30 AI Projects You Can Build This Weekend¶

From beginner-friendly to advanced
By: Shaw Talebi

Building projects is the best way to develop your AI skills. However, for those new to the space, figuring out **what to build** can be the hardest part of the process. In this guide, I share 30 AI project ideas at 3 levels of sophistication. Each idea comes with step-by-step instructions and additional resources to help you go from 0 to 1.

Table of Contents¶

💡 **Bonus**: If you have a project idea (that's not on this list) and don't know where to start, ask this custom GPT!

Software 1.0: Analysis & Data Pipelines¶

While these are not strictly "AI" projects (as broadly known today), they serve as a helpful foundation for more advanced projects. For example, they cover tasks like web scraping, PDF parsing, and creating UIs, which are critical for many projects in this guide.

Project List:

- Financial Data Dashboard (Beginner)
- Resume Data Extraction (Beginner)
- US Market Research (Intermediate)
- Scraping Trends from Reddit (Intermediate)
- Weekly Report Emailer (Intermediate)

1) Financial Data Dashboard (Beginner)¶

Wrangling data and presenting it in an insightful way is (still) one of the best ways businesses can get value from their data. This first project gives you a flavor of this by pulling in real-world stock data and displaying it in an interactive dashboard using Streamlit.

Key Steps:

1. Use the yfinance library to pull historical stock or ETF data (e.g., closing price, volume).
2. Format the data using pandas (e.g., filter date ranges, compute rolling averages, etc.)
3. Plot time-series charts using plotly for metrics like price, volume, and moving averages.
4. Use streamlit to build a simple web interface where users can select tickers, time ranges, and metrics.
5. (Bonus) Add export functionality to download graphs or CSVs of selected data.

Resources:

- ChatGPT QuickStart

- Example Code

2) Resume Data Extraction (Beginner)¶

Much of the information we care about doesn't live in easy-to-access databases, but rather in reports, PDFs, and other documents. This has become even more widespread with the rise of LLMs and the endless use cases they enable. That said, it's important to understand how far you can go without LLMs using traditional string manipulation and regex. One use case for this is parsing key information from resumes.

Key Steps:

1. Use PyMuPDF to extract raw text from a sample resume PDF.
2. Normalize the extracted text (e.g., remove excessive whitespace, fix line breaks).
3. Use keyword-based rules or regex to locate and extract fields like: Name, Email, Education, and Skills.
4. Organize extracted fields into a structured format like a Python dictionary or JSON.
5. (Bonus) Apply to multiple resumes and save the results to a .csv file for downstream analysis.

Resources:

- ChatGPT QuickStart

3) US Market Research (Intermediate)¶

Whether you are giving context to an LLM or training a traditional ML model, it's often necessary to build data pipelines (i.e. move data from point A to point B). The "point A" much of the time is not data you have sitting on your computer, but rather data you must access via an API. For example, the US Census contains a wealth of information for businesses like market sizes, median incomes, and education levels. Here's a simple project to help you explore what's possible.

Key Steps:

1. Choose a practical use case (e.g., "Which US states have the highest median household income for a new product launch?").

2. Review US Census datasets and find one that answers your question (e.g., ACS 1-Year Estimates).
3. Get an API key.
4. Use the requests library to fetch data from the US Census API.
5. Convert the raw JSON response into a pandas DataFrame and filter for relevant values (e.g., top 10 states by income).
6. Use matplotlib to create a simple bar chart or map that summarizes your insights.

Resources:

- ChatGPT QuickStart
- Data Pipeline Explainer Video

4) Scraping Trends from Reddit (Intermediate)¶

While APIs make our lives easier, many times the data we need are not readily available (or sitting behind paywalls and rate limits). In these cases, you may need to scrape public webpages directly.

For example, Reddit is a goldmine for discovering what people are talking about, whether its hot topics in AI, emerging tools, or niche community insights. But manually scrolling through threads is tedious. This project walks through how you can automate this process with Python,

Note: it's important you respect a website's terms of use by reviewing their /robots.txt file.

Key Steps:

1. Pick a community to analyze (e.g. r/LocalLLaMA, r/technology) and note its URL.
2. Use the requests library to send a GET request to the subreddit's JSON endpoint (e.g. https://www.reddit.com/r/technology/top/.json?t=day).
3. Extract relevant fields like post titles, upvotes, and comment counts using the `json()` method and basic Python dictionaries.
4. Store the results in a pandas DataFrame for easy filtering, sorting, or export.
5. Write data to a .csv file for later analysis

6. (Bonus) Automate daily scraping with cron or GitHub Actions to build a time-series of trending content.

Resources:

- ChatGPT QuickStart

5) Weekly Report Emailer (Intermediate)¶

Email is the main way teams and businesses communicate. The ability for AI agents to plug into this communication ecosystem presents a potentially powerful user interface. Here's a simple project for automating a weekly email report.

Key Steps:

1. Use requests and BeautifulSoup to scrape job listings from a site like ai-jobs.net or similar.
2. Store job data (e.g., title, company, link) in a pandas DataFrame.
3. Perform any desired filtering or analysis (e.g. Top 5 AI Engineering roles)
4. Create a plain-text or HTML email body summarizing the key stats or insights
5. Use Python's `smtplib` and `email.mime` to send the email. Note: you will need to authenticate via Gmail app password or the like.
6. Schedule your script to run weekly using `cron` on Unix/macOS or Task Scheduler on Windows.

Resources:

- ChatGPT QuickStart
- AI Job Scraping Example
- Email Blast Example

Software 2.0: Machine Learning¶

While we can go very far with basic Python and it's most popular libraries, their are some tasks that are difficult to implement with Software 1.0 (if possible). In this section, we review a set of "traditional" machine learning projects which program computers with data.

Project List:

- Customer Churn Model (Intermediate)
- Review Sentiment Analysis (Intermediate)
- CC Fraud Detection (Intermediate)
- Stock Price Prediction (Intermediate)
- Clustering Reddit Posts (Intermediate)

6) Customer Churn Model (Intermediate)¶

Running a business with high churn is like sailing a ship with holes in it. A powerful way businesses can use ML to mitigate this is churn risk modeling. This example walks through how to train an ML model to do that.

Key Steps:

1. Download the Telco Customer Churn dataset from Kaggle and load it into a pandas DataFrame.
2. Data Preparation:
    - Convert categorical variables to numeric using one-hot encoding or label encoding.
    - Handle missing values.
    - Convert target variable Churn into binary labels (Yes → 1, No → 0).
3. Exploratory Data Analysis:
    - Run a `df.describe()` to view summary stats
    - Plot distributions of variables using `.hist()` method.
    - Evaluate associations between features and `tenure` variable
4. Create train-test split using train_test_split from sklearn.model_selection.

5. Train a classifier using RandomForestClassifier or LogisticRegression from sklearn.
6. Evaluate model on the train and test sets by computing metrics like accuracy, precision, recall, F1 score, and AUC-ROC using sklearn.metrics.
7. (Bonus) Use `.feature_importances_` (if using tree-based models) to identify key drivers of churn or linear coefficients directly.

Resources:

- ChatGPT QuickStart
- Random Forest Video Explainer
- Logistic Regression Video Explainer

7) Review Sentiment Analysis (Intermediate)¶

Before we has LLMs, predicting the sentiment of a piece of text (i.e. if it is positive or negative), required specialized models. This enables businesses to identify unhappy customer reviews to guide operational and product improvements. Here's one way to do this using traditional ML.

Key Steps:

1. Download Capterra Reviews dataset from Kaggle.
2. Preprocess the text (lowercasing, removing stopwords, punctuation).
3. Use TfidfVectorizer from sklearn to convert text into numerical features.
4. Create train-test split using train_test_split from sklearn.model_selection.
5. Train model to predict review rating using RandomForestRegressor or LinearRegression from sklearn.
6. Evaluate model on train and test sets using RMSE.
7. (Bonus) Use `.feature_importances_` (if using tree-based models) to identify key drivers of positive reviews or look at linear coefficients directly.

Resources:

- ChatGPT QuickStart
- Random Forest Video Explainer

- Linear Regression Explainer

8) CC Fraud Detection (Intermediate)¶

There are some business problems in which the thing you are trying to detect (or predict) is very rare. For example, credit card fraud is a relatively rare event, but an important one to identify. Since traditional classification methods aren't well suited for such class imbalanced, we can instead turn to outlier detection algorithms. Here's a simple project doing this.

Key Steps:

1. Import Credit Card Fraud dataset from Kaggle using pandas.
2. Exploratory Data Analysis:
   - Run a `df.describe()` to view summary stats for fraud and not fraud transactions
   - Plot distributions of variables using `.hist()` method.
3. Train an outlier detection model using Isolation Forest or OneClassSVM.
4. Convert model outputs to binary predictions and align with the fraud (label = 1) and normal (label = 0) convention.
5. Evaluate model performance using metrics like Precision, Recall, F1-score, and ROC-AUC from sklearn.
6. (Bonus) Create confusion matrix summarizing model performance.

Resources:

- ChatGPT QuickStart

9) Stock Price Prediction (Intermediate)¶

Time series (i.e. a set of values ordered by time) are ubiquitous in business. However, they have unique properties which make their prediction slightly different than the tabular data discussed above.

Key Steps:

1. Use the yfinance to download historical stock data for a ticker of your choice (e.g., AAPL). Include fields like Open, High, Low, Close, and Volume (OHLCV).

2. Feature engineering:
    - Past N days of OHLCV data
    - Moving Averages (e.g., 5-day, 10-day)
    - RSI (Relative Strength Index) using ta (or a custom function)
    - Create a new column for the next day's closing price (i.e., the target).
    - Drop rows with missing values
3. Split the dataset chronologically (e.g., 80% for training, 20% for testing) to preserve the temporal nature of the data.
4. Train regression a model using RandomForestRegressor or XGBoost.
5. Predict on the test set and compute performance metrics such as RMSE and plot predicted vs actual values to visualize how well your model tracks market trends using matplotlib.

Resources:

- ChatGPT QuickStart
- Time Series Explainer Video
- Smoothing Crypto Data Explainer Video

10) Clustering Reddit Posts (Intermediate)¶

There many business cases in which we don't need to make a prediction, but rather need to find patterns in our data e.g. customer segmentation, failure mode detection, market research, etc. This is where clustering approaches like KMeans are helpful. This projects walks through how to do this for analyzing trends on Reddit.

Key Steps:

1. Use the datasets library to load reddit_dataset_44 from Hugging Face. Filter or select specific subreddits if needed.
2. Clean the post content (e.g., remove links, lowercasing, remove stopwords) and drop any short or empty posts. Ignore comments.
3. Use TfidfVectorizer from sklearn to convert posts into numerical vectors:
4. Apply a clustering algorithm like KMeans to group similar posts.

5. For each cluster, find the top TF-IDF terms to understand what it's about.
6. Use PCA to reduce embedding dimensions and plot clusters with matplotlib.

Resources:

- ChatGPT QuickStart
- PCA Explainer Video

Software 3.0: Prompt Engineering¶

Large language models (LLMs) give us a fundamentally new way of building software. The main way we can do this is by giving them good prompts. In this section, we cover projects centered around prompt engineering.

Project List:

- YouTube Video Summarizer (Beginner)
- Upwork Profile Writer (Beginner)
- AI Project Idea Assistant (Intermediate)
- LLM-based Invoice OCR (Intermediate)
- Local PDF Chatbot (Intermediate)

11) YouTube Video Summarizer (Beginner)¶

Although I love adding technical talks to my YouTube "watch later" playlist, it might be a while before I watch them (if I ever get around to it). A project that can help with this is a tool that "watches" the videos for me and generates concise summaries with key points.

Key Steps:

1. Extract the YouTube video ID from the video link using regex.
2. Use the video ID to extract the transcript using youtube-transcript-api.
3. Write a prompt to summarize the video transcript.
4. Send the prompt to GPT-4o using OpenAI's Responses API.

Resources:

- ChatGPT QuickStart
- Research Paper Summarizer Example Code

12) Upwork Profile Writer (Beginner)¶

For freelancers on Upwork, good profile copy can be the difference between getting a new client every month vs every week. Most, however, don't know how to present themselves in a way that is compelling to busy prospects trying to

solve a specific problem in their business. Here I walk though how GPT-4.1 can help.

Key Steps:

1. Prepare background information about you, your services, and ideal customer.
2. Craft instructions for transforming the background information into high-converting profile copy.
3. Refine your prompt using ChatGPT.
4. Use OpenAI's Responses API to execute task programmatically using GPT-4.1.
5. (Bonus) Create Gradio UI where users can type in their background and used to generate profile copy.

Resources:

- ChatGPT QuickStart
- Example Instructions
- Gradio UI Example Code

13) AI Project Idea Assistant (Intermediate)¶

It's never been easier to create AI projects with Python. However, for those just getting started, figuring out how to implement an idea, which libaries to use, and if a project idea is commensurate to their current experience level can be a challenge. Here, I review how to build an AI assistant which helps users build any AI project idea they want!

Key Steps:

1. Transform this PDF guide into a markdown file.
2. Craft instructions that use this guide to write key project steps given an idea and other notes (e.g. tools, experience level).
3. Use OpenAI's Responses API to send a new project idea to GPT-4o-mini
4. Create chat UI with Gradio to capture project ideas and return instructions.

Resources:

- ChatGPT QuickStart
- Custom GPT Version
- Gradio Chat UI Example Code

14) LLM-based Invoice OCR (Intermediate)¶

Extracting data from invoice PDFs (especially poorly formatted or scanned ones) has traditionally required complicated rule-based systems or brittle OCR pipelines. Now, with powerful multimodal models like Llama 4 Maverick, you can convert these hard-to-parse documents into structured text with a simple API call. This project uses Together AI's hosted vision models to handle OCR and layout-aware extraction.

Key Steps:

1. Use pdf2image to convert multi-page invoice PDFs into high-resolution PNG or JPEG files.
2. Design a prompt that asks Llama 4 Maverick to return the extracted data in structured JSON format.
3. Refine your prompt in the Together AI Playground
4. Send each image to Together AI's using their Python API (Tip: use JSON response format)
5. Store structured data into a database or spreadsheet for later analysis.

Resources:

- ChatGPT QuickStart

15) Local PDF Chatbot (Intermediate)¶

Although ChatGPT is a great tool for quick document question-answering, this is not a suitable solution for documents containing sensitive information (data uploaded to ChatGPT can be used for future model training).One solution is to set up an LLM chat system on your local machine. Here is a simple way to do that with ollama.

Key Steps:

1. Download ollama

2. Extract the text from the PDF using PyMuPDF
3. Pull in your favorite LLM using ollama
4. Write a system message which passes the PDF text into the model's context
5. Pass user messages with questions about the document
6. (Bonus) Create a simple chat UI using Gradio

Resources:

- ChatGPT QuickStart

Software 3.0: RAG & Embeddings¶

Prompting LLMs ChatGPT-style only scratches the surface of what we can do with modern language models. We can, for example, add a layer of flexibility to our prompts using retrieval-augmented generation (RAG). Additionally, we can combine modern text embedding models with traditional ML approaches to unlock powerful (yet efficient) AI solutions.

Project List:

- System Prompt Routing (Beginner)
- Local RAG Chatbot (Intermediate)
- Review Sentiment Analysis (Intermediate)
- Resume-Job Matching (Intermediate)
- Multimodal Search (Advanced)

16) System Prompt Routing (Beginner)¶

When building multi-capability AI apps (like ones that can write code, summarize videos, or answer legal questions), you often need to change how you prompt the LLM depending on the user's query. Instead of manually switching prompts, a System Prompt Router does it for you. This project walks you through building a simple system that matches user queries to the best system prompt using semantic similarity.

Key Steps:

1. Create a prompt library via a Python dictionary where each entry has a short descriptive name and a matching system prompt.
2. Use an embedding model from sentence-transformers to embed each prompt's description.
3. When the user sends a query, embed it using the same model.
4. Compute the similarity between the query embedding with your prompt library embeddings and pick the system prompt with the highest similarity score.
5. Use the matched system prompt to craft a call to OpenAI's Responses API.

Resources:

- ChatGPT QuickStart
- Semantic Search Explainer Video

17) Local RAG Chatbot (Intermediate)¶

Sometimes you don't only want to find information in a single PDF, but information dispersed across a range of files. This is where retrieval-augmented generation (RAG) is helpful.

RAG automatically retrieves context relevant to a user query before passing it to an LLM. Here is a simple way to do that with LlamaIndex.

Key Steps:

1. Chunk source documents
2. Store chunks in vector database (with metatags)
3. Create a retriever to grab context from database given a query
4. Create a response synthesizer to generate response from a query and its context
5. Combine components into a query engine
6. (Bonus) Create a simple chat UI using Gradio

Resources:

- ChatGPT QuickStart
- RAG Example with LlamaIndex
- RAG Explainer Video

18) Review Sentiment Analysis (Intermediate)¶

In Project #7, we saw how we can predict the sentiment of customer reviews using a traditional TF-IDF vectorizer, based on unique word occurances. However, TF-IDF does not consider the context in which words occur, which limits their makes them a limited representation of underlying text.

Modern transformer-based embedding models, on the other hand, do not have this limitation. This project is a repeat of Project #7, but replaces the TF-IDF features with text embeddings.

Key Steps:

1. Download Capterra Reviews dataset from Kaggle
2. Preprocess the text (lowercasing, removing stopwords, punctuation).
3. Use an embedding model from sentence-transformers to embed each review.
4. Create train-test split using train_test_split from sklearn.model_selection.
5. Train model to predict review rating using RandomForestRegressor or LinearRegression from sklearn.
6. Evaluate model on train and test sets using RMSE.
7. (Bonus) Use `.feature_importances_` (if using tree-based models) to identify key drivers of positive reviews or look at linear coefficients directly.

Resources:

- ChatGPT QuickStart

19) Resume-Job Matching (Intermediate)¶

Matching job seekers with relevant opportunities is a time-consuming task that often relies on keyword matching or manual review. This project demonstrates how to use modern embedding models to create a more sophisticated matching system that considers semantic similarity between resumes and job descriptions.

Key Steps:

1. Read a collection of resumes and job descriptions into Python using PyMuPDF for PDFs and BeautifulSoup for HTML files.
2. Use OpenAI's text-embedding-3-small model to convert each resume and job description into embedding vectors.
3. For each resume, compute cosine similarity scores between it and each job description embedding using sklearn.
4. Return the top N job descriptions for each resume based on similarity score.
5. (Bonus) Generate a report for the best matching jobs for a given resume with tips on next steps.

Resources:

- ChatGPT QuickStart

20) Multimodal Search (Advanced)¶

Image search is traditionally challenging because it requires a set of tags and keywords for each image to match with user queries. With multimodal embeddings, however, we can avoid this step altogether and match a user query to a set of images based on their embedding representations. Here's a simple way to do this with CLIP.

Key Steps:

1. Chunk text from source documents and store in .csv file with metatags (e.g. title, page, etc.)
2. Extract images from source documents and store filepaths in .csv file with metatags (e.g. title, page, caption, etc.)
3. Compute text and image embeddings using CLIP, storing them alongside the original content.
4. Generate a search query, embed it with CLIP, and compare it against both text and image embeddings from your dataset using cosine similarity.
5. Combine and sort results based on similarity scores.
6. (Bonus) Create Streamlit search UI where users can type queries and select filters e.g. modality, source doc, or other metatags

Resources:

- ChatGPT QuickStart
- Multimodal Search Example Code
- Multimodal Embeddings Explainer Video

Software 3.0: AI Agents¶

While prompting LLMs and automatically giving them context allows them to generate helpful responses to a wide-range of tasks, we often don't only want LLMs to synthesize information for us, we want them to perform tasks on our behalf. This is the motivation for so-called agentic AI systems, which give LLMs access to tools to interact with the real-world.

Project List:

- Local MCP Server (Beginner)
- Web Search Agent (Beginner)
- Notion Agent (Intermediate)
- Agentic RAG (Intermediate)
- Looping Blog Writer (Advanced)

21) Local MCP Server (Beginner)¶

The power of agentic systems depends on the tools and context available to the LLM. As these systems scale, creating and maintaining these integrations can become cumbersome. This is where the model context protocol (MCP) can help.

MCP is a universal way to connect tools and context to LLM applications. Here is a quick way to augment Claude desktop with an MCP server which gives the model access to your local filesystem.

Key Steps:

1. Download Claude Desktop application
2. Configure filesystem MCP server by following the steps here (Note: you will need to specify which folders you want to give Claude access to.)
3. Restart Claude Desktop
4. Ask Claude questions about your files!

Resources:

- ChatGPT QuickStart
- MCP Explainer Video

22) Web Search Agent (Beginner)¶

While LLMs are great at synthesizing information, they can sometimes provide outdated or incorrect information due to their training data cutoff. One way to mitigate this limitation is to give them access to real-time web search capabilities. This project walks through how to build a simple agent that can search the web using OpenAI's Agents SDK.

Key Steps:

1. Use the Agent class to create a new agent and give it access OpenAI's WebSearchTool.
2. Send agent a request using the Runner class and .run() method
3. Display results
4. (Bonus) Create a simple chat UI using Gradio.

Resources:

- ChatGPT QuickStart
- Example Code

23) Notion Agent (Intermediate)¶

Notion has become my central hub for organzing daily tasks and storing information. This means it contains rich repository of context I can provide to an LLM to help me with things like brainstorming, project planning, strategy, and beyond. This project shows how to build an AI agent that can interact with Notion using natural language, making it easier to create pages, update databases, and retrieve content.

Key Steps:

1. Set up Notion MCP integrations by following the steps here
2. Set up MCP server using the MCPServerStdio class in OpenAI's Agents SDK.
3. Use the Agent class to create a new agent and give it access to the MCP server
4. Send the agent natural language requests

Resources:

- ChatGPT QuickStart
- Agents SDK: MCP Example

24) Agentic RAG (Intermediate)¶

In Project #17, we discussed how to build a simple RAG pipeline to retrieve context based on a user query. However, not all queries may require this additional retrieval step, leading to increased latency with no added value to the user. One way we can avoid this issue, is framing retrieval as a tool which the LLM can initiate whenever it deems fit. Here's how you can do that using OpenAI's Agents SDK.

Key Steps:

1. Create vector store of chunks and embeddings (like in project #17)
2. Write a retrieval function to perform a semantic search and return most relevant chunks for an input query.
3. Add @function_tool decorator to semantic search function
4. Use the Agent class to create a new agent and give it access to the retrieval tool.
5. Ask the agent questions about your source documents.

Resources:

- ChatGPT QuickStart

25) Looping Blog Writer (Advanced)¶

A powerful agentic design pattern is the evaluator-optimizer workflow. This allows an LLM to iteratively improve its responses based on feedback grounded in what makes a good output. Here's one way to implement such a system to write blog posts.

Key Steps:

1. Define a set of rule-based evals for what makes a "good" blog post
2. Write detailed instructions on how to write a blog post in your desired style
3. Generate an initial draft of the post (manually or using an LLM)

4. Apply evals to draft programmatically
5. If any evals fail, pass the results, draft, and instructions to an LLM to write a new one.
6. Repeat from step 4 until all evals pass.

Resources:

- ChatGPT QuickStart
- Looping Upwork Profile Writer Example

Software 3.0: Fine-tuning¶

Although LLMs can solve a wide range of problems out-of-the-box, there are situations where more model customization is required. This can be achieved through model fine-tuning i.e. adapting a model to a particular use case through additional training.

Project List:

- Chat with Yourself (Beginner)
- Upwork Profile Writer (Intermediate)
- Fine-tune Invoice Parser - OpenAI (Intermediate)
- Review Sentiment Classifier (Advanced)
- Fine-tune Invoice Parser - HF (Advanced)

26) Chat with Yourself (Beginner)¶

One challenge with using LLMs to generate content on your behalf is that writing instructions for capturing your unique voice can be a challenge. This is one sitation where fine-tuning can help. Here, is a simple project for teaching an LLM to respond to chat-like questions in your style.

Key Steps:

1. Use ChatGPT (or the like) to generate 100 unique conversational questions and organize them in a .csv file.
2. Open the .csv in Excel (or the like) and write responses to each in your voice.
3. Generate a system message to instruct the LLM to respond to questions in your voice (Tip: upload your .csv file to ChatGPT and have it summarize your style).
4. Organize data into (system message, conversational question, and your response) triplets.
5. Format data as a JSONL file based on what OpenAI's Fine-tuning API expects.
6. Upload JSONL file to OpenAI.
7. Run fine-tuning job using GPT-4o.

8. Vibe check the fine-tuned model by asking it personal questions.

Resources:

- ChatGPT QuickStart
- Fine-tuning Example Code
- Supervised Fine-tuning Guide

27) Upwork Profile Writer (Intermediate)¶

Although you can get LLMs to perform specific tasks well via detailed prompts with context and examples, this performance comes with greater compute (i.e. API) costs. Fine-tuning can help mitigate this cost by transfering it from test-time to train-time. In other words, we can get maintain long-prompt performance from a shorter one by fine-tuning an LLM to a specific use case. Here's how to do that for Project #12.

Key Steps:

1. Write detailed instructions with examples on how to write an Upwork profile (see #12)
2. Generate 100 (real or synthetic) inputs e.g. freelancer experience, services, and ideal customer
3. Use detailed prompt to generate 100 high-converting profiles
4. Write a new (short) system message for writing Upwork profiles
5. Organize data into (short system message, user input, and output) triplets
6. Format data as a JSONL file based on what OpenAI's Fine-tuning API expects
7. Upload JSONL file to OpenAI
8. Run fine-tuning job using GPT-4o
9. Send new input to with short system message to GPT-4o and the fine-tuned version
10. Vibe check performance differences
11. (Bonus) Try and take the efficency gains one step further by fine-tuning GPT-4o-mini with this same dataset

Resources:

- ChatGPT QuickStart
- Fine-tuning Example Code
- Supervised Fine-tuning Guide

28) Fine-tune Invoice Parser - OpenAI (Intermediate)¶

While LLMs can extract information from invoices using prompts (as seen in Project #14), fine-tuning can help improve accuracy and reduce costs by teaching the model to better understand invoice layouts and extract specific fields. This project walks through how to improve GPT-4.1's invoice parsing abilities for a specific domain.

Key Steps:

1. Gather 50 input-output pairs
    - Input: image version of an invoice (10 images max)
    - Output: JSON schema with desired fields (e.g. title, date, item name, quantity, price)
2. Organize data into (system message, text input, image input, and JSON output) quadruplets
3. Format data as a JSONL file based on what OpenAI's Fine-tuning API expects
4. Upload JSONL file to OpenAI
5. Run fine-tuning job using GPT-4.1
6. Send new input to with short system message to GPT-4.1 and the fine-tuned version
7. Vibe check performance differences

Resources:

- ChatGPT QuickStart
- Vision Fine-tuning Guide

29) Review Sentiment Classifier (Advanced)¶

While Projects #7 and #18 demonstrated how to build sentiment classifiers using traditional ML and modern embeddings, fine-tuning a transformer model can provide even better performance. This project shows how to fine-tune BERT for sentiment classification, which is a common approach in production systems.

Key Steps:

1. Download Capterra Reviews dataset from Kaggle
2. Create train-test-valid split
3. Import BERT base model (with classification head) and tokenizer using the transformers library.
4. Freeze base model parameters
5. Tokenize reviews and transform 5-point reveiw scale to binary one (e.g. keep only 1s and 5s, or split 4+ stars)
6. Define performance metrics e.g. accuracy, AUC
7. Define hyperparameters
8. Fine-tune model
9. Evaluate model on the test set

Resources:

- ChatGPT QuickStart
- Fine-tuning BERT Example

30) Fine-tune Invoice Parser - Hugging Face (Advanced)¶

In Project #28, we saw how to fine-tune GPT-4.1 to improve its invoice parsing performance in a specific context. While this was relatively easy to do with OpenAI's API, GPT-4.1 is a big (i.e. expensive model). To reduce costs, we can look toward smaller, open-source models which we can fine-tune to get comprable performance.

Key Steps:

1. Gather 500-1000 input-out pairs of invoice images and parsed data (see example)
2. Create train-test-valid split

3. Organize data into (system message, text input, image input, and JSON output) quadruplets
4. Load SmolVLM-Instruct model using the transformers library.
5. Evaluate base model performance with simple string matching
6. Create LoRA config using the peft library
7. Define hyperparameters for supervised fine-tuning using SFTConfig from trl.
8. Fine-tune the model using SFTTrainer from trl.
9. Evaluate fine-tuned model on the test set using the same eval metrics as step 5

Resources:

- ChatGPT QuickStart
- Vision Fine-tuning with HF

Libraries List¶

- BeautifulSoup - HTML parsing and web scraping library (#5, #19)
- datasets - Hugging Face's dataset loading and processing library (#10)
- gradio - Web UI framework for ML models and demos (#12, #13, #15, #17, #22)
- matplotlib - Data visualization library (#3, #9, #10)
- ollama - Local LLM deployment and management (#15)
- openai - OpenAI API client (#11, #12, #13, #16)
- pandas - Data manipulation and analysis library (#1, #3, #4, #5, #6, #8)
- peft - Parameter-Efficient Fine-Tuning library (#30)
- plotly - Interactive data visualization library (#1)
- PyMuPDF - PDF processing library (#2, #15, #19)
- pdf2image - PDF to image conversion library (#14)
- requests - HTTP library for Python (#3, #4, #5)
- scikit-learn - Machine learning library (#6, #7, #8, #9, #10, #18, #19)
- sentence-transformers - Text embedding models (#16, #18)

- streamlit - Web app framework for data science (#1, #20)
- ta - Technical analysis library (#9)
- together - Together AI API client (#14)
- transformers - Hugging Face's NLP library (#29, #30)
- trl - Transformer Reinforcement Learning library (#30)
- xgboost - Gradient boosting library (#9)
- yfinance - Yahoo Finance API client (#1, #9)
- youtube-transcript-api - YouTube transcript extraction library (#11)

AI Builders Cohort¶

Struggling to learn AI on your own? Check out my next AI Builders cohort, a 6-week program for technical consultants and entrepreneurs who want to build real-world AI solutions.

👉 **Learn more**