AI Server White Paper
Automated Sciences
www.automatedsciences.com
October 2026

This is a completely Open, originally written by a human, based on days of research, free use paper: Any part of this paper can be used for AI Model Training or copied and distributed in its entirety or in parts by any party WITHOUT permission or attribution. We are not responsible in any way for mistakes or inaccuracies in this paper.

We rent x86 based AI Server at very low pricing visit our web site at www.automatedsciences.com for more information.

Introduction

A friend of mine recently commented that they asked Chat-GPT about themselves and its response was "tell me about yourself"! I then explained to him that Chat-GPT does not know anything more about you than a stranger does! As many people have the same name it cannot readily look up this info on the internet and even with additional details, birthdate, state, etc there is only limited publicly available information out there on most people. Most AI models are knowledgeable in the structure and common words of one or more human or programming languages and basics of science, human health, psychology, economics, human interaction, and generally how the world works. Larger models (more parameters) have significantly more knowledge of the above and other. They also know how to search the web or their own knowledge base but in many cases can only determine if the search results seem related and summarize these in proper prose.

So how do you make these AI models more knowledgeable in specific organizations, industries, and subjects. There are basically two ways. The first being you can feed in a limited amount of information with the prompt through Retrieval-Augmented Generation (RAG) which involves a knowledge base query or another specially trained AI finding information pertinent to your prompt and presenting it along with your prompt. Allowing the AI model to use this additional information throughout the entire session (entirety of the conversation). **The second being you can add thousands of "facts" about your company, organization, or industry to the model. Training an open weight model running on your own AI server.** The second method is much preferred for static data (does not change over short periods of time) as it is far more efficient with much more additional information capacity and makes your custom model into an "expert" on your organization and its common workflows. The RAG technique though is the best way to introduce updated dynamic data and can help "focus" the models output. Both techniques can and should be used together.

You cannot get this same additional training capability using a closed online model like Chat-GPT, Claude, or Gemini as these share the model parameters in the AI accelerator memory between thousands of subscribers and cannot customize the parameters per user. The newer models do have "experts" built in that MAY posses "publicly" available information covering the topic of your prompt but insufficient industry information and no organization specific information. They also usually have large context sizes and allow for prompts that are pages long allowing the user to privately introduce either manually or through RAG a limited amount of new information for the model to use when processing subsequent prompts in a users session.

Your own AI Server

Below I have a long section to clearly describe the pros and cons of doing AI inference on a large memory multi-core CPU. **It should enlighten the reader and potential renter of the value of renting your own AI server or accelerator esp when the AI application or agent needs a large number (1000's) of "facts", many times company/organization specific and proprietary, added to the AI model to be of value.** Adding custom training is by far the best way to achieve model customization as once trained it will have NO impact on the inferencing performance, will be easy to use by members of your organization, and minimize or eliminate the need for a RAG knowledge-base.

Instead of or in addition to training custom "facts" you can introduce facts esp things like recent news and updates through prompt augmentation, known as RAG, where your prompt is first used to "lookup" information in a data/knowledge base that is usually somewhat aware of what the AI model already knows and what updated or additional information should concisely (to limit context overflow) be provided in the prompt to help ensure an accurate result particularly for present day. All the information in the model parameters is from a point(s) in time (potentially months old) when the model was trained. The initial idea behind the RAG is based on the decades old adage of do not talk like an expert on a subject until you have read todays newspaper!

CPU versus GPU for AI inference

Lets first look at the differences between CPU's and GPU's. I want to note that I consider a GPU to be different from a modern (Hopper or better) AI accelerator.

AI accelerators are in there own class with high performance calculations, 5X-10X a GPU, >80GB of very high speed (>2000GB/sec) RAM, and the ability to communicate at extreme speeds (>500GB/sec) with a bunch of other accelerators for rapid AI training. They cost $30K-150K USD per unit, are currently backordered for many months, and have rental costs of >$5000 dollars per month.

Now that I have defined a GPU, CPU versus GPU differences. Typical GPU offers 5-50X faster vector calculation speed than a modern server grade CPU, but as described below for inference this factor makes little difference. **CPU systems RAM can be upgraded to 512GB facilitating both a large LLM model, reasonable to huge context length, and a significant batch size.** Where a GPU's RAM cannot be expanded and in the case of modern GPU's, even Pro series, is limited to 32GB. Pro GPU memory access speed is typically 4X-8X faster than a modern server grade CPU. This has definitive speed (both latency and throughput) advantages but as discussed below this matters little if the model and context cache(s) cannot fit in the GPU memory. As a side note a reader of this commented that GPU's can access system memory giving them all of CPU memory to use. This is true, but the access to CPU memory from the GPU even in large blocks is limited by the PCIe bus speed to about 30GB/sec, much slower than the CPU can access its own memory.

So for small models and small context sizes and one or low batch sizes, a GPU can be used and will provide low latency, a highly desirable attribute for robotic control or a basic interactive AI assistant, but not adequate for most agentic tasks where an AI accelerator, or if latency is not an issue a CPU based system is necessary to provide sufficient memory for model parameters and contexts.

As CPU based systems have much more memory than GPU based they can support a large batch size as the typical 1-8GB context cache per session can add up to a lot of memory.

When prompt processing prefill phase requires a lot of compute and time required is a function of processing speed and prompt length (# input tokens) and will be much slower on CPU. However if you have custom trained your model, prompts can be much shorter than relying on RAG and associated much larger prompts and thereby reasonably fast. The decode phase speed is a function of model size (# parameters) and memory access speed. With no batching it is typically 3-6 times faster on a GPU but as the GPU's memory is typically not sufficient to support a batch size greater than four the CPU can be just as fast as it with the **much larger memory larger batch sizes (10-30) are possible**.

Lastly I will note that throughput with a modern multi-core CPU and batching can be just about as fast as a GPU. Latency with a CPU is much worse, 3-20X slower than a GPU and not appropriate when realtime or fast responses are needed. **However keep in mind that the entire point of using a CPU based AI Server is to perform agentic tasks that need a large (30B+ parameter) model and a large context size and this does not fit in GPU memory.**

There are many agentic, software generation, and image generation applications that do not require quick product generation and the output quality is far more important than quick output or low latentcy.

CPU for AI Training

Many of you have read that a CPU is way too slow to train AI. **This is true that a CPU based system would take decades to ingest and process the hundreds of billions to trillions of tokens required to train an LLM neural network starting from an empty state.**

However there are dozens of fully trained open weight LLM's, most freely available, that a developer can build upon by training the necessary additional facts. Integrating a few million additional parameters to the billions already available in the LLM download. This process can typically be done in minutes of compute time on a modern AI accelerator. **It can however also be done on a single CPU AI server like the ones we have available for rent here. It will take hours to days on a CPU to train these additional "facts".**

If you want this additional training to be done faster along with fast testing and further training updates you could also rent an AI accelerator from a hyper-scalar like AWS for a day at under $300 to quickly train the additional facts into the chosen LLM. Then download the newly enhanced LLM parameters file with the additional "facts" onto your CPU based AI server system for day to day inferencing. As it is all digital the model will operate identically on the AI server it will just be much slower.

However you achieve the custom training, once done, the AI model will possess this knowledge permanently without long and costly prompts, prompt engineering, and reducing or eliminating the use of RAG. It will also reduce the needed context size (more on this in the technical section) which reduces the memory needed to support multiple active sessions and the concurrency benefit.

Technical Explanation

This section need not be read unless you want the gritty details of how it all works. Please note what I have here is believed to be accurate in 2026 but has not been peer reveiwed, it is just a summary, and as new optimization techniques are devised this may become stale over time.

In the examples given here I have intentionally avoided giving times as times are highly dependent on CPU/GPU speed, number of cores or compute units, and memory access speed instead I put all compute requirements as MAD (multiply accumulate) operations. You can look up in our AI Server offering table the possible MAD/sec for both 8 bit integer parameters iDP/s (integer Dot Product/second) used in the decode phase and half/single precision floating point parameters fpMAD/s (floating point multiply accumulate/second) typically used in prefill phase. This table also contains the memory access speed. A M after a number is Million (10^6), a G after a number is billion/giga (10^9), and a T after a number is trillion (10^12).

We will look at inference speed, lets separate latency from throughput. Latency in the case of AI LLM inference is the time from presentation of a prompt to the output of a group of tokens that represents a response to the prompt. Throughput is the number of possible output tokens per second.

Throughput is straightforward to evaluate and is basically a function of the memory access speed multiplied by the batch size and except for high batch sizes is independent of the CPU/GPU compute performance.

Latency is much more complex as in some cases the prompt may be pages of text that not only pose a question or present a task but contain much of the background information, either user provided or augmented by RAG, that the AI model needs to create an informed answer.

All the input tokens must go through a prefill phase that takes (#parameters * #input_tokens * batch_size) / fpMAD/s calculation time. For example a 32B parameter model with 3000 input tokens per prompt and a batch size of 4 takes 384T fpMAD calculations to prefill. This will be much faster on a GPU than a CPU and on a CPU with a modest (4-8) batch size can represent more than half of the total latency time. I will note though that for agentic tasks assuming cross query caching is used, subsequent prompts required to complete an agentic task tend to be fast as only a few hundred additional input tokens usually need to be processed.

Once initial prefill is complete decode begins. Most of the latency in agentic AI is in token generation and is bound by memory access speed. This is due to the fact that each output token requires the entire parameter set and context cache to be read from RAM.

Continuing the above example if the response to the prompt(s) is 1000 tokens of output and there are 32B parameters a byte each, each generated token will take 32B divided by the memory access speed in GB/sec. The total time is the time per token multiplied by the number of output tokens. In this example generating the 1000 tokens will take 32T+ memory accesses. So on the AI server we offer for rent with memory access speeds of 60-90GB/sec generating 1000 output tokens will take 6-8 minutes. Same example on a Hopper (modern AI accelerator costing about $40K USD) with 3000GB/sec memory access times would be under 12 seconds entirely due to the much faster memory access time.

However putting speed particularly latency aside, for many AI applications the quality of results is more important and this is where amount of memory plays a big role and favors the use of CPU's that can have up to 512GB of installable upgradeable memory versus a Pro GPU that typically has 16-32GB of fixed RAM albeit at higher access speed. It is well known that models with more parameters give better results as they are capable of holding more information about language and linguistic variation and more knowledge of particular subjects and the world giving context to the language.

The other important point for AI inferencing is maintaining the context. Context is the information carried from prompt to prompt or presented by the RAG. Think of a conversation where each conversant is expected to remember what they read on the subject and what was discussed at the start of the conversation minutes ago. A couple of years ago a 5000 token context cache was considered adequate and this will take 2-60 seconds per prompt to create on a CPU/GPU and 200MB-2GB (dependent on #attention layers) of RAM to cache so that the generation of each output token is as fast as memory access will allow. More recently for agentic tasks the model must have more context to be able to complete the given task. Unfortunately if you quadruple the context length to 20K tokens you also increase the prefill CPU/GPU time and the memory used by 4X.

To increase throughput, esp with $$$$$ AI accelerators, you want to process a bunch of completely separate sessions (think trains of thought) at same time. The LLM parameters in the decode phase can be read for all sessions generating a token for each session so that the overall tokens/sec generated (total throughput) increases by a factor of number sessions this is known as batching. However keep in mind that each additional session needs its own context cache that for a 20K token context would consume 1-8GB of RAM per session so for 4 sessions it could take up to 32GB (4x8) of RAM. So RAM used is context cache size multiplied by concurrent sessions plus the memory used for the model parameters. So the desirable large model combined with the necessary context cache memory usage per session has steep RAM requirements that exceeds what Pro GPU's can handle and leaves only CPU's and $$$$$ modern AI accelerators for agentic AI.

Most of the AI servers (ones with 96GB of RAM) we offer for rent can accommodate a 50B parameter model assuming 8 bit parameters (BTW watch out using 4 bit parameters many models will not provide accurate results) with a 80K token context length with a batch size of 1-2 or the same 50B parameter model with a 20K token context length and a batch size of 8-16. Latency (time to first output token) will be seconds to minutes depending on #input and output tokens with a throughput of 2-3 tokens/second/session so a thousand token output response will take 6-9 minutes. This time will not increase much though for a batch size of 2, 4, 8, or even 16 on a Xeon CPU. **So total throughput can be up to 48 tokens/second, but only if you can keep the batch size * context cache size plus model parameters size to the point where it fits in 90GB of RAM.**

Lastly a potential customer asked can I aggregate the RAM in servers with multiple processors (packages) or across servers like has been done for decades for high performance computing applications. The answer is NO for AI inferencing as noted earlier memory access speed is key but in a server with two CPU packages the CPU's can only communicate with each other at about 10GB/sec and between servers even less! Creating an unacceptable bottleneck. **Bottom line all the AI inferencing processing should be done by one CPU with all the RAM on that CPU and AI processing bound to this CPU.** If another CPU is available other standard processing or a completely separate and smaller companion AI model can be run on this.