CHAPTER 7

Advanced NLP Techniques and Language Understanding

Now that we've established the foundations of NLP, let's roll up our sleeves and tackle the fun stuff—the advanced techniques that make language AI truly powerful. I've spent years watching businesses struggle to keep up with customer messages across dozens of channels. But here's the thing: modern NLP has changed the game completely. From banks handling thousands of queries daily to ecommerce sites managing product questions 24/7, we're seeing real-world solutions that seemed like sci-fi just a few years ago.

Imagine a smart Q&A system that doesn't just match keywords but actually "gets" what people are asking. Think about the last time you searched through a massive company wiki or documentation page. Frustrating, right? We'll explore how to create systems that cut through the noise and pull out exactly what you need. It's not just about understanding words—it's about grasping context and intent, like knowing the difference between "How do I reset my password?" and "Why can't I log in?"

Chat support used to mean endless wait times and frustrated customers. Not anymore. Consider how Netflix uses NLP to help subscribers find shows they'll love through natural conversations. These aren't just fancy tech demos—they're practical solutions that save companies real money and make life easier for both businesses and customers. Throughout this chapter, I'll show you how to build similar systems with tools you can access today.

Working with Language Understanding Models

Language understanding models interpret, comprehend, and generate human language in a way that captures nuances, context, and intent. These models go beyond

the basic tasks like translation and keyword recognition, enabling more advanced, intuitive machine–human interactions. Table 7-1 summarizes the core capabilities of language understanding models.

Table 7-1. Core capabilities of language understanding models

Capability	Description
Intent recognition	The identification of the purpose behind a user's input, allowing applications to respond appropriately to users' requests.
Entity extraction	The categorization of key pieces of information from text—such as names, locations, or dates—to help understand context and specifics of a given conversation.
Contextual understanding	The interpretation of text within specific contexts, such as sentiment, tone, or culture.
Language generation	The production of human-like text responses, enabling machines to engage in conversation in a meaningful way.
Dialog management	The maintenance of conversation flow. This enables the model to recall past interactions and manage the conversation state, ensuring responses remain relevant to the user's input.
Customization of solutions	The adaptation of models to specific domains or business requirements, improving accuracy and relevance in specialized contexts.

Creating Intents, Utterances, and Entities

When developers are working in the realm of conversational AI, they must create models that understand and process human language effectively. Azure AI's conversational language understanding (CLU) capability allows developers to build sophisticated language understanding models that can recognize user intents, extract relevant information, and provide accurate responses. This development process includes three key components: intents, utterances, and entities.

In Azure AI's CLU, *utterances* are varied phrases or sentences that users may input while interacting with an application that's powered by a language understanding model. They represent the different ways users express their desires, requests, or questions to the system. For example, a user might say, "What's the weather like today?" or "Tell me the latest news," both of which serve the same purpose through different expressions.

Intents are the underlying goals or actions that users aim to achieve with their specific utterances. They represent the tasks that are embedded in the user's input. When you define intents, you train the model to accurately interpret and categorize the aims behind different utterances. For instance, an intent such as `GetBriefing` can be associated with utterances like "Get me my news briefing for the day," or "Show me today's headlines," which indicate the user's desire to receive the latest news updates.

Entities provide context for an intent by identifying and categorizing key pieces of information within utterances. They help the model understand and act on particular

details in a user's request, enhancing the accuracy and relevance of the interaction. For example, in the utterance "Book a flight to Paris next Monday," the `Destination` entity would be "Paris" and the `Date` entity would be "next Monday."

There are three types of entities in CLU:

Learned entities
: These are the most common type of entity due to their flexibility. Learned entities are derived from the training data and can recognize a wide range of associated words and phrases within training utterances. They adapt to the language that users use, which allows the model to identify entities even when they're presented in various forms. For example, a learned entity for `FoodType` might recognize "sushi," "pizza," or "tacos," based on the training data.

List entities
: These entities are used when there's a predefined set of possible values. They are ideal for scenarios where specific categories or options need to be recognized, such as cities, product names, or predefined commands. For instance, a list entity for `AustralianCities` might include "Sydney," "Melbourne," "Brisbane," etc., ensuring that only valid city names are recognized.

Prebuilt entities
: These entities cover common data types and are ready to use without additional training. Prebuilt entities handle standard information such as names, dates, numbers, and locations. They simplify the development process by providing out-of-the-box recognition for frequently used data types. For example, a prebuilt `Number` entity can automatically recognize numerical values in an utterance, while a prebuilt `Date` entity can identify dates in various formats.

Here's an example of how utterances, intents, and entities play out. It chronicles the sequence of steps that a user and a bot follow in a simple booking scenario:

- *User*: "Book me a table for two at a sushi restaurant in Melbourne this Friday at 7 p.m."
- Recognized intent: `BookReservation`
- Extracted entities:
  - `CuisineType = "sushi"`
  - `PartySize = 2`
  - `Location = "Melbourne"`
  - `DateTime = "Friday at 7 p.m."`

- System action: Query restaurant database with those parameters
- *Bot reply*: "Sure! I found three sushi restaurants in Melbourne available this Friday at 7 p.m. for two people: Sushi Haven, Ocean Roll, and Tokyo Bites. Which one would you like to choose?"

This allows the bot to properly respond, based on the context that the user has given.

By effectively defining and utilizing intents, utterances, and entities, CLU enables developers to build sophisticated conversational applications that accurately understand and respond to user inputs.

To get the best results, it's important to follow a few best practices when working with these components:

- Capture utterances that reflect diverse variations in phrasing, length, and grammatical correctness. That helps the model understand the nuances of human language.
- Define clear, distinct intents to ensure that all relevant user scenarios are covered.
- Use entities to extract actionable details from utterances, enhancing the model's ability to respond appropriately.

Building Language Understanding Models

Building effective language understanding models is a systematic process that's designed to ensure that the models accurately interpret and respond to user inputs. This process consists of four essential steps: training, evaluation, implementation, and refinement.

Training the model to recognize intents and entities

The first step in building a language understanding model is training it to recognize intents and entities by using sample expressions. To recap, intents represent the user's goals or the actions they wish to perform, while entities are specific pieces of information within the user's input that provide context for these intents. During training, you provide the model with a diverse set of sample utterances that exemplify the different ways in which users might express their intents. For example, for an intent like `BookFlight`, sample utterances could include, "I need to book a flight to New York," "Reserve a ticket for me to NYC next Monday," or "Can you help me find a flight to JFK?" Alongside these utterances, you should annotate entities such as `Destination`, `Date`, and `PassengerCount` to help the model identify and categorize key information within each expression.

To ensure effective intent classification, you should aim to provide at least 5–10 utterances per intent as a bare minimum for a simple proof of concept. This is the lower

bound supported by CLU projects. For production-ready models, you should plan on supplying 15–20 diverse utterances per intent in straightforward domains and scale up to 30–50+ examples in complex or highly variable domains where user phrasing overlaps significantly. This helps the model generalize better and reduces misclassifications. This training process also enables the model to learn patterns and variations in language, which in turn enhances its ability to accurately interpret user inputs in real-world scenarios.

Evaluating model performance with labeled datasets

Once you've trained the model, the next crucial step is to evaluate its performance using datasets with predefined labels. You'll test the model with a separate set of data that it did not encounter during training to assess its ability to generalize and accurately predict intents and entities. Commonly used metrics include precision, recall, and the F1-score. *Precision* indicates the proportion of correctly identified intents relative to all predicted intents, while *recall* measures the proportion of correctly identified intents relative to all actual intents in the dataset. The F1 score is then defined as the harmonic mean of precision and recall which by construction gives a single metric that balances the two. By applying these metrics, you can identify areas where the model excels and areas that require improvement. Additionally, you can use confusion matrices to visualize how often the model confuses one intent with another. That will give you insights into specific challenges that you need to address through further training or data augmentation.

Handling ambiguous user input and improving accuracy

Ambiguous user input arises when utterances lack context or contain phrases that map to multiple intents. For example, "Book this for me" could refer to flights, restaurants, or appointments. Such ambiguity reduces classification accuracy. To address this, implement follow-up clarifying questions, such as "Sure, would you like to book a flight, hotel, or restaurant?" You should also leverage conversation context carryover by storing previous dialog states to inform current intent detection and using contextual embeddings from transformer models like BERT that capture sentence-level meaning. This will further reduce ambiguity.

Entity recognition errors often stem from unseen mentions, incorrect span detection, or ambiguous categories. You can improve accuracy through *data augmentation*—generating paraphrases and slot value variations to broaden entity coverage—and by integrating gazetteers or lookup dictionaries to supplement learned entities. You can also enforce validation schemas and postprocessing rules to ensure correct entity spans and types. Finally, implementing joint modeling of intents and entities allows the model to exploit inter-task dependencies, resulting in better overall performance.

Implementing your trained model on a publicly accessible endpoint

After you successfully train and evaluate the model, the next step is to implement it on a publicly accessible endpoint. This involves deploying the model to a cloud service, such as Azure AI, which provides the infrastructure needed to host the model and make it accessible to your applications. By deploying the model to an endpoint, you enable seamless integration with various applications, such as chatbots, virtual assistants, and other conversational interfaces.

During deployment, you'll typically need to configure the endpoint for scalability and reliability, ensuring it can handle varying levels of user traffic without compromising performance. Additionally, you'll need to secure the endpoint with appropriate authentication and authorization measures to protect sensitive data and maintain user privacy. Once you've deployed the model, it will be able to process real-time user inputs, interpret intents and entities, and return meaningful responses, enhancing the interactivity and functionality of your applications.

Analyzing predictions and refining the model

The final step in building language understanding models is continuously analyzing the model's predictions and refining its learning accordingly. After you deploy your model, you must monitor how it performs in real-world scenarios. This involves collecting and reviewing data on how accurately the model identifies intents and entities, as well as gathering user feedback to identify any misunderstandings or errors. Techniques such as error analysis can help pinpoint specific instances where the model fails to correctly interpret user inputs. These insights will guide your next steps.

Begin the analysis and refinement process, by examining intent errors using a confusion matrix that highlights incorrect assignments (false positives) and missed assignments (false negatives). Focus on examples where the model indicates low confidence, or where the intents are vital to achieving your business objectives.

Sort entity mistakes into three categories:

Missing entities
: These occur when the model fails to recognize expected values.

Boundary errors
: These occur when the captured text spans are too large or too small.

Ambiguity errors
: These occur when expressions are classified into incorrect categories.

Record how frequently each error type occurs for every entity to help identify recurring patterns. Then, prioritize which errors to fix first by comparing how often they occur with their potential impact on your business. Creating a chart to map error

volume to impact or running a Pareto analysis can help you focus on the most common or costly mistakes first.

Armed with these insights, refine the model by updating the training data with new examples, adjusting your intent and entity definitions, or tweaking parameters to enhance its performance. By following this iterative refinement process, you'll ensure that the model evolves over time and adapts to changing user behaviors and language patterns, helping maintain high levels of accuracy and relevance.

Optimizing Language Understanding Models

Optimizing language understanding models is an essential part of building conversational systems that accurately understand user intents, handle diverse utterances, and maintain high performance over time. One primary optimization method is retraining models based on performance evaluations. By regularly assessing a model's accuracy and effectiveness, you can identify areas where it needs improvement. For instance, if the model frequently confuses the `BookFlight` and `CancelFlight` intents, you can provide additional diverse examples of each intent to help it distinguish between them more effectively. Enhancing the diversity and quality of your training data is one of the best ways to help a model generalize. Including synonyms, colloquialisms, and varied sentence structures ensures that the model can handle a wide range of user inputs. You can integrate Azure OpenAI to enhance this process by generating diverse, realistic utterance suggestions to expand the training dataset.

You should also continuously refine intents and entities to enhance the model's precision and contextual understanding. As the application evolves, new user goals may emerge, and that means you'll need to add new intents. Regularly assessing user interactions helps ensure that the model remains aligned with user needs. Improving entity extraction, by defining more specific entities or refining existing ones, helps the model accurately capture key information within user utterances, such as dates, locations, and product names.

Active learning and user feedback are powerful tools that you can use for ongoing optimization. You can implement mechanisms where the model actively requests feedback on uncertain predictions, enabling targeted data collection to improve model accuracy with minimal effort. Encouraging users to provide feedback on the system's responses helps identify common issues and areas for improvement, which you can then incorporate into model retraining.

You can also utilize Azure's suite of monitoring and diagnostic tools, such as Azure Monitor and Application Insights, to provide continuous oversight of model performance. These tools track key performance metrics like intent recognition accuracy, response times, and user satisfaction scores, allowing you to quickly identify and address performance bottlenecks. They also support comprehensive logging and diagnostics to capture detailed information about user interactions, which is

invaluable for troubleshooting and improving model behavior. In combination with semantic analysis from advanced language models, such as those powered by Azure OpenAI, you can gain deeper insight into where and why misclassifications occur, helping you refine both intent recognition and entity extraction over time.

Another essential aspect of maintaining optimized models in production is using effective deployment strategies. By leveraging Azure's scalable infrastructure, you can ensure that your model can handle varying loads without compromising performance. Establishing CI/CD pipelines allows you to automate the deployment process and integrate updates seamlessly into the production environment. Version control and A/B testing let you compare different model iterations and deploy the best-performing version with confidence. It's also very important to perform post-deployment monitoring and maintenance, because optimization does not end with deployment. After deployment, real-time monitoring helps you detect and promptly address performance degradation or emerging issues. By implementing regular updates based on new data, user feedback, and evolving application requirements, you can maintain high accuracy and relevance.

Finally, adhere to best practices to ensure a systematic and effective optimization process. Treat optimization as an ongoing, iterative process rather than a one-time task. Conduct comprehensive testing across different scenarios to ensure that the model performs reliably under various conditions, and maintain thorough documentation of optimization processes, decisions, and outcomes to support collaboration and knowledge sharing within your team.

Backing Up and Recovering Language Understanding Models

To ensure your conversational language understanding models are available and resilient, you should have a clear backup and recovery strategy. This is especially important if your applications depend heavily on these models. Start by identifying critical artifacts such as the project name, model name, and deployment name. Capture the definitions of key intents and entities early in the process. This ensures you can recover or redeploy the model accurately in the event of data loss or system failure, helping maintain uninterrupted service.

Here are the steps you'll need to perform to implement a backup and recovery strategy:

1. Set up two Azure AI Language resources in different Azure regions to facilitate failover in the event of a regional outage. This will ensure that the CLU model remains accessible, which will make sure service continuity is maintained. You should also select resource locations that Azure has paired with each other. This will reduce synchronization time and provide a prioritized recovery order during

widespread failures, because paired regions receive updates at different times and have optimized network links.

2. Export the project assets from the primary Azure AI Language resource. To do this, generate an export job using an API request. This will return a job ID and a URL that you can use to track the job's status.
3. Once the export job completes, use the provided URL to download the exported project assets. These assets will include the settings, intents, entities, and utterances of the project. You also need to include your travel domain definitions so that in the secondary resource, `BookFlight` utterances like "Travel from Cairo to Paris" and `CancelReservation` phrases will continue to be recognized correctly.
4. Replicate the project to the secondary AI Language resource in the other region. You'll need to submit an import job using an API request and include the keys and endpoint of the secondary resource.
5. Train the models in the new environment to ensure that they are operational. After training, deploy the models so they can be accessed through runtime APIs.
6. To maintain performance and reliability, regularly check and synchronize the projects across the primary and secondary resources. To do this, you'll have to compare the last modified timestamps of both projects and update the secondary project with any changes made in the primary project.
7. Design the system to seamlessly switch to the secondary resource in the event of an outage in the primary region. This will help prevent service interruption and ensure continued access.

Practical: Building and Integrating Your Own Language Understanding Model

In this exercise, you'll create a custom language understanding model for a travel agency's chatbot. The chatbot will assist users in booking flights, hotels, and car rentals. Your goal is to build, train, and integrate the model using Azure AI services, ensuring that it can accurately understand and process user requests.

This exercise will guide you through all the steps of the process.

Step 1: Create an Azure AI Language resource

A. Click the blue "Create a resource" button in the top-left corner of the Azure portal dashboard.

B. Search for the Language service and select it from the search results.

C. Click Create to create a new Language resource.

D. On the Basics tab, configure the resource as follows:

i. Subscription: select your Azure subscription from the drop-down menu.

ii. Resource group: create a new resource group by clicking on "Create new," enter TravelChatbotRG as the name, and click OK. Alternatively, you can select an existing resource group if you have one.

iii. Region: select the region closest to you (e.g., East US) to reduce latency.

iv. Name: enter TravelLanguageResource as the name of your Language resource.

v. Pricing tier: select Standard S (or Free F0 if it's available and suitable for your needs).

E. Click "Review + create" at the bottom of the page. After validation passes, click Create to deploy the resource.

F. Wait for deployment. You'll see a message saying "Your deployment is underway," followed by one saying "Your deployment is complete."

Step 2: Set up Language Studio

A. Access Language Studio by doing the following:

i. Open a new browser tab and navigate to Language Studio (*https://language.azure.com*). Sign in with your Azure account credentials if prompted to do so.

B. Select your Language resource:

i. After you sign in, you may see a prompt to "Select a Language resource."

ii. From the drop-down menu, select TravelLanguageResource (the resource you just created).

C. Click OK or Continue to proceed.

Step 3: Define intents, utterances, and entities

A. Create a CLU project:

i. Click the "Create new project" button on the Language Studio dashboard.

ii. Under "Select a feature," choose Conversational Language Understanding.

iii. Click Next to proceed to the "Enter basic information" screen.

B. Configure the project as follows (see Figure 7-1):

i. Name: enter **TravelChatbotCLU**.

ii. Description: enter **CLU model for travel agency chatbot to understand booking intents.**

iii. Language: select English (or your preferred language) from the drop-down menu.

iv. Project: ensure that Conversation is selected.

v. Click Next to continue.

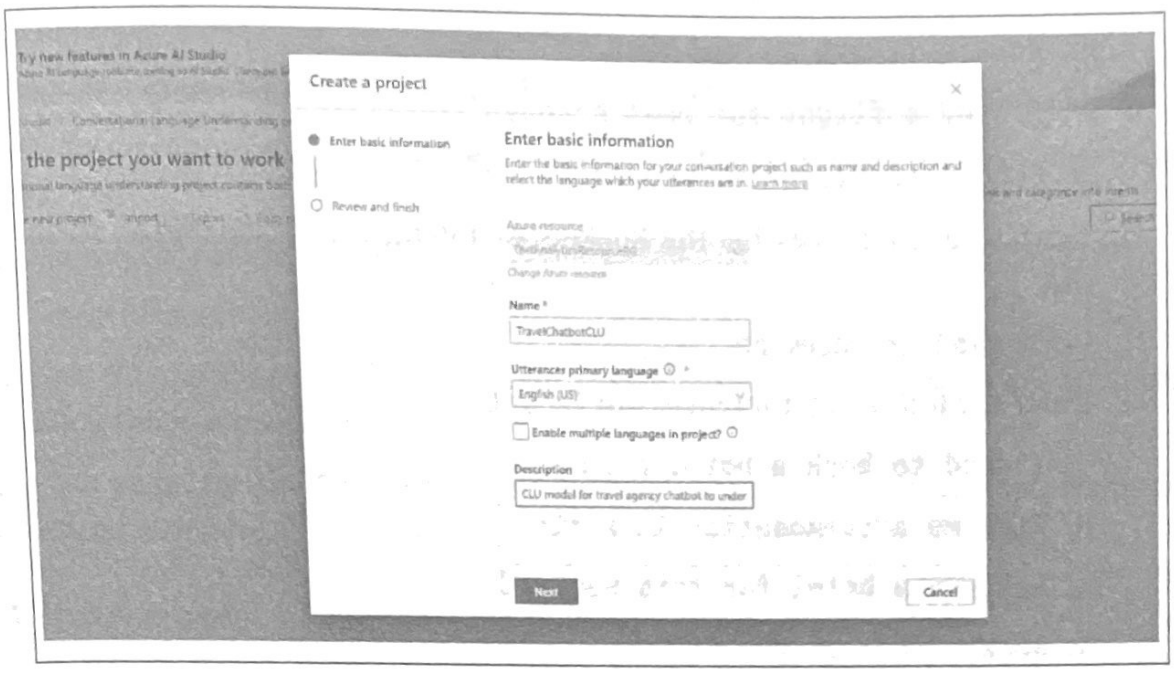

Figure 7-1. Entering basic information for the CLU project on Language Studio

vi. On the Review and finish page, the Resource should already be displayed as `TravelLanguageResource`.

vii. Ensure that the Location matches the region you selected earlier.

viii. Click "Create project."

C. In your CLU project dashboard, locate and select Intents in the left menu.

D. Click "Add intent" to bring up a screen where you can start adding intents. Proceed as follows:

i. Name: enter **`BookFlight`**.

ii. Description: enter **`Intent to handle flight booking requests`**.

iii. Click Save, then click "Add intent" again.

iv. Name: enter **`BookHotel`**.

v. Description: enter **`Intent to handle hotel booking requests`**.

vi. Click Save, then click "Add intent" again.

vii. Name: enter **`RentCar`**.

viii. Description: enter **`Intent to handle car rental requests`**.

ix. Click Save.

E. Next, you'll add utterances for the intents. Select `BookFlight` in the intents list.

F. Go to the Utterances tab, and add the following utterances one by one:

i. **`I want to book a flight`**

ii. **`Can I reserve a plane ticket?`**

iii. **`Book a flight to New York`**

iv. **`I need a flight for next Monday`**

v. **`Find me a flight to London`**

G. Next, add utterances for the `BookHotel` intent. Select `BookHotel` in the intents list.

H. Go to the Utterances tab.

I. Enter the following utterances one by one:

i. **`I need to book a hotel room`**

ii. **`Find me accommodation in Paris`**

iii. **`Reserve a hotel for this weekend`**

iv. **`Book a room at the Hilton`**

v. **`I want a hotel in Tokyo`**

J. Then, to start adding utterances for the `RentCar` intent, select `RentCar` in the intents list.

K. Go to the Utterances tab, and enter the following utterances one by one:

i. **`I need to rent a car`**

ii. **`Book a rental car for my trip`**

iii. **`Can I get a car rental for tomorrow?`**

iv. **`Reserve an SUV for next week`**

v. **`I want to hire a car in Los Angeles`**

L. Locate and select Entities in the left menu.

M. Click "Add entity" to bring up a screen where you can start entering entities. Proceed as follows:

i. Name: enter **`Location`**.

ii. Type: select "Machine learned entity."

iii. Click Save, then click "Add entity" again.

iv. Name: enter **`Date`**.

v. Type: select "Machine learned entity."

vi. Click Save.

N. To start labeling entities in utterances, select Intents in the left menu.

O. Select the `BookFlight` intent, and go to the Utterances tab.

P. To configure the `"Book a flight to New York"` utterance, do the following:

   i. Highlight "New York" in the utterance.

   ii. A small menu will appear. Select "Location" to label it.

Q. Next, to configure the `"I need a flight for next Monday"` utterance, do the following:

   i. Highlight "next Monday."

   ii. Select "Date."

R. Repeat this process for all utterances where locations or dates are mentioned, labeling them appropriately.

S. Do the same for the `BookHotel` and `RentCar` intents, labeling any locations or dates mentioned.

Step 4: Train your model

A. Review your labeled data, ensuring that all your sample utterances have the correct intents assigned and entities labeled.

B. Navigate to "Train model" and select "Training jobs" in the left menu.

C. Start a training job by clicking "Train new model."

D. Configure the training as follows:

   i. Model name: enter **`TravelChatbotModel`**.

   ii. Data splitting: select "Automatic splitting," which will automatically split your data into training and validation sets.

   iii. Training mode: if English is your selected language, and you should choose "Standard training" for faster training. For other languages or advanced scenarios, select "Advanced training."

   iv. Click Train to start the training process.

   v. Wait for training to complete. It may take a few minutes, and you can monitor the progress on the training page. Once it's complete, you'll see a notification and the model's status will change to Trained.

Step 5: Evaluate and improve your model

A. Evaluate your model's performance by viewing its metrics:

   i. Select Model Performance.

   ii. This will display metrics such as precision, recall, and the F1 score for your model.

B. To improve your model's accuracy if its metrics are not satisfactory, consider adding more utterances and correcting any mislabeled data:
   i. To add more utterances, go back to the Intents tab and add more diverse utterances for your intents.
   ii. Ensure that all entities in the utterances are correctly labeled.
C. After making these changes, retrain the model by repeating the training process in Step 4.

Step 6: Deploy your model

A. Select Deploy in the left menu, and click "Deploy model."
B. Configure the deployment as follows:
   i. Deployment name: enter **`TravelChatbotDeployment`**.
   ii. Model version: select `TravelChatbotModel` (the model you trained).
   iii. Click on Deploy, and wait for deployment to complete. It may take a few minutes. Once it's done, the model's status will change to Deployed.

Step 7: Integrate the model into your application

A. Retrieve the API key and endpoint as follows:
   i. Go back to the Azure portal (*https://portal.azure.com*).
   ii. Navigate to your TravelLanguageResource resource.
   iii. In the left menu, under Resource Management, select "Keys and Endpoint."
   iv. Copy one of the key values (these are your API keys), and the endpoint URL.
B. To use the runtime API, open a text editor or IDE that supports Python, and copy and paste the following code into your editor:

```
import requests
import json

# Replace the placeholders with your actual values
endpoint = "https://your-resource-name.cognitiveservices.azure.com"
api_key = "your-api-key"
project_name = "TravelChatbotCLU"
deployment_name = "TravelChatbotDeployment"

# The endpoint for calling the deployed model
url = (
    f"{endpoint.rstrip('/')}"
    "/language/:analyze-conversations"
    "?api-version=2022-10-01-preview"
)
```

```
# Headers including the API key
headers = {
    "Ocp-Apim-Subscription-Key": api_key,
    "Content-Type": "application/json"
}

# The data to send in the request
data = {
    "kind": "Conversation",
    "analysisInput": {
        "conversationItem": {
            "text": "I want to book a flight to New York next Monday",
            "id": "1",
            "participantId": "user1"
        }
    },
    "parameters": {
        "projectName": project_name,
        "deploymentName": deployment_name,
        "stringIndexType": "TextElement_V8"
    }
}

response = requests.post(url, headers=headers, json=data)
result = response.json()
print(json.dumps(result, indent=2))
```

Replace the placeholders with your actual values and verify the other code as follows:

i. Replace `https://your-resource-name.cognitiveservices.azure.com` with your actual endpoint URL (from the Azure portal).

ii. Replace *your-api-key* with the API key you copied earlier.

iii. Ensure that the `project_name` matches the name of your CLU project (e.g., `TravelChatbotCLU`).

iv. Ensure that the `deployment_name` matches your deployment name (e.g., `TravelChatbotDeployment`).

C. Run the code:

i. Save the file as *test_travel_chatbot.py*.

ii. Open a command prompt or terminal.

iii. Navigate to the directory containing your script.

iv. Run the script using the `python test_travel_chatbot.py` command.

D. Interpret the results. The script will output a JSON response that contains the predicted intent and extracted entities. Here's some sample output:

```
{
  "kind": "ConversationResult",
  "result": {
    "query": "I want to book a flight to New York next Monday",
    "prediction": {
      "topIntent": "BookFlight",
      "projectKind": "Conversation",
      "intents": [
        {
          "category": "BookFlight",
          "confidenceScore": 0.95
        }
      ],
      "entities": [
        {
          "category": "Location",
          "text": "New York",
          "offset": 24,
          "length": 8,
          "confidenceScore": 0.98
        },
        {
          "category": "Date",
          "text": "next Monday",
          "offset": 33,
          "length": 11,
          "confidenceScore": 0.97
        }
      ]
    }
  }
}
```


Step 8: Clean up your resources

A. To avoid incurring unnecessary charges, delete the resources if you no longer need them. You may wish to defer this until after completing the second practical exercise in this chapter. You can delete the resource group as follows:

   i. In the Azure portal, navigate to "Resource groups" in the left menu.

   ii. Find and click on TravelChatbotRG.

   iii. Click "Delete resource group" at the top of the screen.

   iv. Type **`TravelChatbotRG`** in the confirmation box.

   v. Click Delete.

B. Confirm deletion by waiting for the deletion process to complete. It may take a few minutes, and it will delete all resources within the resource group, including the language resource and bot service.

And with that, you have successfully built, trained, and integrated a custom language understanding model for a travel agency chatbot using Azure AI services! You have also created a robust and effective model for your applications.

Building Question-Answering Solutions

A *question-answering solution* is an application that uses NLP and machine learning algorithms to understand, interpret, and respond to user questions expressed in natural language. These solutions can analyze large volumes of data to provide direct and helpful answers, making them well suited to scenarios such as customer service and educational tools. Now let's walk through how to build your own question answering solution with Azure AI.

Understanding Question-Answering Solutions

Azure AI Language provides robust capabilities for creating sophisticated question-answering solutions. These solutions leverage the following key features to enhance the user experience. In this section, we'll explore how they work and I'll guide you through building your own question-answering solution with Azure AI.

Key features of question-answering solutions

There are three key features of question answering solutions: semantic search, knowledge mining, and customization and tuning:

Semantic search
: This feature enables the model to understand the context and intent behind a user's query to provide more accurate and relevant answers. It goes beyond keyword matching by considering the meanings of words and the context in which they are used.

Knowledge mining
: Azure AI can extract useful information from unstructured data sources such as documents, FAQs, manuals, and web pages. It organizes this information into a knowledge base that can be queried effectively.

Customization and tuning
: You can tailor the question-answering models to better fit your specific needs by editing question-answer pairs, defining synonyms, and adding metadata tags. This helps ensure that the system will provide the most relevant responses to user queries.

Fundamentals of question answering

Question answering (QA) is a subfield of natural language processing focused on building systems that automatically answer questions posed by humans in natural language. QA systems can vary in complexity and functionality, ranging from simple keyword-based searches to advanced systems that can understand and process complex queries.

Components of a QA system. There are four main components of a QA system, each operating in sequence:

1. The *question processing* component comes into play first. It's responsible for understanding the user's question and involves tasks such as identifying the type of question, extracting keywords, and determining the context. Efficient question processing must balance accuracy and speed. QA systems often use lightweight tokenization and basic parsing to achieve subsecond performance while reducing CPU usage—each additional analysis step increases processing time and affects overall throughput.
2. Next comes *information retrieval*, in which the system searches through its knowledge base or external data sources to find relevant information that can provide an answer to the question. Information retrieval performance depends on the indexing strategy and search algorithm complexity. Inverted index lookups deliver high throughput for keyword queries, while dense vector search methods improve recall at the expense of higher memory and compute requirements, as recent vector database benchmarks show.
3. Then, in the *answer processing* step, the system processes the retrieved information to generate a concise and accurate answer. This may involve extracting specific data from documents or generating new text based on the information retrieved. Extractive methods are fast, often completing in tens of milliseconds, whereas transformer-based generation models provide richer responses but can require hundreds of milliseconds to seconds of inference time, depending on model size and output length.
4. Finally, in the *response generation* step, the system presents the answer to the user in a clear and contextually appropriate format. Simple template-based rendering adds minimal delays (often, under a millisecond). Dynamic natural language generation using LLMs requires token-by-token generation that can significantly increase latency, especially for longer answers.

Types of QA systems. There are two types of QA systems:

Closed-domain QA
: These systems are designed to answer questions about a specific domain or dataset. For example, a QA system for medical information would only answer questions related to medical topics.

Open-domain QA
: These systems can handle questions about a wide range of topics. They often leverage large datasets, like Wikipedia, to find answers.

Real-world use cases. QA systems are already in use in many industries. Examples include:

Customer support
: QA systems can automate customer support by providing instant answers to frequently asked questions. This reduces the workload of human agents and improves response times for customers.

Healthcare
: In the medical field, QA systems can assist healthcare professionals by quickly providing answers to medical queries, helping with diagnosis, and recommending treatments based on a vast repository of medical literature.

Monitoring usage and identifying areas for improvement

To ensure that your QA solution remains effective, compile usage logs for every query and record detail such as the user's text, the model's confidence score, and the response timestamp. It's important to track queries that return low confidence or no answer, because they indicate gaps in the knowledge base. To assess response accuracy and relevance, collect user feedback through ratings or click-through behavior. You should also set up dashboards in Azure AI Foundry or Power BI to visualize key metrics such as unanswered query rate, fallback counts, and average confidence scores. Review the top unanswered questions weekly to identify areas where your knowledge base needs to be updated or expanded. When you're fine-tuning semantic search parameters or question-answer pairs, use A/B testing to compare variations. Finally, to continuously improve coverage and accuracy, you should establish a feedback loop where you incorporate user corrections and new FAQ entries into your knowledge store on a regular schedule.

Now that you have a general understanding of these solutions, you can start building one of your own.

Practical: Building Your Own Question-Answering Solution

Suppose you're working for an organization that wants to implement a customer support chatbot that can answer frequently asked questions (FAQs) about their products and services. Your goal is to use Azure AI services to create a question-answering solution that can understand user questions and provide accurate answers from a knowledge base. In this section, we'll walk through how to do this.

You'll create a knowledge base for the question-answering solution by adding FAQs and custom question-and-answer pairs. You can use the Azure AI Language resource you created in the previous exercise, or, if you cleaned up your resources at the end of that exercise, follow the instructions in "Step 1: Create an Azure AI Language resource" on page 251 to create a new one.

Step 1: Build the question-answering knowledge base

A. Create a question-answering project in your Azure AI Language resource:
   i. Click the "Create" button in the Language Studio dashboard.
   ii. Under "Select additional features," leave the Custom features as is.
   iii. Click "Continue to create your resource" to proceed.

B. Configure the project as follows:
   i. Project name: enter **CustomerSupportQA**.
   ii. Description: enter **Question answering knowledge base for customer support chatbot**.
   iii. Language: select English (or your preferred language) from the drop-down menu.
   iv. Enter this as the default answer to use when no answer is found: "I'm sorry, I couldn't find an answer to your question. Please contact our support team for assistance."
   v. Click Next to continue.

C. Select a resource:
   i. The Azure AI Language resource should already display CustomerSupportLanguage.
   ii. For the Azure AI Search resource, select CustomerSupportSearch from the drop-down menu.
   iii. Click "Create project."

D. On the project dashboard, click "Add sources."

E. You can populate the knowledge base with existing FAQs from your own documents (PDF, Word, or text files containing question-and-answer pairs) or from

published web pages, by providing URLs. For this exercise, we'll add sample FAQs manually, so you can skip adding data sources at this time.

F. Manually add question-and-answer (Q&A) pairs. Select "Edit knowledge base" in the left menu and click "Add question pair" to add each of the following items in turn (click Save after entering each pair):

   i. Question: enter **What is your return policy?**

   ii. Answer: enter **Our return policy allows you to return products within 30 days of purchase for a full refund.**

      i. Click on Save.

      ii. Click on "Add Q&A pair" again.

   iii. Question: enter `How can I track my order?`

   iv. Answer: enter `You can track your order by logging into your account and visiting the Order History section.`

   v. Question: enter `Do you offer international shipping?`

   vi. Answer: enter `Yes, we offer international shipping to selected countries. Please check our shipping policy for more details.`

   vii. Question: enter `How do I reset my password?`

   viii. Answer: enter `Click "Forgot Password" on the login page and follow the instructions to reset your password.`

G. Review the Q&A pairs to ensure that all questions and answers are correctly entered.

Step 2: Train and test your model

A. Click "Save and train" in the top-right corner of the screen and wait for the training process to complete. When it's done, you'll see a "Training successful" message.

B. To test your solution, select Test in the left menu.

C. In the "Test your knowledge base" section, enter a question that's similar to those in your Q&A pairs. For instance, you can enter `How do I track my order?`

   The model should provide the corresponding answer, such as `You can track your order by logging into your account and visiting the Order History section.`

D. Enter other questions to test the robustness of the model, such as these:

   i. `What's your policy on returns?`

   ii. `Can I get a refund?`

The model should provide the most relevant answer based on your Q&A pairs.

Step 3: Deploy your model

A. Deploy the model:
   i. Select Deploy in the left menu.
   ii. Click Deploy.
   iii. For the deployment name, enter **`CustomerSupportQADep`**.
   iv. Click Deploy.

B. Wait for deployment to complete. It may take a few minutes. Once it's done, the model's status will change to Deployed.

Step 4: Integrate the model into your application

A. Get the API key and the endpoint:
   i. Go back to the Azure portal (*https://portal.azure.com*).
   ii. Navigate to your "CustomerSupportLanguage" resource.
   iii. In the left menu, under Resource Management, select "Keys and Endpoint."
   iv. Copy one of the key values (these are your API keys) and the Endpoint URL.

B. Open a text editor or IDE that supports Python, and copy and paste the following code into your editor:

```
import requests
import json

# Replace the placeholders with your actual values
endpoint = "https://your-resource-name.cognitiveservices.azure.com"
api_key = "your-api-key"
project_name = "CustomerSupportQA"
deployment_name = "CustomerSupportQADep"

# The endpoint for calling the deployed question answering model
url = f"{endpoint}/language/:query-knowledgebases?api-version=2021-10-01"

# Headers including the API key
headers = {
    "Ocp-Apim-Subscription-Key": api_key,
    "Content-Type": "application/json"
}

# The question to ask
question = "How can I track my shipment?"

# The data to send in the request
data = {
```

```
        "question": question,
        "top": 1,
        "confidenceScoreThreshold": 0.2,
        "includeUnstructuredSources": True,
        "shortAnswerOptions": {
            "confidenceScoreThreshold": 0.2,
            "top": 1,
            "answerSpanRequest": {
                "enable": True,
                "confidenceScoreThreshold": 0.2,
                "topAnswersWithSpan": 1
            }
        },
        "knowledgeBaseQuestionAnsweringOptions": {
            "enable": True
        },
        "projectName": project_name,
        "deploymentName": deployment_name
    }

    # Send the request
    response = requests.post(url, headers=headers, json=data)
    result = response.json()

    # Print the result
    print(json.dumps(result, indent=2))
```

Replace the placeholders with your actual values and verify the other code as follows:

i. Replace `https://your-resource-name.cognitiveservices.azure.com` with your actual endpoint URL (from the Azure portal).

ii. Replace *your-api-key* with the API key you copied earlier.

iii. Ensure that the `project_name` matches the name of your question-answering project (e.g., `CustomerSupportQA`).

iv. Ensure that the `deployment_name` matches your deployment name (e.g., `CustomerSupportQADep`).

C. Run the code:

i. Save the file as *test_customer_support_qa.py*.

ii. Open a command prompt or terminal.

iii. Navigate to the directory containing your script.

iv. Run the script using the `python test_customer_support_qa.py` command.

Interpret the results. The script will output a JSON response that contains the answer to your question. Here's a sample output:

```
{
  "answers": [
    {
      "questions": [
        "How can I track my order?"
      ],
      "answer": "You can track your order by logging into your account\n\
      and visiting the 'Order History' section.",
      "confidenceScore": 0.95,
      "id": 1,
      "source": "Editorial",
      "metadata": {},
      "dialog": {
        "isContextOnly": false,
        "prompts": []
      }
    }
  ]
}
```

The `"answer"` field contains the answer from your knowledge base.

Step 5: Integrate with Azure AI Bot Service (optional)

If you want to create a chatbot that uses this question-answering model, you can integrate it with Azure AI Bot Service by following these steps:

A. Create a bot:
   i. In Language Studio, go to the Deploy tab.
   ii. Click the "Create bot" button next to your deployment.
   iii. That will redirect you to the Azure portal, which has a pre-filled form where you can create a new Azure Bot resource.

B. Configure the bot settings:
   i. Bot handle: enter **`CustomerSupportBot`**.
   ii. Subscription: ensure that your subscription is selected from the menu.
   iii. Resource group: select CustomerSupportRG from the menu.
   iv. Location: ensure that the location matches your other resources.
   v. Pricing tier: select F0 (Free) (or whichever option meets your requirements) from the menu.
   vi. Microsoft App ID and Password: leave the selections as is to auto-create.
   vii. Click "Review + create," then Create.

C. Wait until the bot deployment is complete.

D. Test the bot:

i. Navigate to your newly created Bot resource in the Azure portal.
ii. In the left menu, select "Test in Web Chat."
iii. Type `How do I reset my password?` into the chat window.
iv. The bot should respond with the corresponding answer from your knowledge base.

Step 6: Clean up your resources

A. To avoid incurring unnecessary charges, delete the resources if you no longer need them:
   i. In the Azure portal, navigate to "Resource groups" in the left menu.
   ii. Find and click on CustomerSupportRG.
   iii. Click "Delete resource group" at the top of the screen.
   iv. Type `CustomerSupportRG` in the confirmation box.
   v. Click Delete.
B. Confirm deletion by waiting for the deletion process to complete. This may take a few minutes. This action will delete all resources within the resource group, including the language resource, search resource, and bot service.

And with that, you have successfully built a custom question-answering solution with Azure AI services! You have created a knowledge base with question-and-answer pairs, trained and tested your model, and integrated it into an application. Now, this solution can provide users with quick and accurate answers to their questions, enhancing customer support and satisfaction.

Advanced Capabilities

You can build upon the foundational knowledge that you've gained so far to implement additional capabilities, such as multiturn conversations and alternate phrasing. In this section, I'll also show you how to add chit-chat and export a knowledge base.

Adding multiturn conversations

To create an effective knowledge base, you need to compile a comprehensive set of question-and-answer pairs. But to better understand user queries and provide accurate answers, your model may need to engage in a more dynamic kind of dialog in which additional questions are asked. This kind of interaction, known as a *multiturn conversation*, mimics the natural flow of human dialog by allowing follow-up questions and responses.

Once you've enabled multiturn conversations in your knowledge base, you can configure them using structured source material (such as documents or web pages) or manually craft follow-up prompts for specific Q&A pairs. For example, a flight agency knowledge base might include the question "How can I cancel a reservation?" Since the word *reservation* may pertain to a hotel, flight, or car rental, the system may need to respond with a clarifying question to provide the most relevant help. You can implement this by either attaching follow-up prompts to existing answers or crafting custom responses tailored to specific follow-up scenarios.

You must also maintain conversational context across turns to create a natural, coherent user experience. This requires implementing state tracking to store previous user inputs, entities, and dialog history in a context object, so the bot can reference prior information when it's interpreting new queries. You can apply design patterns, such as the state pattern, to define clear conversation stages and transitions to improve flow and reduce repetition.

Practical: Implementing multiturn conversations

Let's implement a question-answering solution for the travel agency with support for multiturn conversations. This section will walk you through the process.

Step 1: Create a question-answering project.

A. In Language Studio, click Projects in the left sidebar and click "New project."

B. Configure the project as follows:

   i. Project name: enter a name of your choice (e.g., "FlightAgencyQnA").

   ii. Description: enter a brief description (e.g., "Q&A project for Flight Reservation Agency").

   iii. Language: select the primary language for your chatbot (e.g., English).

   iv. Project type: choose "Custom question answering" from the menu.

   v. Click Create.

Step 2: Add new question-and-answer pairs.

A. Click on the QnA Pairs section in the left sidebar.

B. Enter the following question and answer:

   i. Question: **`How can I cancel a reservation?`**

   ii. Answer: **`You can cancel a reservation by visiting our website or contacting customer support.`**

C. Click Save.

D. For a more comprehensive knowledge base, add other relevant Q&A pairs by following the same steps (this is optional).

E. In the QnA Pairs section, click "View options" at the top right.

F. Select "Show context" from the drop-down menu. This will display the context tree view, which allows you to manage multiturn prompts effectively.

G. Locate the Q&A pair you added earlier (`How can I cancel a reservation?`) and click on it so you can edit it.

H. Click "Add follow-up prompts," and configure the prompt as follows:

   i. Display text: enter **`Is this for a hotel reservation?`**

   ii. Link to the answer by selecting or creating the Q&A pair that provides information on canceling hotel reservations.

I. Then click "Add follow-up prompt" again to configure another prompt:

   i. Display text: enter **`Is this for a flight reservation?`**

   ii. Link to the answer by selecting or creating the Q&A pair for canceling flight reservations.

J. After adding all necessary follow-up prompts, click Save.

K. To edit a follow-up prompt, click on it, edit the text, and click Save.

Step 3: Test multiturn conversations.

A. Select Test in the left sidebar.

B. In the text entry box, type **`How can I cancel a reservation?`**

C. To view the response and follow-up prompts, press Enter.

D. The chatbot should respond with `You can cancel a reservation by visiting our website or contacting customer support.` These follow-up prompts should appear as well:

   i. `Is this for a hotel reservation?`

   ii. `Is this for a flight reservation?`

E. Click on the second follow-up prompt.

F. The chatbot should provide a more specific answer related to flight reservations.

Step 4: Deploy your project.

A. Select Deploy in the left sidebar.

B. Verify that all your Q&A pairs and follow-up prompts are correctly configured.

C. Click "Deploy" and wait for the deployment to complete. A success message will appear when it's done.

Step 5: Integrate with applications.

A. Retrieve the endpoint URL and API key:

   i. In the Azure portal, navigate to your Language Service resource.

   ii. Go to the "Keys and Endpoint" page and make a note of the endpoint URL and one of the keys.

B. Set up the API integration:

   i. Use your key and endpoint to configure your application or bot to communicate with the deployed Q&A service.

   ii. Ensure that your application can send HTTP requests to the endpoint and handle JSON responses.

Example JSON requests and responses

When a user asks a question, your application will send a JSON request to the Q&A service.

Here's an example of what such a request might look like:

```
{
  "question": "How can I cancel a reservation?",
  "top": 10,
  "userId": "Default",
  "isTest": false,
  "context": {}
}
```

It contains the following fields:

- `question`: This is the user's query.
- `top`: This is the number of answers to return.
- `userId`: This is a unique identifier for the user.
- `isTest`: This indicates whether the request is for testing purposes.
- `context`: This maintains the conversation state; it's empty for the first question.

The service will respond with an answer and, if necessary, follow-up prompts, such as in the following JSON response:

```
{
  "answers": [
    {
      "questions": ["How can I cancel a reservation?"],
      "answer": "You can cancel a reservation by visiting our\n\
      website or contacting customer support.",
      "score": 100.0,
```

```
        "context": {
          "prompts": [
            {
              "displayOrder": 0,
              "qnaId": 2,
              "displayText": "Is this for a hotel reservation?"
            },
            {
              "displayOrder": 1,
              "qnaId": 3,
              "displayText": "Is this for a flight reservation?"
            }
          ]
        }
      }
    ]
  }
```

Here, `answers` is the array of possible answers and `context.prompts` are the follow-up prompts the service could send to refine the user's query.

If the user selects a follow-up prompt, another request will be sent to the service. Here's a follow-up JSON request from this scenario:

```
{
  "question": "Is this for a flight reservation?",
  "top": 10,
  "userId": "Default",
  "isTest": false,
  "qnaId": 2,
  "context": {
    "previousQnAId": 1,
    "previousUserQuery": "How can I cancel a reservation?"
  }
}
```

Here, `qnaId` is the ID of the previous answer, and `context` maintains the flow of conversation.

The service will then provide a specific answer based on the user's response to the follow-up question. For example:

```
{
  "answers": [
    {
      "questions": ["Is this for a flight reservation?"],
      "answer": "You can cancel your flight reservation by logging\n\
      into your account on our website and selecting 'Cancel\n\
      Reservation'.",
      "score": 100.0,
      "context": {
        "prompts": [
          {
```

```
            "displayOrder": 0,
            "qnaId": 4,
            "displayText": "Do you need help with something else?"
          }
        ]
      }
    }
  ]
}
```

By following these steps, you can effectively set up and test multiturn conversations in your Azure AI question-answering solution, enhancing the user experience by making the solution mimic natural, human-like interactions.

Chatbots occasionally get stuck in loops, repeating the same prompts or questions, which will cause frustration. You can prevent these loops from occurring by tracking recent interactions and detecting repeated intents. When the same question arises more than twice, you should escalate to a fallback handler or hand it over to a human agent. You should also define clear termination conditions for multistep flows so that the bot will exit gracefully once a task is completed, or offer to connect users to live support if confusion persists.

Alternate phrasing

Alternate phrasing is the process of adding variations on questions to the knowledge base to cover the different ways in which users may ask the same question. By implementing alternate phrasing, you ensure that your solution will be able to respond to a broader range of user inputs. You can do this by enriching the knowledge base with diverse question variants that are linked to the same answer, improving the model's ability to understand user intent, regardless of the phrasing.

To discover possible question variants, start by mining user logs with clustering methods to reveal the most common alternative expressions and synonyms. You can also group queries that correspond to the same intent, using proven query log analysis techniques. Draw on linguistic resources, such as synonym lexicons and the lexico-syntactic patterns that are identified in paraphrase generation research, to systematically expand your question templates with new phrases and structural variations. You can also use back-translation to generate paraphrases: translate questions into another language and then back into the original language, and integrate new variants into the knowledge base after verifying them for accuracy. Repeat these processes regularly as you collect new user data, so that the system evolves along with language and usage patterns.

We'll walk you through the steps involved in implementing alternative phrasing in the following subsections.

Practical: Alternate Phrasing

Step 1: Access your project in Language Studio.

A. Select your Language resource and open your existing question-answering project (`FlightAgencyQnA`).

B. Select "Edit knowledge base" in the left sidebar.

Step 2: Add alternate phrasing to a Q&A pair.

A. Find the Q&A pair to which you want to add alternate phrasings (e.g., `How can I cancel a reservation?`), and click it to open it for editing.

B. Locate the "Alternate questions" section and click "Add alternate question."

C. Enter the following alternate phrases:

i. **`How do I cancel my booking?`**

ii. **`What is the process to cancel a reservation?`**

iii. **`Can I cancel my reservation online?`**

D. Click Save.

Step 3: Add synonyms and colloquial terms.

A. Think about the different ways in which users might refer to key terms. Here are some examples, with the key term first followed by synonyms:

i. Reservation: booking, appointment

ii. Cancel: void, terminate

B. Add phrases and questions that incorporate these synonyms, such as:

i. **`How can I void my reservation?`**

ii. **`Can I terminate my booking?`**

You should aim to cover as many variations as possible to enhance the chatbot's understanding.

C. Click Save.

Step 4: Test the Q&A pair.

A. Select Test in the left sidebar.

B. Test each alternate question to verify that the chatbot responds correctly. Here's an example of user input and the chatbot's expected response:

i. *User*: How do I cancel my booking?

ii. *Chatbot*: You can cancel a reservation by visiting our website or contacting customer support.

C. Ensure that follow-up prompts appear as configured for each alternate question.

Step 5: Add alternate phrasing to a Q&A pair. Now, you should add alternate phrasing to a Q&A pair. Alternate phrasing means asking the same question in different ways that are semantically similar. The initial JSON request would look like this:

```
{
  "question": "How can I cancel a reservation?",
  "top": 10,
  "userId": "Default",
  "isTest": false,
  "context": {}
}
```

Let's see what happens when you send this request:

```
{
  "question": "How do I cancel my booking?",
  "top": 10,
  "userId": "Default",
  "isTest": false,
  "context": {}
}
```

The response should look like this:

```
{
  "answers": [
    {
      "questions": [
        "How can I cancel a reservation?",
        "How do I cancel my booking?",
        "What is the process to cancel a reservation?",
        "Can I cancel my reservation online?"
      ],
      "answer": "You can cancel a reservation by visiting our\n\
      website or contacting customer support.",
      "score": 100.0,
      "context": {
        "prompts": [
          {
            "displayOrder": 0,
            "qnaId": 2,
            "displayText": "Is this for a hotel reservation?"
          },
          {
            "displayOrder": 1,
            "qnaId": 3,
            "displayText": "Is this for a flight reservation?"
          }
        ]
      }
    }
```

```
    ]
  }
```

And with that, you have added alternate phrasing.

Adding chit-chat

Chit-chat allows a bot to be more conversational by making small talk or asking casual questions, both of which enhance user engagement.

You can add chit-chat to a knowledge base in Language Studio with just a few steps.

Step 1: Add chit-chat to your sources.

A. In your project, select "Manage sources" in the left sidebar.

B. Click "Add source."

C. Select "Chitchat" from the list of available source types.

D. Select a personality type that aligns with your bot's intended voice or personality:

   i. Select Professional for a formal and businesslike interaction.

   ii. Select Friendly for a warm and approachable tone.

   iii. Select Witty for clever and humorous interaction.

   iv. Select Enthusiastic for an energetic and lively personality.

   Here are some example responses to a question about the bot's age from each personality type:

   i. Professional: `Age doesn't really apply to me.`

   ii. Friendly: `I don't really have an age.`

   iii. Witty: `I'm age-free.`

   iv. Enthusiastic: `I'm a bot, so I don't have an age.`

E. After choosing the desired personality, click Add. The predefined chit-chat Q&A pairs associated with the selected personality will be integrated into your project.

F. Verify that the chit-chat source appears in the "Manage sources" pane.

Step 2: Edit chit-chat Q&A pairs.

A. Select "Edit knowledge base" in the left sidebar.

B. Find the chit-chat Q&A pairs that have been added for the selected personality. They are usually labeled or grouped accordingly.

C. Click on a chit-chat Q&A pair to edit it, as in this example:

   i. Question: `'When is your birthday?'`

   ii. Original answer (Friendly): `'I don't really have an age.'`

iii. Edited answer: `'I don't have a birthday, but I'm always here to help you!'`

D. After you finish editing, click Save to apply the changes.

E. To view the metadata for the Q&A pairs, select "Show columns" in the toolbar and enable the metadata view."

F. Each Q&A pair will have metadata key-value pairs that help categorize them. Review them.

Step 3: Add custom chit-chat Q&A pairs.

A. In addition to editing the chit-chat Q&A pairs, you can add custom pairs. Click "Add QnA pair."

B. Enter a custom question and answer:

i. Question: **`How are you today?`**

ii. Answer: **`I'm just a bot, but I'm here to help you!`**

C. Add metadata:

i. Click "Add metadata."

ii. Key: enter **`Editorial`**.

iii. Value: enter **`chitchat`**.

D. Click Save to add the custom chit-chat pair.

To prevent conflicts, avoid adding questions that already exist.

Step 4: Test chit-chat integration.

A. Select Test in the left sidebar.

B. Enter casual questions to verify chit-chat responses. Here are some examples:

i. `How are you?`

ii. `Can you tell me a joke?`

iii. `What's your favorite color?`

C. Verify that the chatbot responds appropriately based on the selected personality and any custom edits.

Step 5: Deploy your updated knowledge base.

A. Select Deploy in the left sidebar.

B. Click Deploy to update the deployed model with the new chit-chat capabilities.

C. Wait for the deployment process to complete and confirm success.

Meeting user expectations and avoiding bias. While chit-chat boosts engagement, it can backfire if its tone does not align with user expectations or the context is formal. Inappropriate small talk may be perceived as unprofessional or insensitive, so you should avoid chit-chat in high-stakes scenarios, such as financial or medical inquiries, and allow users to opt out of casual dialog.

Furthermore, deploying chit-chat at scale carries risks of reinforcing biases or displaying cultural insensitivity. Perform regular bias audits on your chit-chat Q&A pairs. The audits should involve diverse stakeholder reviews, and your model should have surface disclaimers that tell users they are interacting with an AI. By being transparent about the bot's persona and having failsafe user opt-outs, you'll help maintain user trust and meet ethical standards.

Best practices for chit-chat integration. To conclude, here are some best practices for chit-chat integration:

Choose the right personality
: Select a chit-chat personality that aligns with your bot's overall tone and the brand image you wish to convey.

Customize responses
: Tailor predefined responses to fit your specific context and user expectations. This will ensure consistency and relevance.

Perform regular updates
: To keep interactions fresh and engaging, periodically review and update chit-chat responses based on user feedback and evolving user needs.

Now that you've implemented chit-chat in your Q&A pairs, we can look at exporting a knowledge base.

Exporting a knowledge base

There are two main ways to export a knowledge base in Azure AI Language through the Language Studio interface or by using the authoring API.

To use Language Studio, sign in and navigate to the "Answer questions" section. Open the custom question answering service, select the specific project you want to export, choose the Export option, and download the exported file.

Alternatively, for a more automated process, you can use the API export functionality. This approach is commonly used for integrating with CI/CD pipelines or to manage backups and migrations across different regions.

You can also export or import only specific sets of Q&A pairs, rather than the entire project.

Chapter Review

In this chapter, we explored advanced NLP techniques using Azure AI Language. You learned how to manage language understanding models by working with intents, utterances, and entities, as well as how to train, evaluate, deploy, test, and optimize those model. You also gained hands-on experience with implementing question-answering solutions. Finally, you learned how to create, train, and test knowledge bases, and how to extend them with advanced capabilities such as multiturn conversations, alternate phrasing, and chit-chat.

To be successful on the exam, you'll need to know how to do the following things that we covered in this chapter:

- Implement and manage a language understanding model with Azure AI Language.
- Implement a question-answering solution with Azure AI Language.

In the next chapter, we'll discuss how to implement knowledge mining and document intelligence solutions.

Chapter Quiz

1. You're designing a chatbot using Azure AI Language to assist with customer inquiries. Which feature should you implement to categorize user inputs such as "Book a flight" or "Cancel my reservation" into specific actions that the chatbot can understand and respond to?

   A. Entities

   B. Intents

   C. Knowledge bases

   D. Language models

2. A company is developing a virtual assistant using Azure's conversational language understanding service. During testing, the development team finds that the assistant often fails to understand user commands that are phrased in unexpected ways. To improve its understanding of varied user inputs, what should the development team focus on?