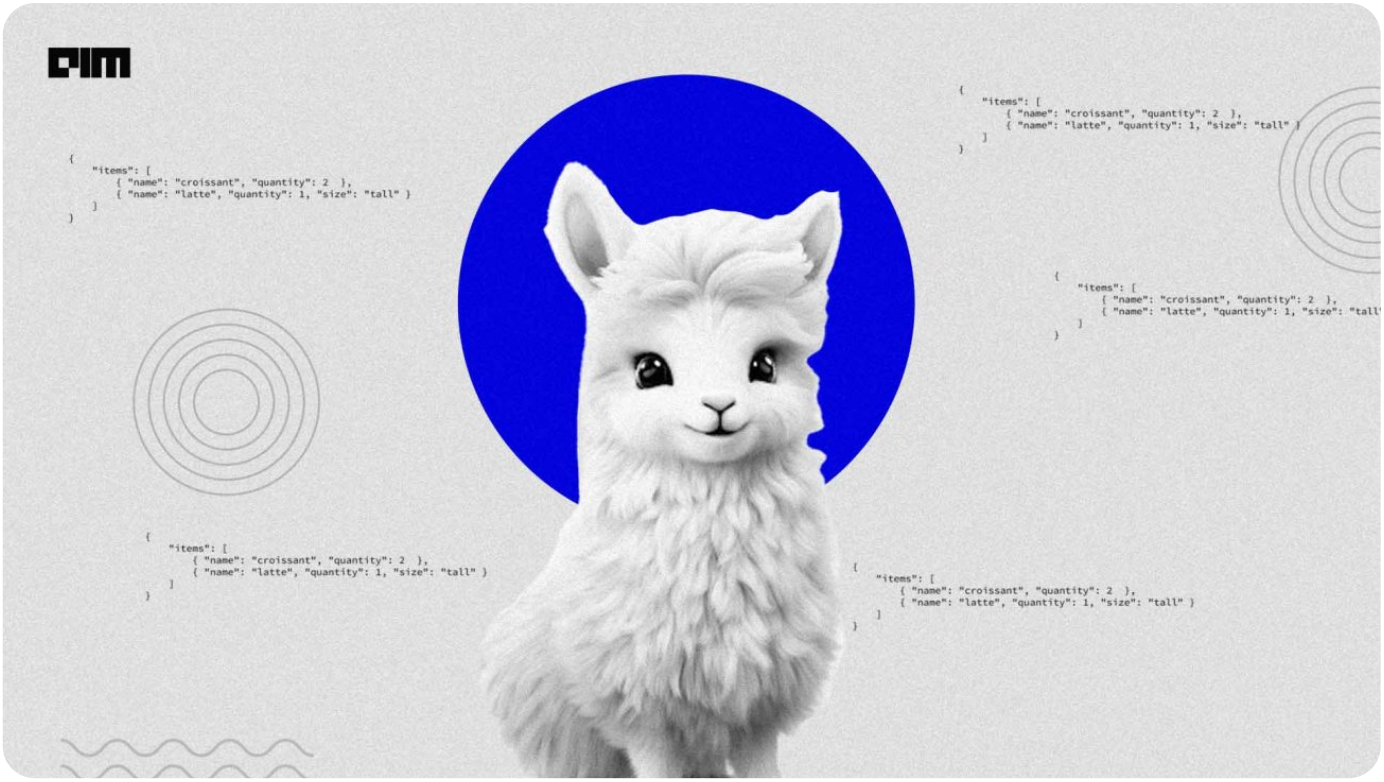
OpenAI's Karpathy Creates Baby Llama Instead of GPT-5

The primary focus of this endeavour was to demonstrate the feasibility of running Llama 2 models on low-powered devices using pure C code

Published on July 24, 2023 **by Mohit Pandey**



Listen to this story

The person who can easily build **GPT-5** over the weekend is surprisingly spending time testing out the capabilities of open source **Llama 2**. The quest for running LLMs on a single computer landed OpenAI's **Andrej Karpathy**, known for his contributions to the field of deep learning, to embark on a weekend project to create a simplified version of the Llama 2 model, and here it is!

For this, "I took nanoGPT, tuned it to implement the Llama 2 architecture instead of GPT-2, and the meat of it was writing the C inference engine in **run.c**," explained Karpathy in Llama2.c **GitHub repository**. His objective was to implement nanoGPT into Llama 2 architecture, instead of GPT within C programming language. The repository has already got 2.2K stars.

The success of Karpathy's **approach** lies in its ability to achieve highly interactive rates, even with reasonably sized models containing a few million parameters and trained on a 15 million parameter model of **TinyStories** dataset. He reports that on his M1 MacBook Air, the Llama 2 model with ~15 million parameters can infer at around 100 tokens per second in fp32, all through the C code he developed. This result is surprising as it demonstrates the feasibility of running complex models on resource-constrained devices with a straightforward implementation.

> *Once upon a time, there was a boy named Timmy. Timmy loved to play sports with his friends. He was very good at throwing and catching balls. One day, Timmy's mom gave him a new shirt to wear to a party. Timmy thought it was impressive and asked his mom to explain what a shirt could be for. "A shirt is like a special suit for a basketball game," his mom said. Timmy was happy to hear that and put on his new shirt. He felt like a soldier going to the army and shouting. From that day on, Timmy wore his new shirt every time he played sports with his friends at the party. Once upon a time, there was a little girl named Lily. She loved to play outside with her friends. One day, Lily and her friend Emma were playing with a ball. Emma threw the ball too hard and it hit Lily's face. Lily felt embarrassed and didn't want to play anymore. Emma asked Lily what was wrong, and Lily told her about her memory. Emma told Lily that she was embarrassed because she had thrown the ball too hard. Lily felt bad achieved tok/s: 98.746993347843922*

Sample Output

Furthermore, in a discussion on **HackerNews**, Karpathy explains how he was surprised that the compilation on MacBook Air M1 was much faster than anticipated with a speed of 100 tokens per second. Encouraged by this result, Karpathy has been actively updating the repository and also started testing on a 44 million parameter model, which is three times larger. Surprisingly, he was able to train 200k iterations with a batch size of 32 on 4 A100 GPUs in about eight hours.

"With this progress, it seems that achieving the 7B Llama model might be within grasp," said Karpathy. He has been known for several courses such as building **GPT from scratch**. People congratulated OpenAI for hiring Karpathy back from Tesla.

What is the Baby Llama approach?

Karpathy said that this approach was heavily inspired by **Georgi Gerganov**'s project – **llama.cpp**, which was almost the same project of using the first version of LLaMA on a MacBook using C and C++.

Karpathy's approach involves training the Llama 2 LLM architecture from scratch using PyTorch. After training, he saves the model weights in a raw binary file. The interesting part comes next: he writes a 500-line C file, named **'run.c'**, which loads the saved model and performs inferences using single-precision floating-point (fp32) calculations. This minimalistic approach ensures a low-memory footprint and requires no external libraries, allowing efficient execution on a single M1 laptop without the need for GPUs.

Karpathy also explores several techniques to improve the performance of the C code, including different compilation flags like -O3, -Ofast, -march=native, and more. These flags optimise the code by enabling vectorization, loop unrolling, and other hardware-specific tuning. By experimenting with these flags, users can achieve even faster inferences on their specific systems.

To try out the baby Llama 2 model on your own device, you can download the pre-trained model checkpoint from Karpathy's repository. The provided code will enable you to compile and **run the C code on your system**, offering a glimpse into the magic of running a deep learning model in a minimalistic environment.

It's crucial to note that Karpathy's project is a weekend experiment and not intended for production-grade deployment, which he acknowledges. The primary focus of this endeavour was to demonstrate the feasibility of running Llama 2 models on low-powered devices using pure C code, a language that for a long time has been not regarded useful for machine learning as it does not involve GPUs.

The Rise of Tiny LLMs

The biggest reason why models have been getting smaller all this while is to train and integrate them on smaller and local devices. Apart from not requiring a GPU, Karpathy's approach sets a precedent for what can be achieved on single devices. It is possible that through Meta's partnership, Microsoft will release a **bunch of tiny LLMs** based on Llama 2.

Along similar lines, Meta's release of Llama 2 also came with an astounding **partnership with Qualcomm**, a chip manufacturer. This partnership is to make Llama 2 run on local hardware. Apple also has a massive developer ecosystem, for which, the company recently released **Transformers architecture which is optimised for Apple Silicon**. Karpathy has already shown that a lot is possible.