What is aiDAPTIV?

How does elastic fine-tuning work?

PHISON

Large Language Model Fine-Tuning

When one interacts with an AI chatbot or agent, like ChatGPT, Google Gemini, Apple Intelligence, or Microsoft Copilot, it references a pre-trained large language model, or LLM, to provide responses. These foundation LLMs are generally pre-trained with immense amounts of data but have a broad focus.

Fine-tuning leverages the foundation LLM as a starting point and continues training it on task-specific and/or domain-specific data, instead of starting completely from scratch. A new and enhanced specialized model checkpoint optimized for a specific use case is created once the fine-tuning training is completed.

Why Fine-Tune?

While most of the LLMs knowledge is learned during pretraining, the foundation models are trained as a jack of all trades, but master of none. Fine-tuning takes that foundation model and adapts it for specific tasks, like summarization, QA, and code generation. It can also optimize the model for domain specific language for legal, medical, and enterprise use cases.

Altogether, fine-tuning has high GPU memory requirements so model size, batch size, and context length are heavily dependent on how much GPU memory is available on a system, not the amount of compute power available.
Phison's Pascari aiDAPTIV technology aids these GPU memory limitations by adding cache memory to the mix.

More fine-tuning training data improves accuracy, relevance, and consistency for those specific tasks. It can also train how the model responds, such as tone, style, output format, and instruction following behavior. Fine-tuning a model with voice lines from a specific character or celebrity results in outputs that reflect the tone and style of the persona of the training data and enable consistent responses across repeated use cases.

Essentially, fine-tuning adjusts behavior and task performance instead of core knowledge, enabling it to master the information it learned previously for your desired use case, and respond the way you want it to.

How Fine-Tuning Works

The concept of fine-tuning is fairly simple – feed the model more data so it can learn and become better. Datasets are labeled as prompt and response pairs in a format that includes (question answer) (instruction response) (prompt response) during the training process.

These dataset pairs include instructions, chat style conversations, and domain specific examples. Here, high-quality data is more important than dataset size and effective datasets include diverse examples, edge cases, and clear mappings. Small, high-quality datasets are sufficient for alignment tasks while large datasets are required for knowledge-heavy tasks.

The training loop takes the dataset and performs a forward pass to generate an output, compute the loss of the generated output compared to the ground truth, then backpropagation to update weights. This loop iterates over datasets for multiple epochs.

A new model, or adapter weights, is the resulting output after multiple epochs of this fine-tuning process. This model is now specialized for the target task or target dataset and now ready to respond with greater nuance than before.

Types of Fine-Tuning

Not all fine-tuning is equal. Full fine-tuning updates all the parameters in a model and has the highest performance potential. However, full fine-tuning is a demanding process that requires a large amount of GPU memory, high-compute capabilities, significant storage space, and a ton of time – it's not the most efficient method of fine-tuning.

There's another method of fine-tuning – Parameter-Efficient Fine Tuning, or PEFT. This fine-tuning method keeps the base model frozen but updates a small subset of parameters for greater efficiency, since it has lower memory usage, is faster to train, and easier to deploy. Examples of PEFT models include LoRA, QLoRA, and Adapters.

Copyright ©2026 Phison Electronics | All Rights Reserved.

Compute and Memory Demands

Training components include model weights, gradients, optimizer states, and activations, while gradients are about equal to weight sizes. Optimizer states are about 2-4 times the size of the weights and activations depend on batch size and sequence length.
Altogether, fine-tuning has high GPU memory requirements so model size, batch size, and context length are heavily dependent on how much GPU memory is available on a system, not the amount of compute power available. Phison's Pascari aiDAPTIV technology aids these GPU memory limitations by adding cache memory to the mix.

Key Challenges

There are many key challenges when it comes to fine-tuning. Memory constraints restrict scalability when large models exceed available GPU memory while fine-tuning demands long compute training cycles that depend on expensive infrastructure.

High-quality labeled datasets also require data preparation – both are non-trivial requirements. Not everything is sunshine and roses with fine-tuning, however. Models may lose general capabilities when over specialized or tuned too finely.

Fine-Tuning vs Alternatives

Fine-tuning has many advantages, including persistent model behavior and low inference latency. The embedded knowledge is also in the weights. Alternatively, prompt engineering does not require training at the expense of consistency, due to temporary behavior control. Retrieval-augmented generation on the other hand, relies on external knowledge sources, receives dynamic updates, at the cost of higher inference latency.

When dealing with repetitive tasks at scale that require consistent behavior with low latency, fine-tuning is the best solution. Prompt engineering and retrieval-augmented generation alone cannot match fine-tuning's consistency and latency advantages for repetitive, predictable tasks, though all three approaches are frequently combined in production systems.

Pascari aiDAPTIV Fine-Tuning Performance

This test provides a baseline set of parameters to compare results across several different system configurations. This includes GPUs, system memory density and speed, CPUs, etc...

Test System

- Intel W5-3435X CPU
- 512GB DDR5 (16x 4800MT)
- Phison E18DC 4TB Boot Drive
- 2x Pascari AI100 2TB (LVM RAID 0)
- Ubuntu 24

Test Settings

- GPU: NVIDIA with CUDA Support
- **Epoch: 1**
- Train Batch Size: (depends on available GPU memory)
- Total Batch Size: (number of GPUs x Train Batch Size x 10)
- **Max Sequence Length: 12000**
- **Learning Rate: 0.000007**
- **Triton: On**

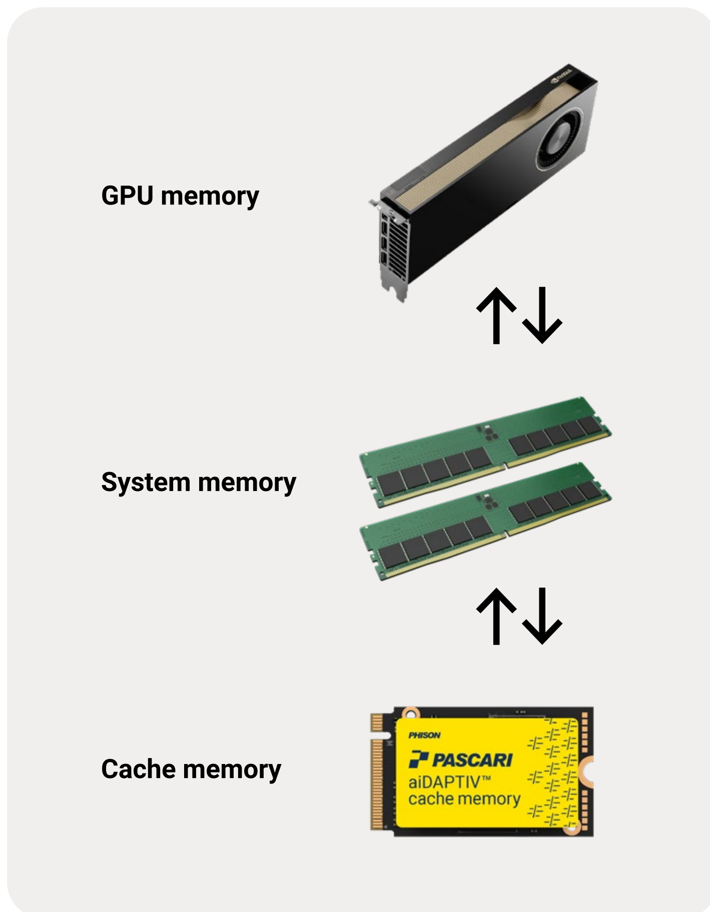
6000 Pro Workstation	8B Model			Mem
	Time	Tokens/S	Batch	
1x	0.31:04	4481	36 / 360	96
6000 Pro MaxQ	**8B Model**			**Mem**
	Time	Tokens/S	Batch	
1x	0.40:28	3402	36 / 360	96
6000 ADA	**8B Model**			**Mem**
	Time	Tokens/S	Batch	
1x	1:14.40	1833	24 / 240	48
4x	0:25.05	5498	24 / 960	196
5000 ADA	**8B Model**			**Mem**
	Time	Tokens/S	Batch	
1x	1:03.08	2207	16 / 160	32
4x	0:20.03	7282	16 / 640	128
4000 ADA	**8B Model**			**Mem**
	Time	Tokens/S	Batch	
1x	2:02.36	1125	8 /80	20
4x	0:33.07	4313	8 / 320	80
RTX 5090	**8B Model**			**Mem**
	Time	Tokens/S	Batch	
1x	0:46.19	3010	16 / 160	32
RTX 5080	**8B Model**			**Mem**
	Time	Tokens/S	Batch	
1x	1:29.27	1528	6 / 60	16
RTX 4070 Ti Super	**8B Model**			**Mem**
	Time	Tokens/S	Batch	
1x	1:54.34	1191	6 / 60	16
4x	0:32.24	4226	6 / 240	64

 Copyright ©2026 Phison Electronics | All Rights Reserved.

Meet Pascari aiDAPTIV

There's a secret cheat code to the high GPU memory demands of fine-tuning – meet Pascari aiDAPTIV. By combining the high-performance and capacity of Pascari SSDs with a middleware secret sauce, aiDAPTIV bridges the gap between fine-tuning capabilities and available GPU memory to let users work do more with less.

The core system architecture for aiDAPTIV elastic fine-tuning is basic – extend GPU memory with a multi-tier memory hierarchy. This approach has three tiers – GPU memory for compute, system memory as a staging buffer and cache, and aiDAPTIV cache memory as large-capacity storage for model data.

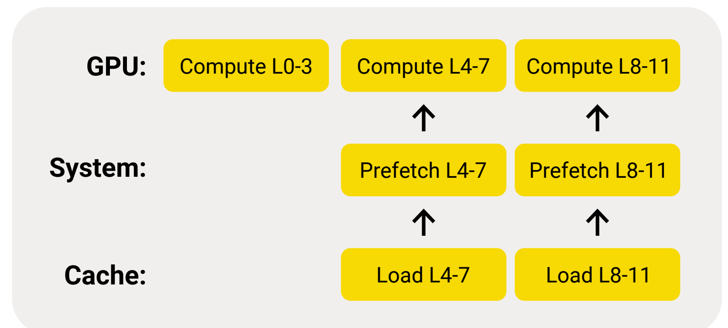


Model weights are primarily stored in the cache memory instead of completely in the GPU memory, so only a portion of the model resides in GPU memory at any given time. Pascari aiDAPTIV dynamically moves model data between the three memory tiers during training for enhanced capabilities with zero user intervention.

Take a Slice

Pascari aiDAPTIV slices up the entire model into small groups of layers versus loading the entire model at once. A typical execution unit consists of multiple layers – 3-4 layers per group. Layers are loaded from the cache memory into system memory then system memory to GPU memory for computation during the forward pass.

As the GPU computes the current group of layers, the next group of layers is prefetched from the cache memory into the system memory. The system moves onto the next group of layers once computation of the layers in system memory is finished. This data movement and compute pipeline have partial overlaps and appears seamless to during compute.



Selected layers are retained in GPU memory during the forward pass for reuse during the backward pass. The backward pass processes these layers in reverse order, reducing the need to reload layers from the cache memory during backpropagation. The amount of layer retention is limited by available GPU memory and controlled by memory pressure.

Activation Management

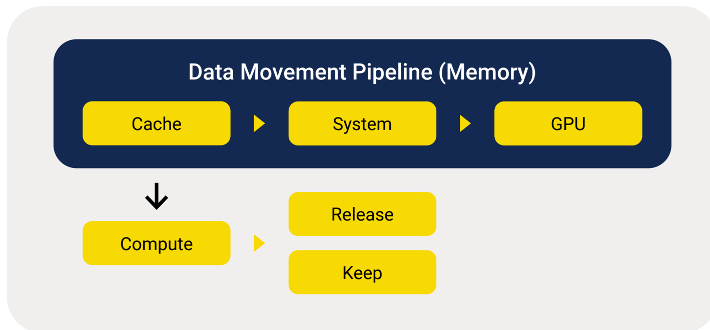
Activations are managed using a hybrid strategy of storage and recomputation with some activations stored for later use in a backward pass. Stored activations are written to the cache memory to reduce GPU memory usage and loaded from cache memory to system memory to GPU memory during the backward pass.

Other activations are recomputed on the GPU when needed and not stored. This approach is like activation checkpointing and lets the system balance memory usage and compute overhead by dynamically selecting between storage and recomputation.

Copyright ©2025 Phison Electronics | All Rights Reserved.

Moving Things Along

Pascari aiDAPTIV moves all model data through a structured movement path from cache memory to system memory then finally to GPU memory – the system memory serves as an intermediate buffer. Access latency is reduced by prefetching data into the system memory so it's ready to move to GPU memory when computation requires it. The data is retained for the backward pass or released after use.



Optimizer and Gradient Processing

Gradients are computed during the backward pass on the GPU while the optimizer step is performed on the CPU. The system uses optimizers such as AdamW, and requires model weights, gradients, optimizer states (momentum, variance, etc…) for this process.

The optimizer states are maintained outside of the GPU for more efficient GPU memory usage. CPU performance can significantly impact overall training time because of this.

Memory Management Strategy

aiDAPTIV treats GPU memory as a finite resource reserved solely for work. Only active layers and necessary data are kept in GPU memory with the cache memory acting as the primary storage layer for model data. System memory is used as a dynamic buffer to stage incoming data between GPU memory and cache memory.

Three major factors influence memory usage – batch size, sequence length, and number of layers loaded at once. These factors determine overall GPU memory pressure.

Performance Characteristics

Phison engineered aiDAPTIV for system memory-efficiency rather than compute-efficiency. The primary bottleneck is the cache memory bandwidth, but nothing can match the capacity-per-dollar of an SSD. GPU computation is often faster than the time it takes to fill the buffer (system memory) from the cache memory.
The CPU can potentially cause a secondary bottleneck during the optimizer step, too. When data is not yet available in GPU memory, the GPU execution may stall while waiting for the cache memory to transfer data to the system memory.

Overall performance depends on cache memory throughput, PCI Express bandwidth, CPU performance, and batch size configuration.

Key Optimizations

The aiDAPTIV Memory Management Middleware features key optimizations that allow better utilization of available hardware resources. Layer grouping reduces frequency of data transfers while prefetching into system memory reduces GPU idle time.

There's also activation checkpointing that reduces GPU memory usage. Triton kernel optimizations reduce system memory and GPU memory pressure, enabling larger batch sizes and improved memory efficiency, too.

System Tradeoffs

It's all about tradeoffs, however. Pascari aiDAPTIV enables large-scale model training by streaming model layers and training data between cache memory, system memory, and GPU memory. This allows larger model training on hardware that typically doesn't have the GPU memory capacity to do so.

This comes with some trade-offs, such as dependency on data movement speed and latency. High cache memory latency can increase training time, and the CPU-based optimizer is a potential bottleneck in certain hardware configurations.

Key Ideas

The tradeoff between memory savings and additional data movement overhead brings greater fine-tuning flexibility when working with limited budgets or hardware availability concerns. Nevertheless, Pascari aiDAPTIV lets systems run larger and more demanding AI workloads, locally.

Copyright ©2026 Phison Electronics | All Rights Reserved.