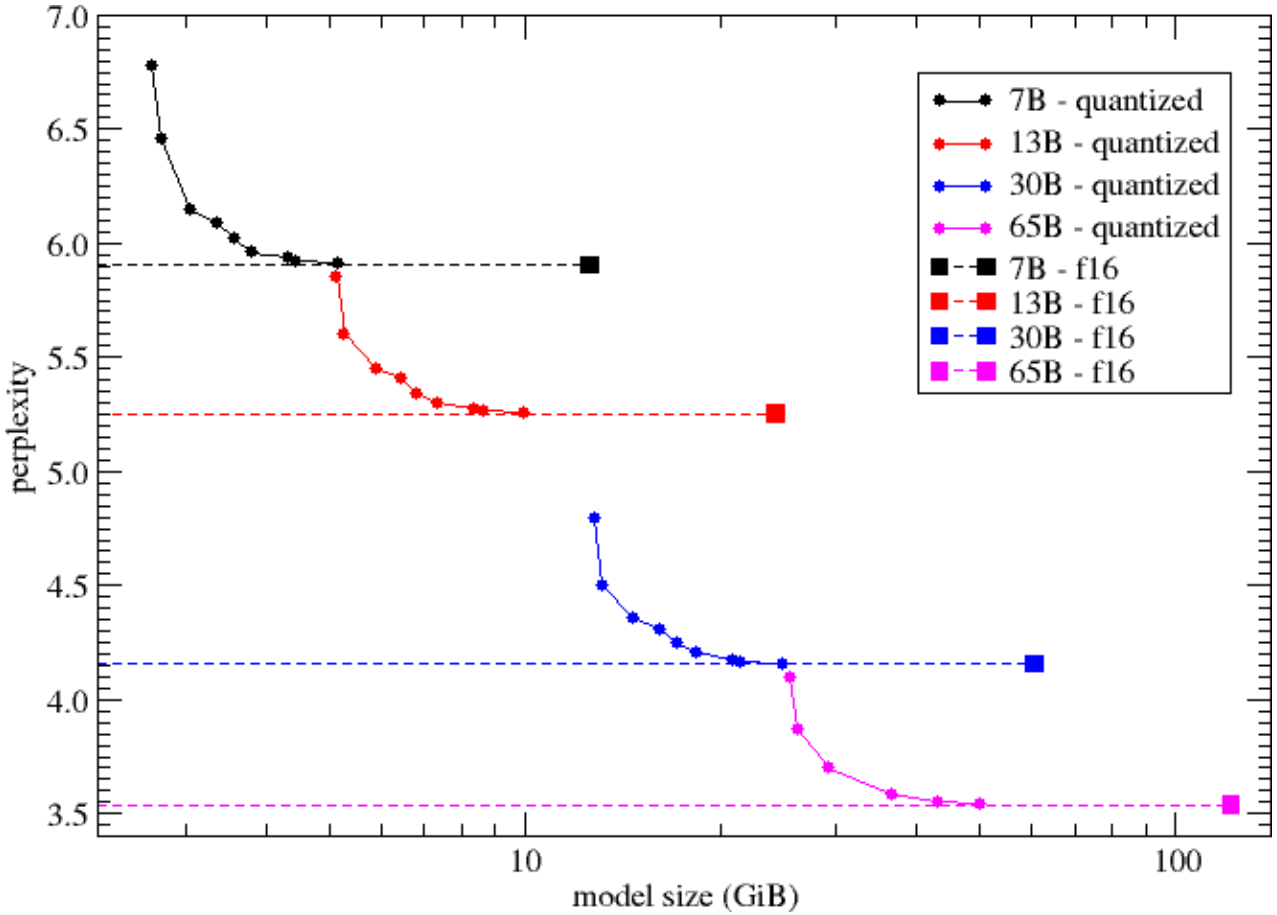
Feraidoon Mehri

Machine Learning Lab, Sharif University of Technology

February 6, 2024

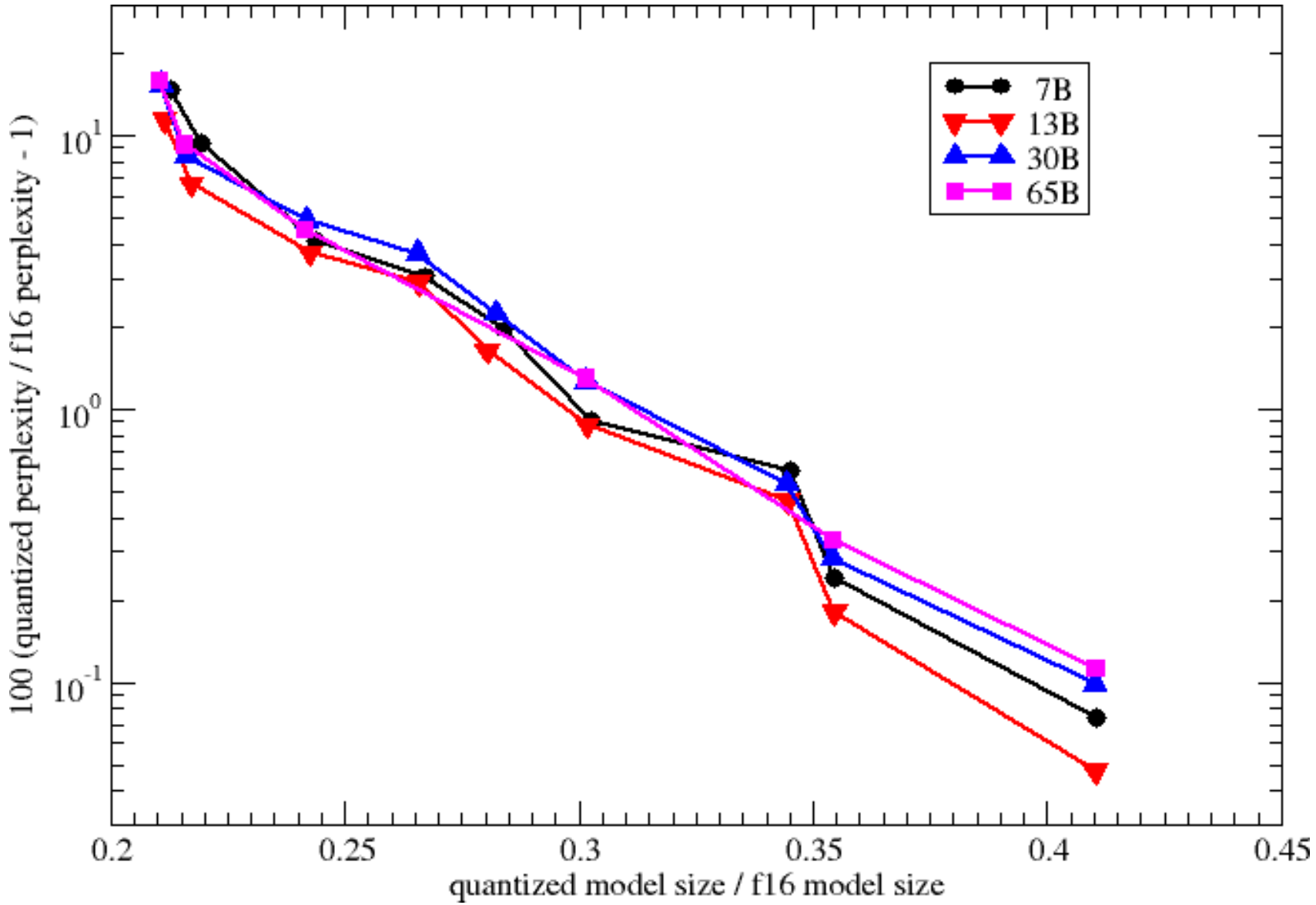
Outline

Perplexity on Wikitext as a Function of Model Size



The various circles on the graph show the perplexity of different quantization mixes. The different colors indicate the LLaMA variant (sizes) used. The solid squares in the corresponding color represent (model size, perplexity) for the original fp16 model. The dashed lines are added for convenience to allow for a better judgement of how closely the quantized models approach the fp16 perplexity.

Relative Quantization Error



Quantization Results

Model	Measure	Q2_K	Q3_K_M	Q4_K_S	Q5_K_S	Q6_K	F16
7B	perplexity	6.7764	6.1503	6.0215	5.9419	5.9110	5.9066
7B	file size	2.67G	3.06G	3.56G	4.33G	5.15G	13.0G
7B	ms/tok @ 4th, M2 Max	56	69	50	70	75	116
7B	ms/tok @ 8th, M2 Max	36	36	36	44	51	111
7B	ms/tok @ 4th, RTX-4080	15.5	17.0	15.5	16.7	18.3	60
7B	ms/tok @ 4th, Ryzen	57	61	68	81	93	214
13B	perplexity	5.8545	5.4498	5.3404	5.2785	5.2568	5.2543
13B	file size	5.13G	5.88G	6.80G	8.36G	9.95G	25.0G
13B	ms/tok @ 4th, M2 Max	103	148	95	132	142	216
13B	ms/tok @ 8th, M2 Max	67	77	68	81	95	213
13B	ms/tok @ 4th, RTX-4080	25.3	29.3	26.2	28.6	30.0	-
13B	ms/tok @ 4th, Ryzen	109	118	130	156	180	414

Time Per Token for WizardCoder-Python-34B

Model	wizardcoder-python-34b
Quant Method	Q4_K_M
Size	20.22 GB
Max RAM Required	22.72 GB
Description	medium, balanced quality - recommended
ms/token CPU 32 cores	316.71
ms/token CPU 1 core	3747.27
ms/token GPU	43.50

Note: the above RAM figures assume no GPU offloading. If layers are offloaded to the GPU, this will reduce RAM usage and use VRAM instead.

- Time per token measured on:
  - Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz
  - Quadro RTX 6000 24GB, compute capability 7.5

llama.cpp

LLM inference in pure C/C++

- Plain C/C++ implementation without dependencies
- 2-bit, 3-bit, 4-bit, 5-bit, 6-bit and 8-bit integer quantization support
- Mixed F16 / F32 precision
- CUDA, Metal and OpenCL GPU backend support
- Apple silicon first-class citizen - optimized via ARM NEON, Accelerate and Metal frameworks
- AVX, AVX2 and AVX512 support for x86 architectures

abetlen/llama-cpp-python: Python bindings for llama.cpp

Install

- Auto-detect hardware acceleration

```
pip install "llama-cpp-python[server]"
```

- CUDA

```
CMAKE_ARGS="-DLLAMA_CUBLAS=on" FORCE_CMAKE=1 pip install "llama-cpp-python[server]"
```

Start an OpenAI-compatible API server

```
python -m llama_cpp.server --model models/7B/llama-model.gguf --n_gpu_layers -1
```

- Navigate to `http://localhost:8000/docs` to see the OpenAPI documentation.

llama-cpp-python: Python API

```python
def print_chat_streaming(output):
    text = ""
    for r in output:
        text_current = None
        choice = r["choices"][0]
        if "delta" in choice:
            delta = choice["delta"]
            if "role" in delta:
                print(f"\n{delta['role']}: ", end="")
            if "content" in delta:
                text_current = f"{delta['content']}"
        elif "text" in choice:
            text_current = f"{choice['text']}"
        if text_current:
            text += text_current
            print(f"{text_current}", end="")

    text = text.rstrip()
    print("\n")
    return text
```

llama-cpp-python: Python API

```python
from llama_cpp import Llama, ChatCompletionMessage

llm = Llama(
    model_path="/models/wizardcoder-python-34b-v1.0.Q4_K_M.gguf",
    n_gpu_layers=0,
    n_threads=48,
)

output = llm.create_chat_completion(
    messages=[
        ChatCompletionMessage(
            role="user", content="Name the planets in the solar system?"
        ),
    ],
    stream=True,
)
print_chat_streaming(output)
```

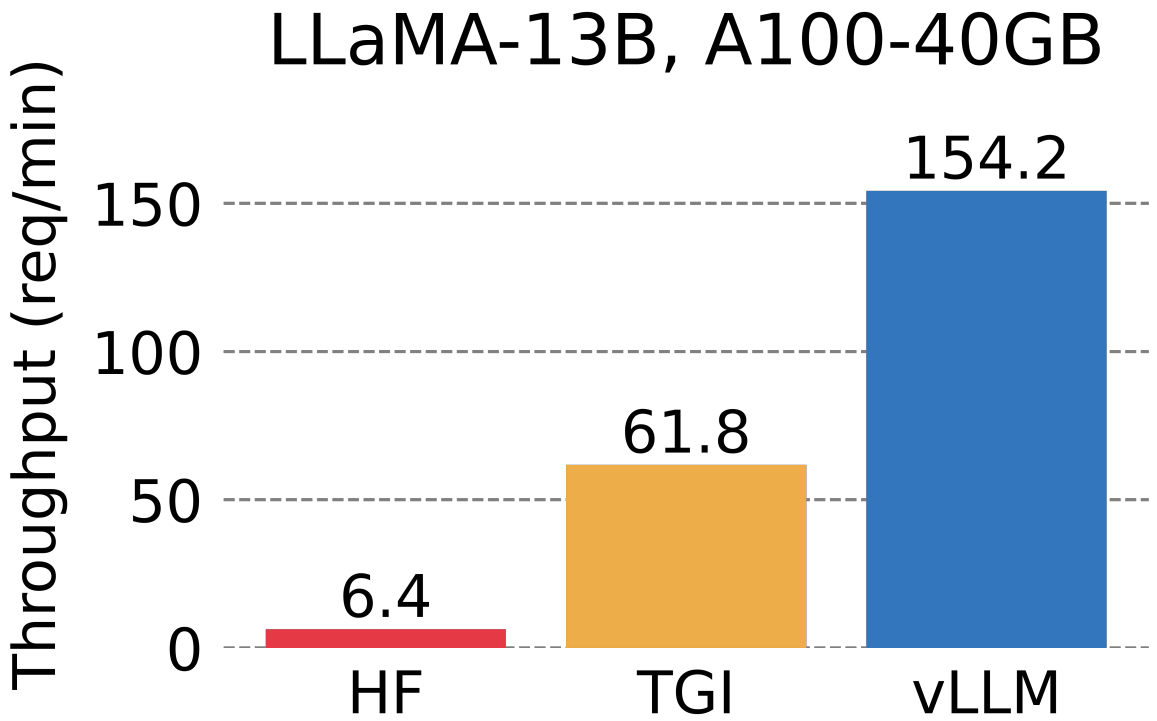
vLLM

vLLM is a fast and easy-to-use library for LLM inference and serving.

- Continuous batching of incoming requests
- Streaming outputs
- OpenAI-compatible API server
- State-of-the-art serving throughput
- Efficient management of attention key and value memory with PagedAttention
- Optimized CUDA kernels
- High-throughput serving with various decoding algorithms, including *parallel sampling*, *beam search*, and more
- Tensor parallelism support for distributed inference
- Seamless integration with popular Hugging Face models

vLLM Vs. HuggingFace TGI

According to vLLM authors, vLLM outperforms Hugging Face Transformers (HF) by up to 24x and Text Generation Inference (TGI) by up to 3.5x, in terms of throughput.



HuggingFace TGI

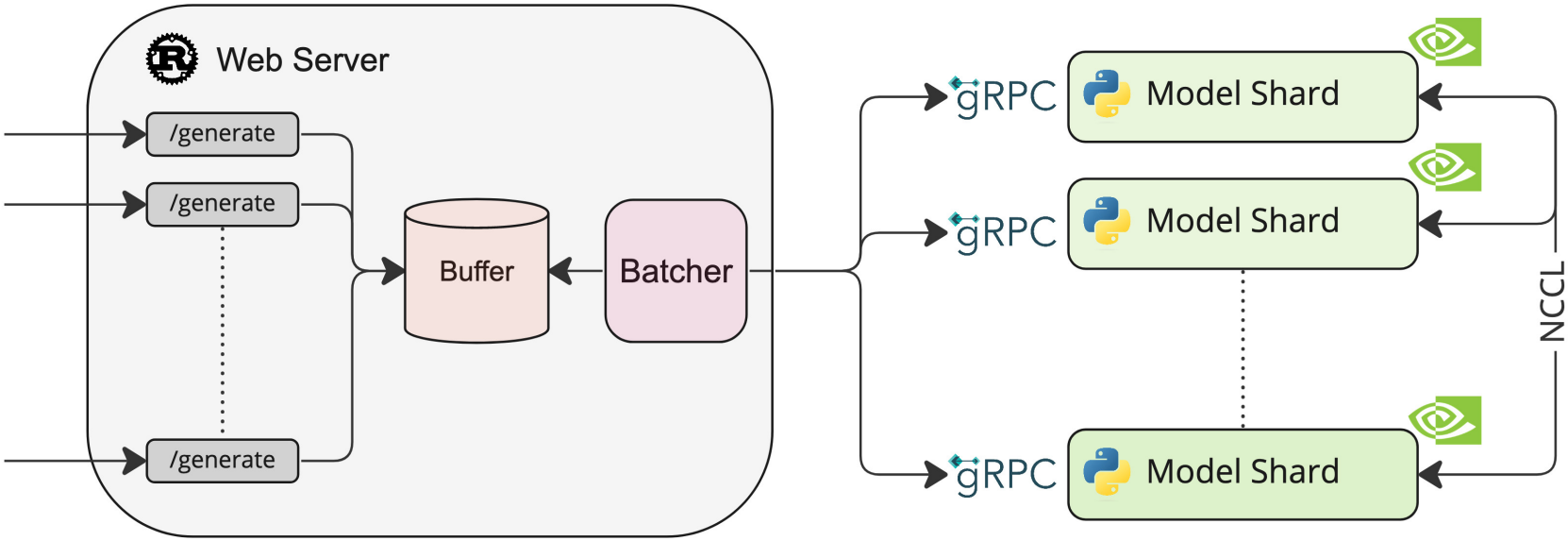
TGI: A Rust, Python and gRPC server for text generation inference. Used in production at HuggingFace to power Hugging Chat, the Inference API and Inference Endpoint.

- Serve the most popular Large Language Models with a simple launcher
- Tensor Parallelism for faster inference on multiple GPUs
- Token streaming using Server-Sent Events (SSE)
- Continuous batching of incoming requests for increased total throughput
- Optimized transformers code for inference using flash-attention and Paged Attention on the most popular architectures
- Quantization with bitsandbytes and GPT-Q
- Logits warper (temperature scaling, top-p, top-k, repetition penalty, more details see transformers.LogitsProcessor)
- Stop sequences

HuggingFace TGI

Text Generation Inference

Fast optimized inference for LLMs



Summary

- A server with 32 CPU cores and roughly 25GB RAM can do inference in almost real-time for a single user.
- A GPU server with 24GB VRAM is faster than real-time.
- Free online alternatives such as Claude V2 and ChatGPT 3.5 are still faster and better than the local models available.

Kaggle is Unviable

Kaggle only provides 20 GB of HDD, which is not enough to download the model weights.