nanollama: Train Your Own Llama 3 From Scratch on Custom Data

February 23, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** nanollama (v0.1.0, Feb 18 2026, 17 stars) is a Python framework forked from Karpathy's nanochat that pretrains Llama 3 architecture models from scratch on your own data. It exports to GGUF v3 for direct use with llama.cpp. Eight model configs from 46M to 7B parameters. The nano (46M) model trains in ~30 minutes on a single H100 for $3-5. Only sizes up to 336M (small) are verified — goldie (1.1B) is in training, larger sizes are untrained. Includes a unique personality injection system and a pure Go inference engine. For most practical work, fine-tuning existing models is far more cost-effective. This is for people who want to understand pretraining, experiment with data mixtures, or need a clean-room model with no licensing concerns.

📚 **Related:** Fine-Tuning with LoRA/QLoRA · VRAM Requirements · Model Formats Explained · LLM Quantization Explained · Planning Tool

Fine-tuning takes an existing model and adjusts it for your task. Pretraining starts from random weights and teaches the model language itself. Fine-tuning costs $5-20 and takes a couple hours. Pretraining costs hundreds to thousands of dollars and takes days.

So why would anyone pretrain from scratch?

Because you want to understand how LLMs actually work. Because you have proprietary data and need a clean-room model with zero licensing concerns from existing weights. Because you want to experiment with data mixtures, tokenizers, or architectures. Or because nanollama's personality injection system — which extracts a "personality vector" from training and transplants it into other models — requires from-scratch training and can't be replicated with fine-tuning.

nanollama (GitHub, v0.1.0, released Feb 18 2026) is a framework for pretraining Llama 3 architecture models from raw text. Forked from Karpathy's nanochat (43,800 stars), it adds the Llama 3 model definition (GQA, SwiGLU, RoPE, RMSNorm), GGUF v3 export, multilingual tokenizers, and a pure Go inference engine.

What You Get

Eight named model configurations, from toy to serious:

Model	Params	Layers	Tokens Needed	Training Time (H100)	Cost	Status
nano	46M	12	0.9B+	~30 min (1x H100)	$3-5	Verified
micro	87M	16	1.7B+	~1 hour (1x H100)	$6-10	Verified
mini	175M	20	3.5B+	~3 hours (4x H100)	$18-30	Verified
small	336M	24	6.7B+	10-24 hours (4x H100)	$60-200	Verified
goldie	1.1B	22	22B+	18-36 hours (4x H100)	$300-600	In progress
medium	1.6B	32	32B+	~48 hours (4x H100)	$1,000-1,600	Untrained
large	3.7B	36	74B+	~96 hours (8x H100)	Higher	Untrained
big	7.0B	38	140B+	~200 hours (8x H100)	Higher	Untrained

Only nano through small have verified training results. Goldie is currently training. Medium through big are defined configurations that have never been trained.

All sizes use the full Llama 3 architecture: grouped query attention, SwiGLU activation, RoPE, RMSNorm, and untied embeddings.

The Pipeline

Install

```
git clone https://github.com/ariannamethod/nanollama.git
cd nanollama
pip install .
```

Requires Python 3.10+, PyTorch 2.4+, SentencePiece.

Prepare Data

```
# Simple (nano/micro) — FineWeb-Edu only
python -m data.prepare_fineweb --samples 1000000   # ~1B tokens

# Multi-corpus (mini/small) — 4 sources
python -m data.prepare_multi_corpus --preset en_only --total-tokens 7B

# Multilingual (goldie+) — 6 sources, 4 languages
python -m data.prepare_multi_corpus --preset goldie --total-tokens 22B
```

Data gets tokenized into memory-mapped uint16 binary shards (~20MB each). The multi-corpus preset mixes FineWeb-Edu (55%), DCLM-Baseline (25%), The Stack v2 (10%), and MegaMath (10%).

Train

```
# Single GPU
python -m scripts.base_train --model-size nano

# Multi-GPU distributed
torchrun --nproc_per_node=4 -m scripts.base_train --model-size small
```

Export to GGUF

```
python -m scripts.export_gguf \
  --checkpoint checkpoints/nano/checkpoint.pt \
  --tokenizer weights/tokenizer.model \
  --output model.gguf --dtype f16
```

The exporter writes F32, F16, and Q8_0 natively. Post-export quantization to Q4_0, Q5_0, Q4_K, Q6_K works via `llama-quantize`.

Run with llama.cpp

```
llama-cli -m model.gguf -p "Once upon a time" -n 100
```

That's the full loop: raw text in, GGUF model out, running in llama.cpp.

Can You Train on Consumer Hardware?

The README documents H100 cloud GPUs exclusively. But we can estimate:

Model	Params	Single RTX 3090 (24GB)	Time Estimate
nano	46M	Fits easily	4-7 hours
micro	87M	Fits easily	6-12 hours
mini	175M	Fits with optimizer states	12-36 hours
small	336M	Tight — need gradient accumulation	Multiple days
goldie	1.1B	Probably doesn't fit (optimizer states + gradients ~13GB+)	Not feasible

An RTX 3090 delivers roughly 5-10x less throughput than an H100 for training. Nano and micro are absolutely feasible. Mini is doable if you're patient. Small is borderline — the model, optimizer states (8 bytes/param for Adam), and gradients push toward the 24GB limit. Goldie and above need multi-GPU or cloud.

For reference: MicroLlama trained a 300M model on 50B tokens using 4x RTX 4090s over 4 days. TinyLlama trained 1.1B on 3 trillion tokens using 16x A100s for 90 days.

The practical path for consumer hardware: Train nano or micro locally to learn the pipeline, then use Lambda Cloud H100 instances (~$2.50/hr) for anything bigger.

Personality Injection

This is nanollama's most distinctive feature and the one that genuinely requires pretraining from scratch.

The concept: train two models on the same data — one with personality data mixed in (20% of batches), one without. Subtract the weights:

```
gamma = theta_personality - theta_base
```

This gamma vector captures the personality without the language knowledge. You can then inject it into any compatible base model:

```
theta_new = theta_base_new + gamma
```

```
# Train base model
python -m scripts.base_train --model-size nano --model-tag base

# Train personality model (20% personality data)
python -m scripts.base_train --model-size nano --model-tag personality \
  --personality-dir data/personality/ --personality-ratio 0.2

# Extract gamma
python -m scripts.extract_gamma \
  --personality_ckpt checkpoints/personality/checkpoint.pt \
  --base_ckpt checkpoints/base/checkpoint.pt \
  --output gamma.npz
```

The gamma file is reportedly ~17MB — you can share a personality without sharing a full model. The authors claim the personality vector is orthogonal to language knowledge (cosine similarity ~0), meaning it transfers cleanly across base models of the same architecture.

The catch: Personality training doubles compute cost because you train the model twice.

The Go Inference Engine

nanollama includes a standalone inference binary in pure Go — zero external dependencies, ~9MB compiled.

```
cd go && go build -o nanollama .

# Interactive chat
./nanollama --model model.gguf --interactive
```

```
# With personality
./nanollama --model model.gguf --gamma gamma.npz

# Web UI with streaming
./nanollama --model model.gguf --serve --port 8080
```

Supports 7 quantization formats (F32, F16, Q4_0, Q5_0, Q8_0, Q4_K, Q6_K), GQA, RoPE, SwiGLU, and gamma injection at the embedding level. On CPU, nano achieves ~47 tok/s. Larger models produce single-digit tok/s without quantization.

How It Compares

Framework	Stars	Architecture	Pretraining	GGUF Export	Key Difference
nanollama	17	Llama 3	Yes	Yes (built-in)	Personality injection + Go engine
nanochat (Karpathy)	43,868	GPT-like	Yes	No	Full ChatGPT pipeline (SFT, RLHF)
nanoGPT (Karpathy)	53,620	GPT-2	Yes	No	Educational, deprecated for nanochat
LitGPT (Lightning)	13,173	Many	Yes	No	Production fine-tuning, 20+ architectures
torchtune (Meta)	5,687	Llama/ Gemma	Minimal	No	Post-training (SFT, DPO, KD)
torchtitan (PyTorch)	5,082	Llama 3	Yes	No	Multi-node distributed pretraining

nanollama is the only tool combining Llama 3 pretraining + built-in GGUF export + personality injection + a standalone inference engine, all in a small codebase (~400 lines for the model definition).

Quality Expectations: The Hard Truth

A model pretrained on 2.6B tokens will be dramatically worse than a fine-tuned version of an existing model at the same parameter count.

Approach	Model	Data	Result
nanollama nano from scratch	46M	2.6B tokens	"Grammatically correct but repetitive"
nanollama small from scratch	336M	2.6B tokens	"Reasonable factual grounding"
Fine-tuning Llama 3.2 1B with LoRA	1B	Meta's 15T tokens + your data	Orders of magnitude more capable

The fine-tuned 1B model costs $5-20 and takes 1-2 hours. The from-scratch 336M model costs $60-200 and takes 10-24 hours. And the fine-tuned model is vastly better because it inherits Meta's $100M+ of pretraining compute.

When pretraining from scratch makes sense:

- Learning how LLMs work (the real reason most people will use this)
- Research into architectures, optimizers, or data mixtures
- The personality injection system (cannot be replicated with fine-tuning)
- Clean-room models for specialized domains with no licensing concerns
- Multilingual models for underrepresented languages

When fine-tuning is better:

- Nearly every practical application

Honest Limitations

Five days old. 17 stars, 4 forks, one developer (Arianna Method). The commit history shows Claude Opus 4.6 as co-author on several commits.

Only verified through small (336M). Goldie (1.1B) is in progress. Sizes medium through big (1.6B-7B) are defined configs that have never been trained.

H100-only documentation. Consumer GPU support is undocumented. Activation checkpointing is not mentioned, which limits larger models on 24GB cards.

GPL-3.0 license. Everything you train with nanollama inherits GPL-3.0 licensing implications, unlike Apache 2.0 alternatives.

No fine-tuning. nanollama is pretraining-only. If you want to add instruction following after pretraining, you need a separate tool like torchtune.

Bottom Line

nanollama is a teaching tool and research framework, not a practical model factory. If you want a capable local model, download one and optionally fine-tune it.

But if you want to understand pretraining — how data becomes a model, how tokenizers shape what the model can learn, how training loss relates to output quality — nanollama is the most accessible Llama 3 pretraining framework available. The GGUF export means you can immediately run your model in llama.cpp and see firsthand what a 46M-parameter model trained on 1B tokens actually produces. It's humbling and educational.

The personality injection system is genuinely novel and worth watching. If it works reliably at scale, shipping 17MB personality vectors instead of multi-gigabyte models could change how we think about model customization.

GitHub: ariannamethod/nanollama
