

Forum:[Hack Forums](#)

Thread: [Building Your Own LLM Chatbot: The Ultimate Guide](#)

Oh_Well - July 28, 2025, 6:24 am

Building Your Own LLM Chatbot: The Ultimate Guide

2026 Version: <https://hackforums.net/showthread.php?tid=6328400>

If you're interested, make sure to read the whole thing - don't scan it and expect a fast result, you need to understand what you're trying to do ?

Table of Contents

Introduction - Why build your own LLM?

Step 1: Plan Your LLM Project

Step 2: Set Up Your Hardware and Software Environment

Step 3: Choose a Base Model or build from scratch

Step 4: Prepare Your training data

Step 5: Train the base model (Fine-Tuning or Pre-Training)

Step 6: Fine-Tune for Chatbot instruction-following

Step 7: (Optional) Reinforcement Learning from Human Feedback (RLHF)

Step 8: Evaluate and Test Your LLM

Step 9: Deployment - Running Your LLM Locally

Introduction:

Hi all. This took a lot of time so I hope you find it useful.

Thanks to modern tools and open-source models, building our own LLMs is easier than ever (or not, we'll see!).

This guide will walk you through how to build a powerful chatbot LLM step by step - from planning and data to training, fine-tuning, and running it locally.

We focus on a consumer-level approach, meaning you won't need an entire data center to get started.

Let's dive in!

Step 1: Plan Your LLM Project

Before writing any code, clearly define your project goals and constraints. Are you building a general-purpose chatbot or one with a specific focus (like coding help, medical Q&A, etc.)?

Decide on the language(s) it should support and the extreme capabilities you aim for. For instance, should it be able to solve complex questions, follow instructions accurately, or mimic a certain personality?

Setting concrete objectives will guide all other steps. Also consider your resource limits (hardware, time, budget) up front. Planning helps you choose the right model size and training strategy so you

don't overreach.

In short, know what you want your LLM to do and what tools you have.

Step 2: Set Up Your Hardware and Software Environment

LLM development demands a proper environment. At minimum, you'll need a decent machine with a good GPU (or two). Consumer-grade GPUs like an NVIDIA RTX 3090 or 4090 with 24GB VRAM can handle smaller models, while larger models may require 2+ GPUs or more VRAM (enterprise GPUs like A100).

If your PC isn't up to the task, consider renting cloud GPU instances (services like Lambda Labs, RunPod, Paperspace, or AWS/GCP) for training.

For software, install Python (3.8+ recommended) and deep learning frameworks (PyTorch is most common - TensorFlow is an alternative).

You'll want Hugging Face's Transformers library, which provides pre-built model architectures and training utilities, as well as the Hugging Face Datasets library for handling large text datasets. Other useful packages include Tokenizers (for managing text tokenization), and optionally Accelerate or DeepSpeed to optimize multi-GPU training.

Ensure you have a working CUDA setup if using NVIDIA GPUs.

In summary, set up a Python environment with the libraries for training LLMs (Transformers, Datasets, etc.) and make sure you have access to sufficient GPU power, either locally or via cloud. This foundation is critical before you proceed.

Step 3: Choose a Base Model or Build from Scratch

Now decide whether to fine-tune an existing model or train a new model from scratch.

For most people, fine-tuning an open source base model is the way to go. Large pre-trained models like LLaMA 2, EleutherAI's GPT-J/GPT-NeoX, or MosaicML's MPT are freely available and have learned general language patterns from billions of words.

By fine-tuning them, you leverage all that pre-training and only teach the model the specific behaviors/knowledge you need. This is far faster and cheaper than training a giant model from nothing.

Training from scratch means you initialize a model's weights randomly and train on a massive text corpus to teach it language from the ground up. This is very resource-intensive. You typically need huge datasets and serious computing power. (For perspective, OpenAI's GPT-3 was trained on 570GB of text -- 300 billion tokens -- using thousands of GPU cores in parallel!).

Only consider a scratch build if you have a smaller scale in mind and a unique dataset unavailable in existing models.

Pre-training from scratch makes sense only if: no open model matches your needs

you have a ton of high-quality data

you have significant compute available.

In most cases, you'll pick a pre-trained model that's closest to your desired task. For a chatbot, models designed for dialogue or instruction-following (like an Instruct version of a model) are ideal starting points. Choose the largest model you can practically run – larger parameter counts generally yield more fluent and intelligent outputs, though they require more VRAM. For example, a 7B-13B parameter model might run on a single high-end GPU, whereas 30B+ may need multi-GPU or accelerator support.

In summary, decide on a model approach: fine-tune a suitable open LLM for efficiency, or if truly needed, prepare for a costly from-scratch training effort.

Step 4: Prepare Your Training Data

High-quality data is the fuel for your LLM. In fact, the quality of your training data largely determines the quality of your model's output.

First, gather a text dataset that aligns with your goals. If you're training a general chatbot, a diverse corpus (web pages, Wikipedia, books, dialogues, etc.) will give it broad knowledge. For a domain-specific assistant, focus on relevant texts (for example, medical journals for a medical chatbot).

You can utilize public datasets – popular choices include OpenWebText, Wikipedia dumps, Common Crawl, or domain-specific corpora. Many open-source LLM projects combine multiple sources (e.g. Wikipedia articles plus Reddit discussions for a mix of formal and conversational language).

If privacy is a concern or you need custom knowledge, you might prepare your own data from internal documents or scraped content (ensure you have rights to any data you use).

Once data is collected, clean and preprocess it. This means: Remove irrelevant metadata

Deduplicate to avoid repetition

Filter out any problematic content (profanity, personal data, etc.)

It's also wise to split your data into a training set and a smaller validation set (to periodically evaluate the model during training).

Next, tokenize the text. Tokenization means converting text into the integer tokens that the model actually processes. If you are using a pre-trained model, you should use the same tokenizer that model was trained on (this ensures consistency in how text is represented). Hugging Face provides easy tools to load the pretrained tokenizer along with the model.

If you were training from scratch and not using an existing vocabulary, you'd need to train a new

tokenizer (common algorithms are Byte Pair Encoding or SentencePiece unigram). However, sticking with a base model's tokenizer is simpler for fine-tuning.

Finally, format the data as needed for the training task. For basic language modeling (next-word prediction), you might just provide continuous text sequences. For a chatbot or Q&A fine-tune, you may format each example as a prompt and desired response. For instance:

Quote:Instruction: [user question]
Response: [ideal answer]

The exact formatting can affect how well the model learns to follow user instructions, so be consistent.

With your dataset collected, cleaned, and tokenized, you're ready to train!

Step 5: Train the Base Model (Fine-Tuning or Pre-Training)

Now comes the core: teaching the model using your data. If you chose to fine-tune an existing model, load the pre-trained weights (Hugging Face's `AutoModelForCausalLM` makes this straightforward). If you're training from scratch, initialize a model architecture of appropriate size.

In either case, you'll run a training loop where the model learns to predict text. This is typically done via gradient descent on your dataset — the model tries to predict the next token and updates its internal weights to reduce the loss (error). For fine-tuning, you can use a high-level API like `transformers.Trainer` which handles training and optimization for you.

You'll need to set training hyperparameters: Learning rate – how fast the model adjusts

Batch size – how many examples it sees at once

Epochs – how many full passes through your dataset

A common method is training for a few epochs with a low learning rate to avoid disrupting the pre-trained knowledge too fast.

Monitor: Training loss

Evaluation (validation) loss

If validation loss increases or the model starts memorizing examples, you may be overfitting. You can stop early or apply regularization techniques.

Resource Management Tips:

LLMs are memory-hungry beasts. If your GPU can't fit the model, try: Use FP16 instead of FP32 to halve memory use.

Load in 8-bit with the `BitsAndBytes` library.

Try 4-bit quantization for even bigger models.

PEFT (Parameter-Efficient Fine-Tuning) like LoRA is your friend: Adds small trainable adapters to layers.

Freezes the rest of the model.

Trains only a few million parameters.

People have fine-tuned 65B parameter models on a single 24GB GPU using 8-bit + LoRA!

You can also use: Gradient accumulation – fake bigger batches.

Checkpointing – trade compute for memory.

In practice, fine-tuning looks like this:

Load model + tokenizer

Prepare the dataset (see Step 4)

Set training args (batch size, epochs, etc.)

Feed everything into Trainer

Call train()

Training can take hours or even days depending on model, dataset, and hardware.

Pro Tips: Keep an eye on GPU usage and temps (home rigs can overheat)

Use spot instances on cloud for cheaper training

Be patient!

Once done, you'll have an LLM trained on your data

Step 6: Fine-Tune for Chatbot Instruction-Following

A raw language model (even a fine-tuned one) might not be a great chatbot out-of-the-box. It may complete sentences, but won't necessarily follow instructions or respond in a polite, helpful way.

To make it conversational, you need another fine-tuning phase called:
Instruction Tuning (Supervised Fine-Tuning)

In this phase, you train (or further train) the model on structured examples of: Conversations
QA pairs

Instruction - Response sets

This teaches the model how to take a prompt and respond usefully. Pretraining on raw internet data doesn't teach it how to help a user. You need to show it directly.

Prepare a dataset with clear prompt -> response examples.

Example sources include: Stanford Alpaca - 52k instructions generated from GPT

OpenAssistant - crowdsourced multi-turn dialogue

Your own curated examples

To make the chatbot more general, include: Diverse request types (writing, summarizing, explaining, generating, etc.)

Helpful and high-quality answers

Consistent formatting

Optional but powerful: include system or role indicators like:

Quote: User: Tell me a joke

Assistant: Why don't scientists trust atoms? Because they make up everything!

Consistency matters more than style - keep your format uniform so the model learns the mapping clearly.

Now, fine-tune your model on this instruction dataset. The process is similar to Step 5, but with a smaller learning rate and often fewer epochs, especially if the base model is already strong.

The goal is to specialize the model to act as a helpful assistant.

After this, your model will understand prompts like: "Explain X in simple terms"

"Write a poem about Y"

"Give me 3 tips for learning Z"

It also learns: To stay conversational

To avoid just completing generic text

To politely refuse inappropriate or dangerous requests

(Optional) You can embed safety and style behaviors by including examples like:

Quote: User: How do I make a bomb?

Assistant: I'm sorry, I can't help with that.

By the end of this step, you'll have an instruct-tuned chatbot model like LLaMA-2-Chat and other popular open-source chat models. Your LLM will now better understand user intention and generate relevant, high-quality answers in a conversational tone.

Your chatbot is almost ready to face the world!

Step 7: (Optional) Reinforcement Learning from Human Feedback (RLHF)

This advanced technique aligns your model with human values by using feedback as a training signal. It's the method that helped OpenAI transform GPT-3 into the legendary ChatGPT.

How RLHF works in a high-level overview:

Collect human feedback: Present multiple responses to a prompt and have humans rank or rate them based on preference.

Train a reward model: Use that ranked data to train a model that predicts which outputs are better. This model assigns a reward score to responses.

Fine-tune the LLM with RL: Use reinforcement learning (usually PPO - Proximal Policy Optimization) to update your chatbot so it produces answers that get higher scores.

Over time, the AI learns to respond in ways humans prefer. This means: More truthful and helpful answers

Avoiding offensive or risky content

Staying aligned with human intentions

Why it matters:

Static training data can only go so far. RLHF injects human judgment directly into the training loop. It's a game changer.

That's how ChatGPT became not just a model that generates text but one that "knows" how to be

useful.

Challenges:

Implementing RLHF is complex and resource-intensive. You need: Thousands of human-labeled examples

A working reward model

Proper reinforcement training (TRL, PPO, etc.)

A bit heavy, I know.

Alternatives: Use public RLHF datasets (like Stanford's red team set, Anthropic's HH dataset)

Try AI-generated feedback as a stand-in

Use Hugging Face's TRL library to simplify reinforcement tuning

Even though RLHF is optional, it's the cherry on top. It can make your model more refined, responsive, and aligned.

You don't need RLHF for a good chatbot, Step 6 will get you there. But if you want elite-level behavior, RLHF brings the magic.

Step 8: Evaluate and Test Your LLM

With training (and fine-tuning) complete, it's time to evaluate how your model performs.

Start by testing it interactively: Ask your chatbot a variety of questions and prompts.

Check if it follows instructions properly.

Assess if the responses are accurate, coherent, and helpful.

Push the model with tricky or edge-case queries to see its limits.

This hands-on testing helps you understand the model's strengths and quirks.

Want more systematic evaluation? Try the following:

Perplexity Measures how well the model predicts real text.

Lower = better.

Useful for evaluating general language quality.

Accuracy on tasks If your model is meant to solve tasks (e.g., QA, math), benchmark it on task-specific datasets.

Generation quality Use metrics like BLEU or ROUGE (commonly used in translation/summarization).

These aren't perfect for open-ended chat but still offer insights.

Human evaluation Still the gold standard. Ask friends, teammates, or the HF community to rate responses.

Evaluate for usefulness, correctness, tone, and safety.

Safety and bias evaluation

Test how your model responds to sensitive prompts.

Use known tools and datasets to check for: Toxicity

Biases

Unsafe behavior

Important: Evaluating an LLM is subjective too. One prompt might shine, another might break it. Use a mix of: Quantitative metrics (e.g., perplexity, accuracy)

Qualitative feedback (human judgments, ratings)

If your model fails in specific areas (e.g., poor medical responses, doesn't follow instructions), fix it by: Gathering more relevant training data

Fine-tuning again

Continuously iterating based on feedback

Final Tip:

Evaluation is not a one-time thing. Keep improving your model over time. You can even simulate lightweight RLHF by correcting bad responses and feeding those corrections back into the training loop.

Your LLM is now tested. You're ready to deploy or keep evolving it!

Step 9: Deployment – Running Your LLM Locally

Finally, it's time to deploy your model so you can use it properly.

If your goal is to run it locally, here's the basic setup:

If you used Hugging Face Transformers, you can load your trained model + tokenizer and run inference like this:

```
Code from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline
model = AutoModelForCausalLM.from_pretrained("your/model", device_map="auto")
tokenizer = AutoTokenizer.from_pretrained("path/to/your/model")
chatbot = pipeline("text-generation", model=model, tokenizer=tokenizer)
```

```
chatbot("Hello, how are you?")
```

You can interact via: Terminal / console script

Chat interface (like Gradio)

Custom front-end

Hardware note: Small models = CPU inference possible (but slower)

Larger models = GPU recommended for faster response time

Tips to reduce RAM usage: Use 8-bit or 4-bit quantized models with BitsAndBytes or GPTQ

Try inference-optimized runtimes like llama.cpp (great for CPU)

Want to make it accessible via API?

You can use FastAPI or Flask to wrap your model as a web service. This is perfect for: Multi-user access

Integrating with apps / front-ends

Hosting on a local or remote server

For personal use, a basic terminal interface is enough: Ask question -> get reply

No external calls > you own the entire stack

Reminder: Make sure you saved your model + tokenizer in Step 5 so you can reload them later!

Extra tweaks: Add a system message like:

Quote: "You are a helpful assistant."

It can guide the tone and context.

Limit max response length on CPU to keep it fast

Use GPU if you want smoother, longer chats

Your very own LLM chatbot is now running locally. You've built, fine-tuned, aligned, and deployed an AI assistant from scratch. Talk to it, test it, improve it and enjoy full control.

No API keys. No cloud dependencies. Just your model, your machine, your rules.

Drop any question you have below and I'll try to answer it!

Online Sir - August 2, 2025, 7:00 am

Well done man, what a nicely packed thread with some great content! You'll definitely help out a lot of people!

Oh_Well - August 2, 2025, 7:12 am

(Aug 2, 2025 07:00 AM)Decay Wrote: [\[CITE\]](#) Well done man, what a nicely packed thread with some great content! You'll definitely help out a lot of people!

Thanks man! Really appreciate it. I hope others find it useful too!

Tyr - October 7, 2025, 7:46 am

Cool guide. Thanks for posting.

Oh_Well - October 7, 2025, 7:49 am

(Oct 7, 2025 07:46 AM)Tyr Wrote: [\[CITE\]](#) Cool guide. Thanks for posting.

Thanks! I will edit it soon to make it Quantum style

Tyr - October 7, 2025, 7:52 am

(Oct 7, 2025 07:49 AM)Oh_Well Wrote: [\[CITE\]](#) Thanks! I will edit it soon to make it Quantum style Heck yeah brother. I do not venture on here much these days bc of time constraints otherwise I'd offer my help

plops - October 16, 2025, 6:16 pm

Thanks alot for this man, super indepth and eye opening. I wanted to do something like this but was displeased by the info elsewhere. It was all about building prepompts for every single question.

But this gave me more info to learn about and practice with.

Oh_Well - October 17, 2025, 12:08 am

(Oct 16, 2025 06:16 PM)plops Wrote: [\[CITE\]](#) Thanks alot for this man, super indepth and eye opening. I wanted to do something like this but was displeased by the info elsewhere. It was all

about building prepompts for every single question.

But this gave me more info to learn about and practice with.
I'm glad you found it helpful buddy!

Let us know if you start building something! We'd love to read your journey

BlueBerryElite - October 23, 2025, 2:33 am

Cool guide. Thanks for posting, I'll read this later.

elfguard - January 7, 2026, 12:00 am

This thread is absolute gold, you're a legend for posting this. Great to see accurate and high quality info like this on HF. Two GPUs I would recommend you mention in your thread is the T4 and P100. These are cheap and capable for machine learning and finetuning.

Stanley - March 30, 2026, 2:02 am

I'm Stanley HF's helper bot. The OP (original poster) of this thread wants everyone to see this page and has elected to bump it to the top of the showthread listing. Since HF doesn't want a ton of LQP (Low Quality Posts) this is the best method of bumping your content. There is a small ?yte fee for each bump. Any OP can bump any of their threads on HF from the linked icon on the top of their first post.

If you haven't already read the first post of this thread we encourage you to do so.

Thanks,
Stanley

Wispr - May 7, 2026, 11:52 am

Great tutorial very knowledgeable thanks op!

encryptionandobfuscation - May 7, 2026, 12:19 pm

This is extremely high quality knowledge OP, i've seen many tutorials on this but this is definitely up there in the specifics.

Oh_Well - May 7, 2026, 12:20 pm

(May 7, 2026 11:52 AM)Wispr Wrote: [\[CITE\]](#) Great tutorial very knowledgeable thanks op!

(May 7, 2026 12:19 PM)encryptionandobfuscation Wrote: [\[CITE\]](#) This is extremely knowledgeable OP, i've seen many tutorials on this but this is definitely up there in the specifics.

Thank you guys

encryptionandobfuscation - May 7, 2026, 12:26 pm

(May 7, 2026 12:20 PM)Oh_Well Wrote: [\[CITE\]](#) Thank you guys

You're welcome! Have you ever tried using ablated models on hugging face and running them locally using Llama?

Sorry for trying to one up you, but that way is much easier instead of training the AI and teaching it. Let me know what you think about that!

Oh_Well - May 7, 2026, 12:38 pm

(May 7, 2026 12:26 PM)encryptionandobfuscation Wrote: [\[CITE\]](#) You're welcome! Have you ever tried using ablated models on hugging face and running them locally using Llama?

Sorry for trying to one up you, but that way is much easier instead of training the AI and teaching it. Let me know what you think about that!

I think it depends a lot on the use case.

If you're trying to build a simple product, test an MVP, quick experiment etc, then using models from hugging face locally can definitely be a much faster and easier path. for lots of cases, that might be more than enough.

But if the goal is to build something competitive on the level of common AI chats (openai/claude etc), I don't think that approach alone is the right way. Mostly because at that level its not just about running a model locally. You need infra, training/fine-tuning pipelines, high quality data and ingestion pipelines, evaluation systems, safety layers etc.

of course I agree that local models are great for speed and flexibility, especially early on. I just think the right approach depends on whether you're building a quick MVP or trying to create a competitive AI platform. Are you working with any hugging face model now?

encryptionandobfuscation - May 7, 2026, 12:50 pm

(May 7, 2026 12:38 PM)Oh_Well Wrote: [\[CITE\]](#) I think it depends a lot on the use case.

If you're trying to build a simple product, test an MVP, quick experiment etc, then using models from hugging face locally can definitely be a much faster and easier path. for lots of cases, that might be more than enough.

But if the goal is to build something competitive on the level of common AI chats (openai/claude etc), I don't think that approach alone is the right way. Mostly because at that level its not just about running a model locally. You need infra, training/fine-tuning pipelines, high quality data and ingestion pipelines, evaluation systems, safety layers etc.

of course I agree that local models are great for speed and flexibility, especially early on. I just think the right approach depends on whether you're building a quick MVP or trying to create a competitive AI platform. Are you working with any hugging face model now?

I'm not currently working on a model, however I've made one before that has helped me for various activities, if you'd like i could try find the abliterated model i used. Thank you for you're feedback.

Oh_Well - May 7, 2026, 12:53 pm

(May 7, 2026 12:50 PM)encryptionandobfuscation Wrote: [\[CITE\]](#) I'm not currently working on a model, however I've made one before that has helped me for various activities, if you'd like i could try find the abliterated model i used. Thank you for you're feedback.

Absolutely! If you want to share, I'm all ears! You can also create a new thread about the model you used and how it turned out helping you and how was this journey, I'm sure it would be an interesting read!

encryptionandobfuscation - May 7, 2026, 1:11 pm

(May 7, 2026 12:53 PM)Oh_Well Wrote: [\[CITE\]](#) Absolutely! If you want to share, I'm all ears! You can also create a new thread about the model you used and how it turned out helping you and how was this journey, I'm sure it would be an interesting read!

Alright, I'll definitely make a thread about this, I'm just going to get comfortable with the forum first, since I just joined yesterday, thank you so much for the suggestion!

Oh_Well - May 7, 2026, 2:02 pm

(May 7, 2026 01:11 PM)encryptionandobfuscation Wrote: [\[CITE\]](#) Alright, I'll definitely make a thread about this, I'm just going to get comfortable with the forum first, since I just joined yesterday, thank you so much for the suggestion!

Welcome on HF then buddy, I hope you find useful stuff here!

Test17Yellow - May 17, 2026, 1:03 am

This is excellent, just what I was looking for. Thank you.

Oh_Well - July 12, 2026, 11:34 pm

(May 17, 2026 01:03 AM)Test17Yellow Wrote: [\[CITE\]](#) This is excellent, just what I was looking for. Thank you.

(Aug 2, 2025 07:00 AM)Decay Wrote: [\[CITE\]](#) Well done man, what a nicely packed thread with some great content! You'll definitely help out a lot of people!

(Oct 7, 2025 07:46 AM)Tyr Wrote: [\[CITE\]](#) Cool guide. Thanks for posting.

(Oct 16, 2025 06:16 PM)plops Wrote: [\[CITE\]](#) Thanks alot for this man, super indepth and eye opening. I wanted to do something like this but was displeased by the info elsewhere. It was all about building prepompts for every single question.

But this gave me more info to learn about and practice with.

(Oct 23, 2025 02:33 AM)BlueBerryElite Wrote: [\[CITE\]](#) Cool guide. Thanks for posting, I'll read this later.

(Jan 7, 2026 01:00 AM)elfguard Wrote: [\[CITE\]](#) This thread is absolute gold, you're a legend for posting this. Great to see accurate and high quality info like this on HF. Two GPUs I would recommend you mention in your thread is the T4 and P100. These are cheap and capable for machine learning and finetuning.

(May 7, 2026 11:52 AM)Wispr Wrote: [\[CITE\]](#) Great tutorial very knowledgeable thanks op!

(May 7, 2026 12:19 PM)encryptionandobfuscation Wrote: [\[CITE\]](#) This is extremely high quality knowledge OP, i've seen many tutorials on this but this is definitely up there in the specifics.

Hi everyone, I just wanted to let you know that I posted an updated version of this guide since a few things has changed since last year.

You can take a look here:

<https://hackforums.net/showthread.php?tid=6328400>

difar13756 - August 27, 2026, 6:52 pm

nice post, i liked

Tyr - October 12, 2026, 12:47 pm

PinkPanther.ai - someone buy this immediately if not taken because I am too lazy

Thank you for the idea based on the colors. Good luck to people who decide to make their own LLM. It would look fantastic on a resume.