Building a Conversational Telugu-to-English Translator Alexa Skill

This document outlines the complete process for creating a custom Alexa skill that translates spoken Telugu into English. The skill is built on a serverless architecture using AWS Lambda and Node.js.

Here's a breakdown of how each concept applies:

---

1. Deep Learning: The Foundation

This is the most accurate and fundamental category for the technologies at play. Deep Learning, a subfield of Machine Learning, uses multi-layered neural networks to learn from vast amounts of data. Your skill relies on several distinct Deep Learning models that Amazon and Google have already trained.

- **Automatic Speech Recognition (ASR):** When you speak to your Echo, a Deep Learning model converts the audio waves of your voice into text.
    - **In your skill:** It turns "nee peru enti" from spoken audio into computer-readable text: `"nee peru enti"`.
- **Natural Language Understanding (NLU):** This is another Deep Learning model that takes the recognized text and figures out your *intent*.
    - **In your skill:** It analyzes the text `ask telugu helper to translate the phrase nee peru enti` and correctly determines:
        1. **Intent:** The user wants to trigger the `TranslateIntent`.
        2. **Slot Filling:** The value for the `{teluguSentence}` slot is `"nee peru enti"`.
- **Neural Machine Translation (NMT):** This is the core of the translation itself. The Google Translate API you are calling uses a highly sophisticated Deep Learning model to translate text from a source language to a target language. It understands grammar, context, and syntax in both languages.
- **Text-to-Speech (TTS) / Speech Synthesis:** After your code sends the translated text back, a final Deep Learning model converts that text into natural-sounding, spoken audio.
    - **In your skill:** It takes the text string `"What is your name."` and generates the audio waveform that your Echo plays.

Conclusion: Your skill acts as a conductor for an orchestra of pre-trained Deep Learning models.

2. Generative AI: The Content Creator

This is also a correct, but more specific, label. Generative AI is any AI that *creates new content* rather than just classifying or predicting. While often associated with creative tasks (like ChatGPT writing a story), the technology is broader.

Your skill leverages two forms of task-specific Generative AI:

1. **Translation is Generative:** The Neural Machine Translation model isn't just looking up words in a dictionary. It is *generating* a brand new sentence in the target language that is grammatically correct and contextually equivalent. The output ("What is your name.") is novel content generated based on the input.
2. **Speech Synthesis is Generative:** The Text-to-Speech model is *generating* a completely new audio waveform from scratch based on the text input.

Conclusion: Your project is a practical, applied use of Generative AI, focusing on the specific tasks of text generation (translation) and audio generation (speech).

3. Agentic AI: The Orchestrator

This is a higher-level concept. An "AI Agent" is a system that can perceive its environment, make decisions, and take actions to achieve a specific goal.

- **Your Lambda function itself is NOT an agent.** It is a simple, stateless piece of code. It follows a direct procedure: if it receives a request, it calls an API and returns the result. It doesn't learn, plan, or make autonomous decisions.
- **The entire Alexa system, however, IS an agent.**
  - **Perception:** It perceives the environment through its microphone.
  - **Decision-Making:** It decides which skill to route the request to (the NLU part).
  - **Action:** It takes the action of invoking your Lambda function with the correct data.

Conclusion: You have not built an AI agent. You have built a custom **tool** or **skill** that a larger, more sophisticated AI agent (Alexa) uses to extend its capabilities and achieve a user's goal. This is a fundamental concept in modern AI development—building specialized modules that can be used by a central agentic system.

Part 1: Creating the Voice User Interface (VUI) in the Alexa Developer Console

This section covers the setup of the "frontend"—how users will interact with the skill.

Step 1: Create the New Skill

1. Navigate to the Alexa Developer Console.
2. Click **Create Skill**.
3. **Skill name:** `Telugu English Voice Translator` (or your preferred name).
4. **Primary locale:** `English (US)`.
5. **Choose a model:** `Custom`.
6. **Choose a hosting method:** `Provision your own`.
7. Click **Create skill** and select the `Start from Scratch` template.

Step 2: Set the Skill Invocation Name

The invocation name is the "wake phrase" for your skill.

1. In the left-hand menu, select **Invocations > Skill Invocation Name**.
2. In the text box, enter a simple, two-word name. We found **`telugu helper`** to be effective.
3. Click **Save**.

Step 3: Create a Custom Intent

An intent represents the user's goal. Our primary goal is to translate a sentence.

1. In the left-hand menu, select **Interaction Model > Intents**.
2. Click **Add Intent**.
3. **Name:** `TranslateIntent`. Using this exact name is crucial as our code will reference it.
4. Click **Create custom intent**.

Step 4: Define the Intent Slot

A slot is a variable that captures a piece of information from the user's phrase. We need one slot to capture the sentence they want to translate.

1. Within the `TranslateIntent` page, scroll to **Intent Slots**.
2. **Name:** `teluguSentence`.
3. **Slot Type:** Select `AMAZON.SearchQuery` from the dropdown. This type is optimized for capturing free-form text that isn't from a predefined list.

Step 5: Provide Sample Utterances

These are the phrases that will trigger our `TranslateIntent`. Providing specific phrases is key to reliability.

1. On the `TranslateIntent` page, find the **Sample Utterances** box.
2. Enter several phrases that link to your `teluguSentence` slot. We found the following to be effective:
   - `translate the phrase {teluguSentence}`
   - `ask for the translation of {teluguSentence}`
   - `what is the translation for {teluguSentence}`
   - `translate to english {teluguSentence}`
3. Click **Save**.

Step 6: Build the Interaction Model

This step compiles all your VUI changes into a machine-learning model that Alexa uses to understand user requests.

1. At the top of the page, click the blue **Build Skill** button.
2. Wait for the process to complete. This can take a few minutes. **This step is mandatory after making any changes to intents, slots, or utterances.**

Step 7: Get the Skill ID

We need this ID to link our backend Lambda function to this skill.

1. In the left-hand menu, select **Endpoint**.
2. Your **Skill ID** is displayed at the top (e.g., `amzn1.ask.skill...`). Copy this value to your clipboard.

Part 2: Creating the Backend Logic in AWS

This section covers the setup of the "backend"—the code that will run when the skill is used.

Step 8: Create an IAM Role for the Lambda Function

Our function needs permission to run and write logs for debugging.

1. Navigate to the AWS IAM Console.
2. In the left menu, select **Roles > Create role**.
3. **Trusted entity type:** `AWS service`.
4. **Use case:** `Lambda`. Click **Next**.

5. In the search box, find and select the `AWSLambdaBasicExecutionRole` policy. This grants permission to write to CloudWatch logs.
6. Click **Next**.
7. **Role name:** `AlexaTranslateSkillRole` (or a similar descriptive name).
8. Click **Create role**.

Step 9: Create the AWS Lambda Function

This is the serverless compute service that will run our code.

1. Navigate to the AWS Lambda Console.
2. Click **Create function**.
3. **Option:** `Author from scratch`.
4. **Function name:** `AlexaTranslateSkill`.
5. **Runtime:** `Node.js 18.x`.
6. **Architecture:** `x86_64`.
7. **Permissions:** Expand "Change default execution role." Select `Use an existing role` and choose the `AlexaTranslateSkillRole` you just created.
8. Click **Create function**.

Step 10: Configure the Alexa Skills Kit Trigger

This connects the Alexa skill (frontend) to the Lambda function (backend).

1. In your new Lambda function's page, click **Add trigger**.
2. Select **Alexa Skills Kit** from the dropdown.
3. Scroll down to **Skill ID Verification**.
4. Paste the **Skill ID** you copied in Step 7.
5. Click **Add**.

Part 3: The Code and Deployment

This covers writing the logic and uploading it to Lambda.

Step 11: Set Up the Project Folder (On Your MacBook)

1. Open the Terminal.
2. Create a new project directory: `mkdir alexa-translate-skill && cd alexa-translate-skill`

Step 12: Write the `index.js` Code

Create a file named `index.js` and paste in the final, refined code:

Generated javascript

```
const Alexa = require('ask-sdk-core');
const translate = require('@iamtraction/google-translate');

// Welcomes the user and opens a session
const LaunchRequestHandler = {
    canHandle(handlerInput) {
        return Alexa.getRequestType(handlerInput.requestEnvelope) ===
'LaunchRequest';
    },
    handle(handlerInput) {
        const speakOutput = 'Welcome to the English translator. What would
you like to translate?';
        return handlerInput.responseBuilder
            .speak(speakOutput)
            .reprompt(speakOutput) // Keep the session open
            .getResponse();
    }
};

// Handles the core translation logic
const TranslateIntentHandler = {
    canHandle(handlerInput) {
        return Alexa.getRequestType(handlerInput.requestEnvelope) ===
'IntentRequest'
            && Alexa.getIntentName(handlerInput.requestEnvelope) ===
'TranslateIntent';
    },
    async handle(handlerInput) {
        const textToTranslate =
Alexa.getSlotValue(handlerInput.requestEnvelope, 'teluguSentence');
        const targetLanguage = 'en'; // Hardcoded to English
        const repromptText = ' What else would you like to translate?';
        let speakOutput = '';

        if (textToTranslate) {
            try {
                const result = await translate(textToTranslate, { to:
targetLanguage });
```

```
                speakOutput = result.text;
            } catch (error) {
                console.error(error);
                speakOutput = 'Sorry, I had trouble with that translation.';
            }
        } else {
            speakOutput = 'I did not catch that. Please tell me what to
translate.';
        }

        return handlerInput.responseBuilder
            .speak(speakOutput)
            .reprompt(repromptText) // Keep the session open for the next
command
            .getResponse();
    }
};

// Handles built-in "Help" requests
const HelpIntentHandler = {
    canHandle(handlerInput) {
        return Alexa.getRequestType(handlerInput.requestEnvelope) ===
'IntentRequest'
            && Alexa.getIntentName(handlerInput.requestEnvelope) ===
'AMAZON.HelpIntent';
    },
    handle(handlerInput) {
        const speakOutput = 'You can ask me to translate a phrase, like
"translate the phrase nee peru enti".';
        return handlerInput.responseBuilder
            .speak(speakOutput)
            .reprompt(speakOutput)
            .getResponse();
    }
};

// Handles "Cancel" and "Stop" requests
const CancelAndStopIntentHandler = {
    canHandle(handlerInput) {
        return Alexa.getRequestType(handlerInput.requestEnvelope) ===
'IntentRequest'
            && (Alexa.getIntentName(handlerInput.requestEnvelope) ===
'AMAZON.CancelIntent'
                || Alexa.getIntentName(handlerInput.requestEnvelope) ===
'AMAZON.StopIntent');
    },
    handle(handlerInput) {
        const speakOutput = 'Goodbye!';
        return handlerInput.responseBuilder
            .speak(speakOutput)
            .getResponse(); // No reprompt, so the session closes
    }
};

// Handles session end events
const SessionEndedRequestHandler = {
    canHandle(handlerInput) {
        return Alexa.getRequestType(handlerInput.requestEnvelope) ===
'SessionEndedRequest';
    },
    handle(handlerInput) {
        console.log(`Session ended with reason:
${handlerInput.requestEnvelope.request.reason}`);
        return handlerInput.responseBuilder.getResponse();
    }
};

// A generic error handler for unexpected issues
const ErrorHandler = {
    canHandle() {
        return true;
    },
    handle(handlerInput, error) {
        console.log(`Error handled: ${error.stack}`);
        const speakOutput = 'Sorry, I had trouble doing what you asked.
Please try again.';
        return handlerInput.responseBuilder
            .speak(speakOutput)
```

```
                .reprompt(speakOutput)
                .getResponse();
    }
};

// Register all handlers
exports.handler = Alexa.SkillBuilders.custom()
    .addRequestHandlers(
        LaunchRequestHandler,
        TranslateIntentHandler,
        HelpIntentHandler,
        CancelAndStopIntentHandler,
        SessionEndedRequestHandler)
    .addErrorHandlers(
        ErrorHandler)
    .lambda();
```

Use code with caution.JavaScript

Step 13: Create the `package.json` File

This file manages our project's dependencies. Create a file named `package.json`:

Generated json

```json
{
  "name": "alexa-telugu-translator",
  "version": "1.0.0",
  "description": "Alexa skill to translate Telugu to English.",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Your Name",
  "license": "ISC",
  "dependencies": {
    "ask-sdk-core": "^2.12.1",
    "@iamtraction/google-translate": "^1.1.2"
  }
}
```

Use code with caution.Json

Step 14: Install Dependencies

In your terminal, inside the `alexa-translate-skill` folder, run:

```
npm install
```

This will create the `node_modules` directory with the necessary libraries.

Step 15: Create the Deployment Package

1. Navigate into your `alexa-translate-skill` folder.
2. Select all contents: `index.js`, `package.json`, `package-lock.json`, and the `node_modules` folder.
3. Right-click and select **Compress [X] Items**. This will create `Archive.zip`. **Crucially, you are zipping the contents, not the parent folder.**

Step 16: Upload and Deploy the Code

1. In your `AlexaTranslateSkill` Lambda function, go to the **Code** tab.
2. Click **Upload from > .zip file**.
3. Select the `Archive.zip` file you just created.
4. Click **Save**.

Part 4: Final Configuration and Testing

Step 17: Configure the Lambda Timeout

1. In your Lambda function, go to the **Configuration > General configuration** tab.
2. Click **Edit**.
3. Set the **Timeout** to **10 seconds**. This gives the function enough time to make an internet call to the translation API.
4. Click **Save**.

Step 18: End-to-End Testing

You can now test on your real Echo device or in the Alexa Developer Console **Test** tab.

Conversational Test:

1. **You:** “Alexa, open telugu helper.”
2. **Alexa:** “Welcome to the English translator. What would you like to translate?”
3. **You:** “Nee peru enti.”
4. **Alexa:** “What is your name.”

One-Shot Test:

- **You:** “Alexa, ask telugu helper to translate the phrase nee peru enti.”
- **Alexa:** “What is your name.”