

1. Inference Engines

1.1. llama.cpp

1.1.1. What is llama.cpp?

llama.cpp is a C++ implementation of Meta's LLaMA models designed for high efficiency and local execution. It allows us to run LLaMA models on a variety of platforms—Windows, macOS, and Linux—with the need for powerful GPUs or external dependencies.

Running LLaMA models locally gives us complete control over data privacy, performance tuning, and costs. We're not sending sensitive data to third-party servers or paying for API usage. It's especially beneficial for developers, researchers, and enthusiasts working on personalized AI applications.

Key features:

- **Cross-platform support:** Works on Linux, Windows, and macOS.
- **Optimized for CPUs:** No GPU required to run models.
- **Quantized models:** Reduced memory usage with minimal performance loss.
- **Python bindings:** Easily integrates with Python using llama-cpp-python.
- **Community support:** Actively maintained with frequent updates.

Next, let's understand how llama.cpp works.

1.1.2. How llama.cpp works

At its heart, llama.cpp is a lightweight, CPU-optimized inference engine built in C++ that enables the use of Meta's LLaMA language models entirely offline, with low resource requirements. It focuses on:

- Quantization to reduce model size
- Memory-mapped inference for efficiency
- Multithreading to utilize all CPU cores

Let's learn more about these focus points one by one.

Model quantization (Compression)

Large models like LLaMA 2 are gigabytes in size when using full-precision floats (FP16/FP32). To make them usable on machines with limited RAM or no GPU, llama.cpp supports quantized models in GGUF format.

These quantized models reduce memory usage and computation by using 4-bit, 5-bit, or 8-bit integers, e.g.:

- Q4_0, Q5_1, Q8_0 — different levels of quantization
- Smaller size means faster load time and lower RAM footprint

Memory mapping with mmap

llama.cpp uses memory mapping (`mmap`) to load models efficiently. Instead of loading the whole model into RAM, it streams only the parts needed at any moment. This:

- Minimizes memory usage
- Speeds up inference

- Makes large models possible on modest hardware

Tokenization and inference

Here's what happens when we send a prompt:

- **Tokenization:** The text prompt is broken into tokens using LLaMA's tokenizer.
- **Feed forward:** These tokens are passed through the neural network layers (transformers).
- **Sampling:** The model samples the next token using parameters like temperature, top_p, and stop.
- **Decoding:** Tokens are converted back to human-readable text.

This loop continues until the desired number of tokens is reached or a stop condition is met.

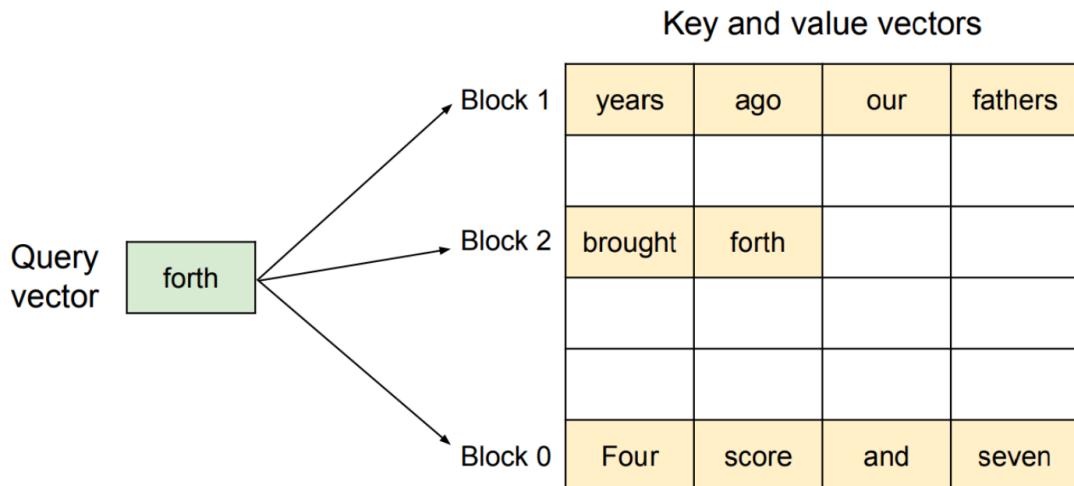
1.2. vLLM

vLLM is an open-source library designed for fast and efficient serving of LLMs. It's specifically optimized for high-throughput inference, making it popular for deploying language models in **production** environments.

The key problem vLLM addresses is the significant computational and memory resources LLMs typically require, especially when handling many user requests simultaneously. Traditional methods can be slow and lead to underutilized hardware.

Paged Attention is the core innovation in vLLM. Inspired by virtual memory and paging concepts in operating systems, PagedAttention manages the memory used by the attention mechanism in LLMs (specifically the KV cache) much more effectively.

Instead of forcing the key/value (KV) cache for every prompt to live in one long, contiguous buffer, it breaks the cache into fixed-size **pages** (GPU blocks) and keeps a lightweight page table that tells the attention kernel where each token's KV vectors live. The kernel can therefore jump to the right pages at run time, just as an operating system's MMU resolves addresses via virtual memory.



Page pool — KV tensors are stored in GPU memory in blocks (e.g., 16 tokens × hidden dim).

Page table — Each sequence keeps a small list mapping its logical token indices → physical pages.

Attention kernel — When a new query vector arrives (green box), the kernel:

- Looks up which pages contain the relevant keys/values (orange blocks).
- Streams those pages into on-chip registers/shared memory.
- Computes softmax-weighted sum exactly as in standard attention.

Because pages don't have to be adjacent (see Block 0 / 2 / 1 jumps in the figure), the runtime can pack thousands of requests densely even while they decode at different speeds.

1.3. SGLang

1.3.1. What is SGLang?

SGLang is a comprehensive framework developed for efficient serving of large-scale language and vision-language models. It is designed to enhance user control and interaction speed with the model through an optimized backend runtime and a user-friendly frontend language.

1.3.2. Why SGLang?

As generative AI models grow in complexity and size, efficiently serving them becomes a significant challenge, especially for low latency and high throughput. SGLang addresses these challenges by providing a highly optimized runtime that leverages GPU capabilities, making it possible to deploy state-of-the-art LLMs and VLMs effectively. SGLang is designed to simplify the deployment process and reduces operational costs.

1.3.3. Use Cases

SGLang is suitable for a wide range of applications, including:

- **Interactive AI Assistants:** Leverage fast response times for real-time interaction.
- **Multimodal Applications:** Seamlessly integrate vision and language capabilities to enable applications such as video captioning, interactive storytelling, and more.

- **Scalable Backend for Generative AI:** Deploy multimodal AI models in the cloud, to scale and support high throughput and large user bases.

1.3.4. Key Features of SGLang

Fast Backend Runtime

- **RadixAttention for Prefix Caching:**

RadixAttention structures and automates the reuse of Key-Value (KV) caches during runtime by storing them in a radix tree data structure. It retains and manages the KV cache after each generation request, allowing different prompts with shared prefixes to reuse the KV cache. The radix tree enables efficient prefix search, insertion, and eviction. This approach reduces redundant memory usage and computation time, enhancing performance without requiring manual configurations. In addition, Least Recently Used (LRU) eviction policy and a cache-aware scheduling policy implemented in RadixAttention manages the GPU memory constraints and improves cache hit rate.

- **Jump-Forward Constrained Decoding for Efficiency:**

Jump-Forward Constrained Decoding allows the model to skip unnecessary computations during generation.

- **Continuous Batching for Optimal GPU Utilization:**

Continuous Batching dynamically adjusts batch sizes to optimize GPU utilization.

- **Optimizations:**
 - **Paged Attention:** Reduces memory usage by partitioning the attention matrix into manageable blocks or pages, allowing the model to handle longer sequences efficiently without exceeding memory limitations.
 - **Tensor Parallelism:** Distributes the computation of neural network layers across multiple GPUs by splitting tensors along specific dimensions, enabling parallel processing of large models that exceed the memory capacity of a single GPU.
 - **FlashInfer Kernels:** Highly optimized GPU kernels designed for rapid inference. They leverage low-level hardware optimizations to accelerate common neural network operations, significantly reducing latency and improving throughput.
 - **Quantization:** Reduces the precision of model parameters and computations (e.g., using 8-bit integers instead of 32-bit floats), model size, and computational load with minimal impact on accuracy. SGLang supports various quantization methods, including Activation-Aware Weight Quantization (AWQ) and Generative Pre-trained Transformer Quantization (GPTQ), and utilizes data types such as INT4 and FP8. These quantization methods and data types can be mixed and matched to optimize model performance and efficiency.

1.4. llama.cpp vs. vLLM vs. SGLang

Feature / Framework	llama.cpp	vLLM	SGLang
Primary Goal	Make LLMs run anywhere, especially on CPUs and consumer hardware.	Maximize throughput & cost-efficiency for high-volume serving on GPUs.	Maximize performance for complex, multi-step LLM workflows.
Core Architecture	Efficient C/C++ library with minimal dependencies.	High-performance inference server & engine for GPUs.	Domain-specific language + runtime designed for complex prompting.
Key Innovation	Quantization & GGUF format for small model sizes and broad hardware support.	PagedAttention for near-zero memory waste in the KV cache.	RadixAttention for automatic reuse of common prompt prefixes.
Hardware Sweet Spot	CPUs, Apple Silicon, Edge Devices. Broad GPU support via backends.	NVIDIA GPUs (optimized). Also supports AMD/Intel via community backends.	NVIDIA GPUs (optimized). Built for data center/cloud environments.

Performance Focus	Low-latency per request on constrained hardware.	High throughput (requests/second) for many concurrent users.	Low latency for complex prompts (e.g., agents, few-shot, structured output).
Model Format	GGUF (its own optimized, quantized format).	Hugging Face transformers format (e.g., .safetensors).	Hugging Face transformers format; also supports GGUF via Llama.cpp backend.
Ease of Use	Command-line tools; many friendly UIs built on top (e.g., LM Studio).	Simple pip install; runs as an OpenAI-compatible API server.	Requires learning its SGLang language (Python-based) for advanced features.
License	Permissive MIT license.	Permissive Apache 2.0 license.	Permissive Apache 2.0 license.

2. Model Used

- Qwen3-VL-8B-Instruct-GGUF:Q4_K_M

3. Usecase

- Unstructured Text Extraction

4. Inference Engine

- llama.cpp

5. Model Performance

5.1. English Printed

Image Text OCR

Start New Chat with Image or PDF

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG, PDF

Browse files

MobileNetV1.png 64.2KB

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5. Conv dw / s1	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000

Context Image

Extract data from this image

[Extract data from this image](#)

[Table 1. MobileNet Body Architecture](#)

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv dw / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
5x Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv / s2	$3 \times 3 \times 512$ dw	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7x7	$7 \times 7 \times 1024$
FC / s1	1024x1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

5.2. English Handwritten



Context Image

[note down the text from this image](#)



note down the text from this image



The Last Lesson Alphonse Daudet (1840-1897) (French novelist & short story writer. Days of Franco Prussian war (1870-1871))

Characters M Hamel Franz Old Hauser Farmer past master Blacksmith / Watcher

Usual things No bustle, No sound of opening and closing desk, lessons repeated in unison teachers ruler sappington table. It was quiet as Sunday morning. classmates were in place. Changed M Hamel

MHamel didn't scold Franz. He wore his best clothes

Morning scene Franz late for school A test on participle Wanted to spend day out It was warm & bright birds were chirruping Russian soldiers were drilling in back of gownie! Beautiful green coat, frilled shirt, black silk cap all embroied.

Bulletin board All bad news came there last battles, the draft, commanding officer Blacksmith and Watcher with apprentice To Franz Don't go so fast you have time. Order from Berlin

An order from Berlin to teach only German in schools of Alsace and Lorraine

5.3. Nepali Printed

Image Text OCR

Start New Chat with Image or PDF



Drag and drop file here

Limit 200MB per file • PNG, JPG, JPEG, PDF

[Browse files](#)



nepali_100_rupee.jpg 28.6KB



Context Image



Perform OCR



श्री नेपाल राष्ट्र बैंक नेपाल सरकारको अनुमति गर्ने नेपाल सरकारको अनुमति गर्ने बाना भएको 100 हुपेयाँ एक सय तुरुन नामन ६१ LUMBINI THE BIRTH PLACE OF LORD BUDDHA E08999
100

5.4. Nepali Handwritten



Note down the information

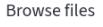
ख्यो थुङ्गा फुलका हामी रुहै माला नैपाली साविग्रों म अङ्ग फिलिएका मैंची महाकाली ।
प्रकृतिका कोटी कोटी समपदाको आँचल वीरहरुका रगतले, रवतन्द्र अतल ।
ज्ञानभूमि, शादित्भूमि तराई, पहाड, हिमाल अखण्ड यो प्यारो हासी जात्भूमि नैपाल ।
बहुल जाति . 2

5.5. National ID Structured

Image Text OCR

Start New Chat with Image or PDF

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG, PDF



 nepali_id.jpg 87.8KB 



Context Image

 Perform OCR



Perform OCR

```
▼ {  
    "nationality" : "Nepalese"  
    "sex" : "F"  
    "surname" : "कोइलरा पोखरल"  
    "given_name" : "भगावती कुमारी"  
    "mother_name" : "सरित कुमारी पोखरल"  
    "father_name" : "विन्दु प्रसाद पोखरल"  
    "date_of_birth" : "1978-02-05"  
    "date_of_issue" : "2017-01-01"  
    "national_id_number" : "023-456-2130"  
}
```