# Advanced Topic: Fine-Tuning and Using Custom Models

## 1. Concept Explanation

Fine-tuning is a powerful technique in machine learning, particularly with large language models (LLMs) and other deep learning models, where a pre-trained model is further trained on a smaller, task-specific dataset. This process adapts the general knowledge of the pre-trained model to a specific domain or task, significantly improving its performance for that particular use case without requiring the extensive computational resources and data needed to train a model from scratch. Custom models, in this context, refer to models that have been fine-tuned or specifically trained for a unique purpose, often leveraging proprietary data.

**Why Fine-Tune?**

* **Domain Adaptation:** Pre-trained models are trained on vast and diverse datasets, making them proficient in general tasks. However, they may lack specific knowledge or nuances of a particular industry or domain (e.g., medical, legal, financial jargon). Fine-tuning allows the model to learn these domain-specific patterns and terminology.
* **Task Specialization:** While a large LLM can perform many tasks (summarization, translation, question-answering), fine-tuning can optimize its performance for a very specific task, leading to higher accuracy and more relevant outputs.
* **Improved Performance with Less Data:** Fine-tuning requires significantly less data than training a model from scratch. The pre-trained model already has a strong foundation, and fine-tuning merely adjusts its weights to better suit the new data.
* **Cost-Effectiveness:** Training large models from scratch is computationally expensive. Fine-tuning offers a more economical way to achieve high performance for specialized tasks.
* **Reduced Latency (for smaller fine-tuned models):** Sometimes, fine-tuning can result in a smaller, more efficient model that performs better for a specific task, leading to faster inference times compared to using a very large general-purpose model.

**Types of Custom Models:**

* **Fine-tuned LLMs:** Pre-trained LLMs (like those from Azure OpenAI Service) adapted for specific tasks such as sentiment analysis on product reviews, generating specific types of creative content, or answering questions from a proprietary knowledge base.
* **Custom Vision Models:** Models trained using Azure AI Foundry Vision for specific image classification (e.g., identifying specific types of defects on a manufacturing line) or object detection (e.g., locating specific equipment in a factory).
* **Custom NLP Models:** Models trained using Azure AI Language service for custom text classification (e.g., categorizing customer feedback into specific issue types) or custom named entity recognition (e.g., extracting specific data points from financial documents).

## 2. Relevant Use Cases and Implementation Scenarios

Fine-tuning and using custom models are applicable across various industries and scenarios where general-purpose AI models fall short of specific business needs:

* **Healthcare:**
  + **Medical Text Analysis:** Fine-tune LLMs on medical literature and patient records to assist doctors in diagnosing rare diseases, summarizing patient histories, or extracting key information from clinical notes with high accuracy, understanding medical jargon and context.
  + **Medical Image Diagnosis:** Train custom computer vision models to detect specific anomalies in X-rays, MRIs, or pathology slides, aiding in early disease detection and personalized treatment planning.
* **Finance:**
  + **Fraud Detection:** Fine-tune models on transactional data to identify subtle patterns indicative of fraudulent activities, improving the accuracy of fraud detection systems.
  + **Financial Document Processing:** Create custom NLP models to extract specific financial entities (e.g., account numbers, transaction IDs, specific clauses) from contracts, invoices, and reports, automating data entry and compliance checks.
* **Manufacturing:**
  + **Quality Control:** Develop custom computer vision models to inspect products on an assembly line for specific defects (e.g., scratches, dents, misalignments) that are unique to a product line, ensuring consistent product quality.
  + **Predictive Maintenance:** Fine-tune models on sensor data from machinery to predict equipment failures more accurately, enabling proactive maintenance and reducing downtime.
* **Customer Service:**
  + **Personalized Chatbots:** Fine-tune LLMs to understand customer queries specific to a company’s products or services, providing more accurate and relevant responses than a general-purpose chatbot.
  + **Sentiment Analysis:** Train custom sentiment analysis models to accurately gauge customer sentiment from reviews or feedback, even when using industry-specific slang or nuanced expressions.
* **Legal:**
  + **Contract Review:** Fine-tune NLP models to identify and extract specific legal clauses, terms, and conditions from large volumes of legal documents, accelerating due diligence and compliance processes.
  + **Legal Research:** Create custom models that can summarize legal precedents or identify relevant case law based on specific legal questions.

## 3. Key Considerations and Best Practices

Effective fine-tuning and deployment of custom models require careful planning and adherence to best practices:

* **Data Quality and Quantity:** The success of fine-tuning heavily relies on the quality and relevance of the training data. While less data is needed than for training from scratch, it must be clean, accurately labeled, and representative of the target task and domain. Insufficient or biased data can lead to poor model performance.
* **Data Annotation:** For supervised fine-tuning, accurate and consistent data annotation is crucial. Invest in clear guidelines and quality control for human annotators.
* **Choosing the Right Base Model:** Select a pre-trained model that is already strong in a related general task. For example, for text generation, start with a powerful LLM. For image classification, start with a model pre-trained on a large image dataset.
* **Hyperparameter Tuning:** Fine-tuning involves adjusting hyperparameters like learning rate, batch size, and number of epochs. Experimentation and validation are necessary to find the optimal configuration for your specific dataset and task.
* **Evaluation Metrics:** Define clear and measurable evaluation metrics relevant to your task (e.g., accuracy, F1-score, BLEU score, ROUGE score). Continuously evaluate the fine-tuned model’s performance on a separate validation set to prevent overfitting.
* **Responsible AI:** Fine-tuning can inadvertently amplify biases present in the fine-tuning dataset. Implement strategies for bias detection and mitigation. Ensure transparency in how the custom model operates and its potential limitations. Consider privacy implications, especially when using sensitive proprietary data.
* **Version Control and Experiment Tracking:** Maintain strict version control for your datasets, code, and model checkpoints. Use experiment tracking tools (e.g., MLflow, Azure Machine Learning) to log hyperparameters, metrics, and model artifacts, enabling reproducibility and comparison of different fine-tuning runs.
* **Deployment and Monitoring:** Once fine-tuned, deploy your custom model to a scalable and reliable inference endpoint (e.g., Azure Machine Learning Endpoints, Azure Kubernetes Service). Implement continuous monitoring of model performance in production to detect drift, degradation, or unexpected behavior, and retrain as necessary.
* **Cost Management:** Be mindful of the computational costs associated with fine-tuning, especially for large models. Optimize resource utilization during training and inference.
* **Security:** Ensure that your fine-tuning data and models are stored and processed securely, adhering to compliance requirements and access controls.

## 4. Sample Implementations

To illustrate the practical application of fine-tuning and using custom models, we will provide sample implementations in both C# and Python. Our scenario is inspired by a real customer case where a company leverages fine-tuning to enhance the performance of an LLM for a specific business domain. We will consider a scenario where a legal firm uses fine-tuned LLMs to summarize legal documents, focusing on specific aspects relevant to their practice.

### Customer Case Inspiration: Legal Document Summarization

Legal firms deal with an immense volume of documents, from contracts and case files to legal precedents and regulations. Manually summarizing these documents is time-consuming and labor-intensive. While general-purpose LLMs can summarize text, they may not always capture the specific legal nuances or focus on the most critical information required by legal professionals. Fine-tuning an LLM on a dataset of legal documents with expert-generated summaries can significantly improve the relevance and accuracy of the summaries for legal use cases. This is a highly innovative application that can disrupt traditional legal workflows.

### Scenario: Fine-Tuned Legal Document Summarizer

Our sample implementation will demonstrate a fine-tuned LLM that specializes in summarizing legal documents. The goal is to produce concise summaries that highlight key legal arguments, parties involved, and relevant outcomes, as a legal professional would. We will simulate the use of a fine-tuned model, assuming the fine-tuning process itself has already occurred.

**Components:**

* **Azure OpenAI Service:** To host the fine-tuned large language model.

**Workflow:**

1. A legal document (represented as text) is provided.
2. The text is sent to the fine-tuned LLM deployed on Azure OpenAI Service.
3. The fine-tuned LLM generates a summary tailored to legal contexts.
4. The summary is returned.

This example will focus on interacting with a fine-tuned model endpoint.

### C# Implementation Example

This C# example demonstrates how to interact with a fine-tuned model deployed on Azure OpenAI Service for legal document summarization. It assumes you have a fine-tuned model (e.g., a fine-tuned gpt-3.5-turbo model) deployed and accessible via an Azure OpenAI endpoint.

First, ensure you have the necessary NuGet package installed:

<ItemGroup>  
 <PackageReference Include="Azure.AI.OpenAI" Version="1.0.0-beta.10" />  
</ItemGroup>

using Azure;  
using Azure.AI.OpenAI;  
using System;  
using System.Threading.Tasks;  
  
public class LegalSummarizer  
{  
 private readonly OpenAIClient \_client;  
 private readonly string \_deploymentName;  
  
 public LegalSummarizer(string endpoint, string apiKey, string deploymentName)  
 {  
 \_client = new OpenAIClient(new Uri(endpoint), new AzureKeyCredential(apiKey));  
 \_deploymentName = deploymentName;  
 }  
  
 public async Task<string> SummarizeLegalDocument(string documentText)  
 {  
 Console.WriteLine($"\nSummarizing legal document (first 200 chars): {documentText.Substring(0, Math.Min(documentText.Length, 200))}...");  
  
 try  
 {  
 // The prompt should be designed based on how your model was fine-tuned.  
 // For summarization, a common approach is to instruct the model to summarize the provided text.  
 string prompt = $"Summarize the following legal document, focusing on key arguments, parties, and outcomes:\n\n{documentText}\n\nSummary:";  
  
 ChatCompletionsOptions chatCompletionsOptions = new ChatCompletionsOptions()  
 {  
 Messages =  
 {  
 new ChatRequestUserMessage(prompt)  
 },  
 MaxTokens = 500, // Adjust based on desired summary length  
 Temperature = 0.7f,  
 NucleusSamplingFactor = 0.95f,  
 };  
  
 Response<ChatCompletions> response = await \_client.GetChatCompletionsAsync(  
 \_deploymentName, chatCompletionsOptions);  
  
 ChatChoice choice = response.Value.Choices[0];  
 return choice.Message.Content;  
 }  
 catch (RequestFailedException ex)  
 {  
 Console.WriteLine($"Error summarizing document: {ex.Message}");  
 return "Error: Could not summarize document.";  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine($"An unexpected error occurred: {ex.Message}");  
 return "Error: An unexpected error occurred.";  
 }  
 }  
  
 public static async Task Main(string[] args)  
 {  
 // Replace with your actual Azure OpenAI credentials and fine-tuned model deployment name  
 string endpoint = Environment.GetEnvironmentVariable("AZURE\_OPENAI\_ENDPOINT") ?? "YOUR\_AZURE\_OPENAI\_ENDPOINT";  
 string apiKey = Environment.GetEnvironmentVariable("AZURE\_OPENAI\_KEY") ?? "YOUR\_AZURE\_OPENAI\_KEY";  
 string deploymentName = Environment.GetVariable("AZURE\_OPENAI\_FINE\_TUNED\_DEPLOYMENT\_NAME") ?? "YOUR\_FINE\_TUNED\_MODEL\_DEPLOYMENT\_NAME";  
  
 if (new[] { endpoint, apiKey, deploymentName }.Any(string.IsNullOrWhiteSpace))  
 {  
 Console.WriteLine("Please set the environment variables for Azure OpenAI credentials and fine-tuned model deployment name.");  
 return;  
 }  
  
 LegalSummarizer summarizer = new LegalSummarizer(endpoint, apiKey, deploymentName);  
  
 string sampleLegalDocument1 = @"""  
 Case Name: Smith v. Jones  
 Court: District Court of Anytown  
 Date: July 8, 2025  
  
 Facts: Plaintiff, John Smith, filed a lawsuit against Defendant, Sarah Jones, alleging breach of contract. The contract, signed on January 1, 2024, stipulated that Jones would deliver 100 widgets to Smith by March 1, 2024. Jones failed to deliver the widgets by the agreed-upon date.  
  
 Arguments: Smith argued that Jones's failure to deliver constituted a material breach, causing him significant financial losses. Jones contended that unforeseen supply chain disruptions made timely delivery impossible and invoked a force majeure clause in the contract.  
  
 Outcome: The court found in favor of the Plaintiff, John Smith, ruling that the force majeure clause did not apply to the specific circumstances of the supply chain disruption. Damages were awarded to Smith in the amount of $10,000 for lost profits.  
 """;  
  
 string summary1 = await summarizer.SummarizeLegalDocument(sampleLegalDocument1);  
 Console.WriteLine($"\nSummary 1:\n{summary1}");  
  
 string sampleLegalDocument2 = @"""  
 Agreement Type: Non-Disclosure Agreement (NDA)  
 Parties: InnovateTech Inc. (Disclosing Party) and Global Solutions LLC (Receiving Party)  
 Effective Date: June 1, 2025  
  
 Purpose: To protect confidential information exchanged during discussions regarding a potential business collaboration. Confidential Information includes, but is not limited to, trade secrets, business plans, financial data, and technical specifications.  
  
 Obligations: The Receiving Party agrees to maintain the confidentiality of the Disclosing Party's confidential information and not to disclose it to any third party for a period of five (5) years from the Effective Date. The Receiving Party shall use the confidential information solely for the purpose of evaluating the potential business collaboration.  
  
 Governing Law: This Agreement shall be governed by and construed in accordance with the laws of the State of Delaware.  
 """;  
  
 string summary2 = await summarizer.SummarizeLegalDocument(sampleLegalDocument2);  
 Console.WriteLine($"\nSummary 2:\n{summary2}");  
 }  
}

**Explanation:**

1. **Initialization:** The LegalSummarizer class is initialized with your Azure OpenAI endpoint, API key, and the deployment name of your fine-tuned model. It creates an OpenAIClient to interact with the service.
2. **SummarizeLegalDocument Method:**
   * **Prompt Construction:** A prompt is constructed to instruct the fine-tuned model to summarize the provided legal document, specifically asking it to focus on key arguments, parties, and outcomes. The effectiveness of this prompt depends heavily on how your model was fine-tuned.
   * **Chat Completions Options:** ChatCompletionsOptions are set, including the user message (the prompt), MaxTokens for the summary length, and Temperature for creativity. For summarization, a lower temperature is often preferred for more factual and less imaginative output.
   * **API Call:** The GetChatCompletionsAsync method is called with the deployment name and options to send the request to your fine-tuned model.
   * **Result Retrieval:** The content of the first choice from the model’s response is returned as the summary.
   * **Error Handling:** Basic error handling is included for RequestFailedException (Azure-specific errors) and general exceptions.
3. **Main Method:** This is the entry point for the console application. It retrieves credentials and the deployment name from environment variables. It then defines two sample legal documents and calls SummarizeLegalDocument for each, printing the generated summaries.

**Before running this code:**

* **Azure OpenAI Service:** You need an Azure OpenAI service instance. It is assumed you have fine-tuned a model (e.g., a gpt-3.5-turbo base model) on your legal document summarization dataset and deployed it to a specific deployment name.
* **Fine-tuning Data:** The quality of the summaries will directly depend on the quality and relevance of the data used for fine-tuning. This data should consist of pairs of legal documents and their corresponding expert-generated summaries, focusing on the desired aspects (arguments, parties, outcomes).
* **Environment Variables:** Set the following environment variables with your actual credentials and fine-tuned model deployment name:
  + AZURE\_OPENAI\_ENDPOINT
  + AZURE\_OPENAI\_KEY
  + AZURE\_OPENAI\_FINE\_TUNED\_DEPLOYMENT\_NAME

This C# example provides a solid foundation for building sophisticated RAG-based AI agents that can leverage your proprietary data for accurate and context-aware responses.

### Python Implementation Example

This Python example mirrors the C# implementation, demonstrating how to interact with a fine-tuned model deployed on Azure OpenAI Service for legal document summarization. Ensure you have the necessary Python package installed:

pip install openai

import os  
from openai import AzureOpenAI  
  
class LegalSummarizer:  
 def \_\_init\_\_(self):  
 self.client = AzureOpenAI(  
 azure\_endpoint=os.environ.get("AZURE\_OPENAI\_ENDPOINT"),  
 api\_key=os.environ.get("AZURE\_OPENAI\_KEY"),  
 api\_version="2024-02-01", # Use the appropriate API version  
 )  
 self.deployment\_name = os.environ.get("AZURE\_OPENAI\_FINE\_TUNED\_DEPLOYMENT\_NAME")  
  
 def summarize\_legal\_document(self, document\_text: str) -> str:  
 print(f"\nSummarizing legal document (first 200 chars): {document\_text[:200]}...")  
  
 try:  
 # The prompt should be designed based on how your model was fine-tuned.  
 # For summarization, a common approach is to instruct the model to summarize the provided text.  
 prompt = (  
 f"Summarize the following legal document, focusing on key arguments, parties, and outcomes:\n\n"  
 f"{document\_text}\n\n"  
 "Summary:"  
 )  
  
 response = self.client.chat.completions.create(  
 model=self.deployment\_name,  
 messages=[  
 {"role": "user", "content": prompt}  
 ],  
 max\_tokens=500, # Adjust based on desired summary length  
 temperature=0.7,  
 n=1,  
 )  
  
 return response.choices[0].message.content  
 except Exception as ex:  
 print(f"Error summarizing document: {ex}")  
 return "Error: Could not summarize document."  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Replace with your actual Azure OpenAI credentials and fine-tuned model deployment name  
 # It's recommended to set these as environment variables  
 required\_env\_vars = [  
 "AZURE\_OPENAI\_ENDPOINT",  
 "AZURE\_OPENAI\_KEY",  
 "AZURE\_OPENAI\_FINE\_TUNED\_DEPLOYMENT\_NAME",  
 ]  
  
 for var in required\_env\_vars:  
 if not os.environ.get(var):  
 print(f"Please set the environment variable: {var}")  
 exit(1)  
  
 summarizer = LegalSummarizer()  
  
 sample\_legal\_document1 = """  
Case Name: Smith v. Jones  
Court: District Court of Anytown  
Date: July 8, 2025  
  
Facts: Plaintiff, John Smith, filed a lawsuit against Defendant, Sarah Jones, alleging breach of contract. The contract, signed on January 1, 2024, stipulated that Jones would deliver 100 widgets to Smith by March 1, 2024. Jones failed to deliver the widgets by the agreed-upon date.  
  
Arguments: Smith argued that Jones's failure to deliver constituted a material breach, causing him significant financial losses. Jones contended that unforeseen supply chain disruptions made timely delivery impossible and invoked a force majeure clause in the contract.  
  
Outcome: The court found in favor of the Plaintiff, John Smith, ruling that the force majeure clause did not apply to the specific circumstances of the supply chain disruption. Damages were awarded to Smith in the amount of $10,000 for lost profits.  
"""  
  
 summary1 = summarizer.summarize\_legal\_document(sample\_legal\_document1)  
 print(f"\nSummary 1:\n{summary1}")  
  
 sample\_legal\_document2 = """  
Agreement Type: Non-Disclosure Agreement (NDA)  
Parties: InnovateTech Inc. (Disclosing Party) and Global Solutions LLC (Receiving Party)  
Effective Date: June 1, 2025  
  
Purpose: To protect confidential information exchanged during discussions regarding a potential business collaboration. Confidential Information includes, but is not limited to, trade secrets, business plans, financial data, and technical specifications.  
  
Obligations: The Receiving Party agrees to maintain the confidentiality of the Disclosing Party's confidential information and not to disclose it to any third party for a period of five (5) years from the Effective Date. The Receiving Party shall use the confidential information solely for the purpose of evaluating the potential business collaboration.  
  
Governing Law: This Agreement shall be governed by and construed in accordance with the laws of the State of Delaware.  
"""  
  
 summary2 = summarizer.summarize\_legal\_document(sample\_legal\_document2)  
 print(f"\nSummary 2:\n{summary2}")

**Explanation:**

1. **Initialization:** The LegalSummarizer class initializes the AzureOpenAI client with your Azure OpenAI endpoint and API key. It also retrieves the deployment name of your fine-tuned model from environment variables.
2. **summarize\_legal\_document Method:**
   * **Prompt Construction:** A prompt is constructed to guide the fine-tuned model in summarizing the legal document, emphasizing key arguments, parties, and outcomes. The effectiveness of this prompt is highly dependent on the fine-tuning data and process.
   * **API Call:** The chat.completions.create method is used to send the request to your fine-tuned model deployed on Azure OpenAI. Parameters like max\_tokens (for summary length) and temperature (for creativity) are set.
   * **Result Retrieval:** The content of the first message from the model’s response is returned as the summary.
   * **Error Handling:** A general exception handler is included to catch and report any errors during the summarization process.
3. **Main Execution Block:** This block checks for the presence of required environment variables. It then creates a LegalSummarizer instance and defines two sample legal documents. Finally, it calls summarize\_legal\_document for each sample and prints the generated summaries.

**Before running this code:**

* **Azure OpenAI Service:** You need an Azure OpenAI service instance. It is assumed you have fine-tuned a model (e.g., a gpt-3.5-turbo base model) on your legal document summarization dataset and deployed it to a specific deployment name.
* **Fine-tuning Data:** The quality of the summaries will directly depend on the quality and relevance of the data used for fine-tuning. This data should consist of pairs of legal documents and their corresponding expert-generated summaries, focusing on the desired aspects (arguments, parties, outcomes).
* **Environment Variables:** Set the following environment variables with your actual credentials and fine-tuned model deployment name:
  + AZURE\_OPENAI\_ENDPOINT
  + AZURE\_OPENAI\_KEY
  + AZURE\_OPENAI\_FINE\_TUNED\_DEPLOYMENT\_NAME

This Python example provides a clear and functional demonstration of how to leverage a fine-tuned LLM on Azure OpenAI for a specialized task like legal document summarization, showcasing the power of custom models in domain-specific applications.

### How to Test the C# Implementation Example

This section provides step-by-step instructions on how to set up, run, and verify the C# code example for the Legal Summarizer.

#### Prerequisites

* .NET SDK (version 6.0 or higher) installed.
* An Azure OpenAI Service instance with a fine-tuned model deployed (e.g., gpt-3.5-turbo fine-tuned for legal summarization). You will need the endpoint, API key, and the deployment name of your fine-tuned model.

#### Setup

1. **Create a new C# Console Project:**

* dotnet new console -n LegalSummarizerApp  
  cd LegalSummarizerApp

1. **Add NuGet Packages:** Open the LegalSummarizerApp.csproj file and add the following ItemGroup:

* <ItemGroup>  
   <PackageReference Include="Azure.AI.OpenAI" Version="1.0.0-beta.10" />  
  </ItemGroup>
* Then, restore the packages:
* dotnet restore

1. **Create Program.cs:** Replace the content of Program.cs with the C# code provided in this document (specifically the C# example). Ensure the using statements and class definitions are correct.

#### Configuration

Set the following environment variables with your Azure OpenAI credentials and fine-tuned model deployment name. Replace the placeholder values with your actual information.

* AZURE\_OPENAI\_ENDPOINT="YOUR\_AZURE\_OPENAI\_ENDPOINT"
* AZURE\_OPENAI\_KEY="YOUR\_AZURE\_OPENAI\_KEY"
* AZURE\_OPENAI\_FINE\_TUNED\_DEPLOYMENT\_NAME="YOUR\_FINE\_TUNED\_MODEL\_DEPLOYMENT\_NAME"

**Example (for PowerShell on Windows):**

$env:AZURE\_OPENAI\_ENDPOINT="https://your-openai-resource.openai.azure.com/"  
$env:AZURE\_OPENAI\_KEY="your-openai-api-key"  
$env:AZURE\_OPENAI\_FINE\_TUNED\_DEPLOYMENT\_NAME="your-fine-tuned-deployment-name"

**Example (for Bash/Zsh on Linux/macOS):**

export AZURE\_OPENAI\_ENDPOINT="https://your-openai-resource.openai.azure.com/"  
export AZURE\_OPENAI\_KEY="your-openai-api-key"  
export AZURE\_OPENAI\_FINE\_TUNED\_DEPLOYMENT\_NAME="your-fine-tuned-deployment-name"

#### Execution

Run the application from the LegalSummarizerApp directory:

dotnet run

#### Expected Results

The application will print the original legal documents (truncated) and then the summaries generated by your fine-tuned Azure OpenAI model. The summaries should be concise and focus on key arguments, parties, and outcomes, reflecting the fine-tuning for legal contexts.

**Example Output Structure (content will vary based on your model and fine-tuning):**

Summarizing legal document (first 200 chars): Case Name: Smith v. Jones  
Court: District Court of Anytown  
Date: July 8, 2025  
  
Facts: Plaintiff, John Smith, filed a lawsuit against Defendant, Sarah Jones, alleging breach of contract. The contract, signed on January 1, 2024, stipulated that Jones would deliver 100 widgets to Smith by March 1, 2024. Jones failed to deliver the widgets by the agreed-upon date....  
  
Summary 1:  
[Summary of Smith v. Jones focusing on breach of contract, parties, and court ruling]  
  
Summarizing legal document (first 200 chars): Agreement Type: Non-Disclosure Agreement (NDA)  
Parties: InnovateTech Inc. (Disclosing Party) and Global Solutions LLC (Receiving Party)  
Effective Date: June 1, 2025  
  
Purpose: To protect confidential information exchanged during discussions regarding a potential business collaboration. Confidential Information includes, but is not limited to, trade secrets, business plans, financial data, and technical specifications....  
  
Summary 2:  
[Summary of NDA focusing on purpose, obligations, and governing law]

### How to Test the Python Implementation Example

This section provides step-by-step instructions on how to set up, run, and verify the Python code example for the Legal Summarizer.

#### Prerequisites

* Python 3.8 or higher installed.
* An Azure OpenAI Service instance with a fine-tuned model deployed (e.g., gpt-3.5-turbo fine-tuned for legal summarization). You will need the endpoint, API key, and the deployment name of your fine-tuned model.

#### Setup

1. **Create a new Python file:** Create a file named legal\_summarizer.py.
2. **Install required packages:**

* pip install openai

1. **Add the Python code:** Copy the Python code provided in this document (specifically the Python example) into legal\_summarizer.py.

#### Configuration

Set the following environment variables with your Azure OpenAI credentials and fine-tuned model deployment name. Replace the placeholder values with your actual information.

* AZURE\_OPENAI\_ENDPOINT="YOUR\_AZURE\_OPENAI\_ENDPOINT"
* AZURE\_OPENAI\_KEY="YOUR\_AZURE\_OPENAI\_KEY"
* AZURE\_OPENAI\_FINE\_TUNED\_DEPLOYMENT\_NAME="YOUR\_FINE\_TUNED\_MODEL\_DEPLOYMENT\_NAME"

**Example (for Bash/Zsh on Linux/macOS):**

export AZURE\_OPENAI\_ENDPOINT="https://your-openai-resource.openai.azure.com/"  
export AZURE\_OPENAI\_KEY="your-openai-api-key"  
export AZURE\_OPENAI\_FINE\_TUNED\_DEPLOYMENT\_NAME="your-fine-tuned-deployment-name"

**Example (for PowerShell on Windows):**

$env:AZURE\_OPENAI\_ENDPOINT="https://your-openai-resource.openai.azure.com/"  
$env:AZURE\_OPENAI\_KEY="your-openai-api-key"  
$env:AZURE\_OPENAI\_DEPLOYMENT\_NAME="your-fine-tuned-deployment-name"

#### Execution

Run the Python script:

python legal\_summarizer.py

#### Expected Results

The application will print the original legal documents (truncated) and then the summaries generated by your fine-tuned Azure OpenAI model. The summaries should be concise and focus on key arguments, parties, and outcomes, reflecting the fine-tuning for legal contexts.

**Example Output Structure (content will vary based on your model and fine-tuning):**

Summarizing legal document (first 200 chars): Case Name: Smith v. Jones  
Court: District Court of Anytown  
Date: July 8, 2025  
  
Facts: Plaintiff, John Smith, filed a lawsuit against Defendant, Sarah Jones, alleging breach of contract. The contract, signed on January 1, 2024, stipulated that Jones would deliver 100 widgets to Smith by March 1, 2024. Jones failed to deliver the widgets by the agreed-upon date....  
  
Summary 1:  
[Summary of Smith v. Jones focusing on breach of contract, parties, and court ruling]  
  
Summarizing legal document (first 200 chars): Agreement Type: Non-Disclosure Agreement (NDA)  
Parties: InnovateTech Inc. (Disclosing Party) and Global Solutions LLC (Receiving Party)  
Effective Date: June 1, 2025  
  
Purpose: To protect confidential information exchanged during discussions regarding a potential business collaboration. Confidential Information includes, but is not limited to, trade secrets, business plans, financial data, and technical specifications....  
  
Summary 2:  
[Summary of NDA focusing on purpose, obligations, and governing law]