Your AI engineer starter plan

By Preston Kirschner

Before you spend a month on a course or a project, look at the work you want to get hired to do. This plan will help you choose what to learn, what to build, and what to show an employer.

Use the planning worksheet to compare roles, choose a learning priority, and outline your project.

Find the work closest to what you already do

In the reel, I talked about moving into AI engineering from software, data, or full-stack work. Start with that experience. The most useful next step depends on which parts of an AI system you want to build.

Open three current job descriptions that interest you. Choose roles at a level and in a location you could realistically apply for. Search both AI engineer and machine learning engineer, then read the responsibilities. The title alone will not tell you whether the work is building applications, developing models, or a mixture of both.

Write down the tasks each role asks someone to do. 'Connect a model to an existing product' is a task. 'Evaluate answers against a test set' is another. A list of tool names gives you less to work with. Keep the job links beside your notes so you can check the wording later.

If you build backend services, look for work involving APIs, data access, testing, and deployment. If you build full-stack products, look for chances to make an AI feature useful inside a real workflow. If you work with data, look closely at evaluation, data quality, and model behavior. These are starting points for your search, not rules about which jobs you can do.

For each responsibility, write one of three things: something you have already done, something you could practise in a small project, or something that needs more foundational study. The planning worksheet is a plain Markdown file you can copy and fill in. Your first learning task should come from a specific gap in those notes.

Learn enough to build the next part

For an application-focused project, start by calling a model from a short program. Send an input, read the response, and handle an error. Then ask for a structured result, such as JSON, and check that your program can use it. You should understand where the request is made, how credentials are supplied, and how you will notice a failed call.

Next, learn how to give the model information it does not already have in the conversation. Retrieval-augmented generation, usually called RAG, means finding relevant source material and supplying it when the model answers. A useful first exercise is to inspect the retrieved text yourself and decide whether it contains the answer.

LangChain can help connect retrieval and model calls. Use it when you can explain the steps you want it to handle. You do not need to begin by learning every abstraction in a framework.

PyTorch serves a different purpose. It is a place to learn about tensors, model building, training, and inference. If the roles you chose require that work, use its Learn the Basics tutorial and plan a model-focused project. Calling a hosted model through an API will not demonstrate that skill.

If Python is new to you, first get comfortable reading files, using functions, working with JSON, and debugging a short program. Work through one small program until you can explain it. Keep the pace tied to what you understand rather than a promise to change careers in a fixed number of weeks.

Build something you can use during your job search

One project you could build is a job-description comparison tool. You give it job descriptions and notes about projects you have actually completed. It extracts the responsibilities, shows the original wording, and helps you see which requirements you can already demonstrate. That gives the project a purpose while you are learning.

Start with a single pasted job description. Have the program return each responsibility alongside a supporting quotation. Check that the quotation appears in the original text and that the extracted task means the same thing. Once that works, compare several descriptions and group similar responsibilities. Keep the source wording available so a reader can inspect the grouping.

Then add your own project notes. For each responsibility, show a relevant example from those notes or leave the match empty. The tool should help you find gaps. It should not invent experience, rewrite a planned project as completed work, or pretend to know your chances of getting hired.

This project lets an application developer practise model APIs, structured output, validation, and a usable interface. You could add retrieval if your project notes grow large enough to need it. Choose additions because the tool needs them, and keep a record of the problem each addition solves.

A suitable task at work can be an even better choice because someone else can review whether it helps. Before using company material, agree with your manager on the task, the documents, and the time available. Pick one task with a clear input and output that you can finish before adding more features.

Keep examples of what goes wrong

Choose a small set of inputs and write down what a correct result should contain before using them to improve the tool. For the job-description project, check whether it misses a responsibility, merges two different tasks, or returns a quotation that does not support its summary.

Also try a case where your project notes do not support a requirement. If a posting asks for model training and your notes only describe calling an API, the match should stay empty. That is a useful test because a polished but invented match would make the tool less useful to you.

Save the input, the model's response, and your assessment together. Change one thing, then run the same cases again. Check the results that got worse as well as the ones that improved. Keep a few additional inputs aside until you are ready to show the project, so you can see how it handles material you did not use to choose your changes.

Write down the model configuration and record any reported usage, cost, and response time. Use the results you actually measured. A small test set can help you diagnose your project, but it does not establish how reliably the tool will handle every job description.

Your demonstration should show a useful result and a failure you can explain. A reviewer should be able to see the source text, understand the mistake, and follow the change you made. Put the exact starting commands and one remaining limitation in the README, then ask someone to try it from those instructions.

Use the project to make your next move

Return to the job descriptions you saved. For each responsibility, ask whether you now have something specific to show. You may have a working feature, a test you wrote, an error you diagnosed, or a deployment you can explain. If a requirement is still a gap, use it to choose the next learning task.

If you want experience on your current team, make a specific request. For example: 'We spend time comparing requirements across incoming requests. Could I try a small tool using a few approved examples? I would review the results with you before we decide whether to use it.' Replace that task with something your team actually does.

On your resume, describe what you built, what it does, and how you checked it. Include only the tools you used and results you measured. A personal project should be described as a personal project.

In an interview, talk through a decision. You might explain that the first version grouped two different responsibilities together, show the example that exposed it, and describe how you changed the output. That gives the interviewer a concrete piece of your work to discuss.

For your first session, save three job descriptions and fill in the first two sections of the worksheet. Finish with one responsibility you want to practise and one small task that would help you practise it. You can choose a course or a project with a much clearer purpose once you have those notes.

References and further reading

LinkedIn News: LinkedIn Jobs on the Rise 2026

The source for the career figures in the reel. The learning sequence, project, and worksheet in this guide are our recommendations.

LangChain: Retrieval

LangChain: Connecting a model

PyTorch: Learn the Basics