**Implement keyword recognition**

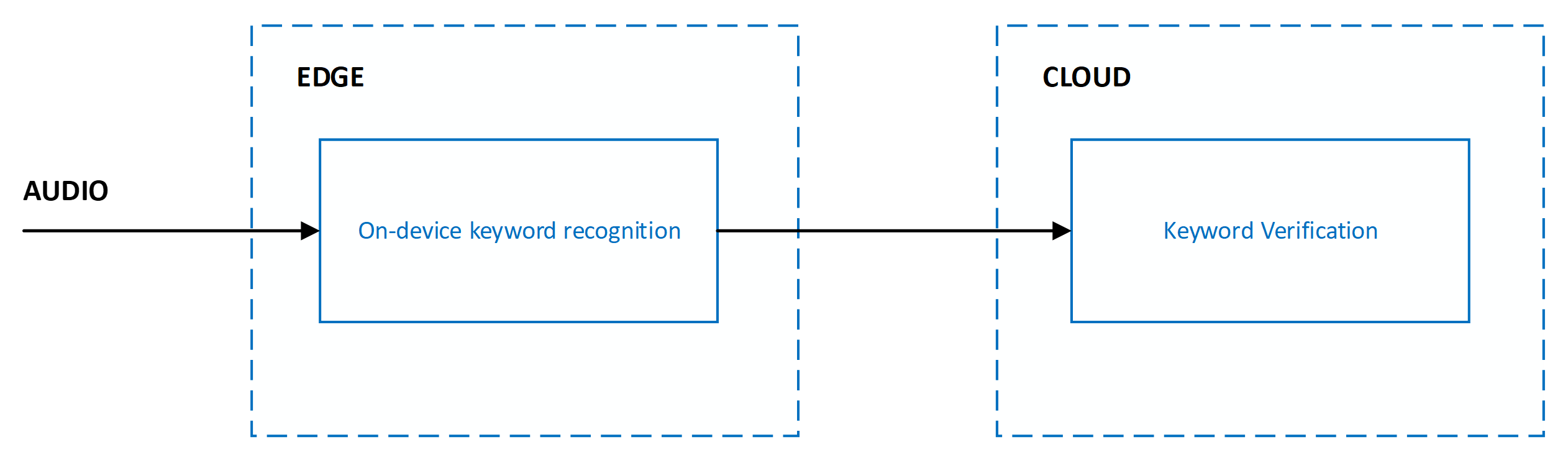
Keyword recognition detects a word or short phrase within a stream of audio. This technique is also referred to as keyword spotting.

The most common use case of keyword recognition is voice activation of virtual assistants. For example, "Hey Cortana" is the keyword for the Cortana assistant. Upon recognition of the keyword, a scenario-specific action is carried out. For virtual assistant scenarios, a common resulting action is speech recognition of audio that follows the keyword.

Generally, virtual assistants are always listening. Keyword recognition acts as a privacy boundary for the user. A keyword requirement acts as a gate that prevents unrelated user audio from crossing the local device to the cloud.

To balance accuracy, latency, and computational complexity, keyword recognition is implemented as a multistage system. For all stages beyond the first, audio is only processed if the stage prior to it recognizes the keyword of interest.

The current system is designed with multiple stages that span the edge and cloud:



Accuracy of keyword recognition is measured via the following metrics:

* **Correct accept rate**: Measures the system's ability to recognize the keyword spoken by a user. The correct accept rate is also known as the true positive rate.
* **False accept rate**: Measures the system's ability to filter out audio that isn't the keyword spoken by a user. The false accept rate is also known as the false positive rate.

The goal is to maximize the correct accept rate while minimizing the false accept rate. The current system is designed to detect a keyword or phrase preceded by a short amount of silence. Detecting a keyword in the middle of a sentence or utterance isn't supported.

**Custom keyword for on-device models**

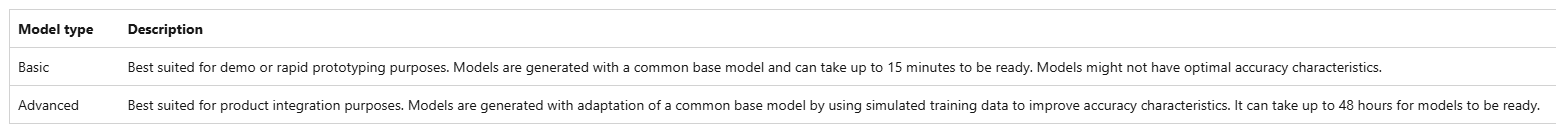
With the [Custom Keyword portal on Speech Studio](https://speech.microsoft.com/customkeyword), you can generate keyword recognition models that execute at the edge by specifying any word or short phrase. You can further personalize your keyword model by choosing the right pronunciations.

**Pricing**

There's no cost to use custom keyword to generate models, including both Basic and Advanced models. There's also no cost to run models on-device with the Speech SDK when used with other Speech service features such as speech to text.

**Types of models**

You can use custom keyword to generate two types of on-device models for any keyword.



Neither model type requires you to upload training data. Custom keyword fully handles data generation and model training.

**Pronunciations**

When you create a new model, custom keyword automatically generates possible pronunciations of the provided keyword. You can listen to each pronunciation and choose all variations that closely represent the way you expect users to say the keyword. All other pronunciations shouldn't be selected.

It's important to be deliberate about the pronunciations you select to ensure the best accuracy characteristics. For example, if you choose more pronunciations than you need, you might get higher false accept rates. If you choose too few pronunciations, where not all expected variations are covered, you might get lower correct accept rates.

**Test models**

After custom keyword generates on-device models, the models can be tested directly on the portal. You can use the portal to speak directly into your browser and get keyword recognition results.

**Keyword verification**

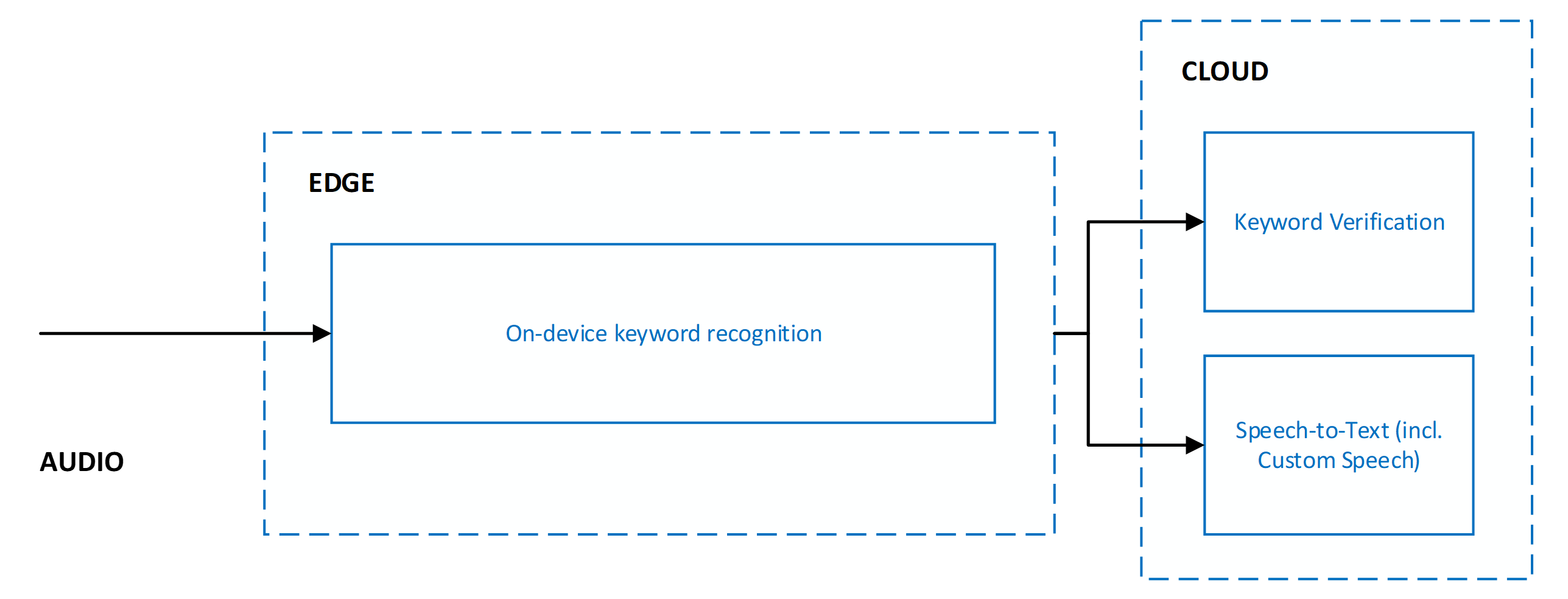
Keyword verification is a cloud service that reduces the effect of false accepts from on-device models with robust models running on Azure. Tuning or training isn't required for keyword verification to work with your keyword. Incremental model updates are continually deployed to the service to improve accuracy and latency and are transparent to client applications.

**Pricing**

Keyword verification is always used in combination with speech to text. There's no cost to use keyword verification beyond the cost of speech to text.

**Keyword verification and speech to text**

When keyword verification is used, it's always in combination with speech to text. Both services run in parallel, which means audio is sent to both services for simultaneous processing.

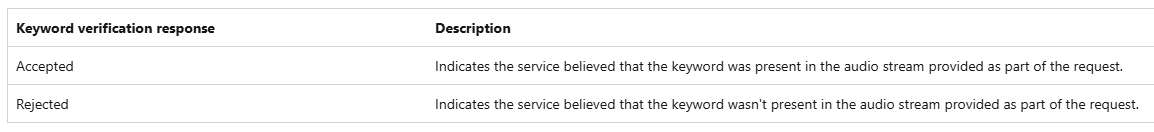


Running keyword verification and speech to text in parallel yields the following benefits:

* **No other latency on speech to text results**: Parallel execution means that keyword verification adds no latency. The client receives speech to text results as quickly. If keyword verification determines the keyword wasn't present in the audio, speech to text processing is terminated. This action protects against unnecessary speech to text processing. Network and cloud model processing increases the user-perceived latency of voice activation. For more information, see [Recommendations and guidelines](https://learn.microsoft.com/en-us/azure/ai-services/speech-service/keyword-recognition-guidelines).
* **Forced keyword prefix in speech to text results**: Speech to text processing ensures that the results sent to the client are prefixed with the keyword. This behavior allows for increased accuracy in the speech to text results for speech that follows the keyword.
* **Increased speech to text timeout**: Because of the expected presence of the keyword at the beginning of audio, speech to text allows for a longer pause of up to five seconds after the keyword before it determines the end of speech and terminates speech to text processing. This behavior ensures that the user experience is correctly handled for staged commands (*<keyword> <pause> <command>*) and chained commands (*<keyword> <command>*).

**Keyword verification responses and latency considerations**

For each request to the service, keyword verification returns one of two responses: accepted or rejected. The processing latency varies depending on the length of the keyword and the length of the audio segment expected to contain the keyword. Processing latency doesn't include network cost between the client and Speech services.



Rejected cases often yield higher latencies as the service processes more audio than accepted cases. By default, keyword verification processes a maximum of two seconds of audio to search for the keyword. If the keyword isn't found in two seconds, the service times out and signals a rejected response to the client.

**Use keyword verification with on-device models from custom keyword**

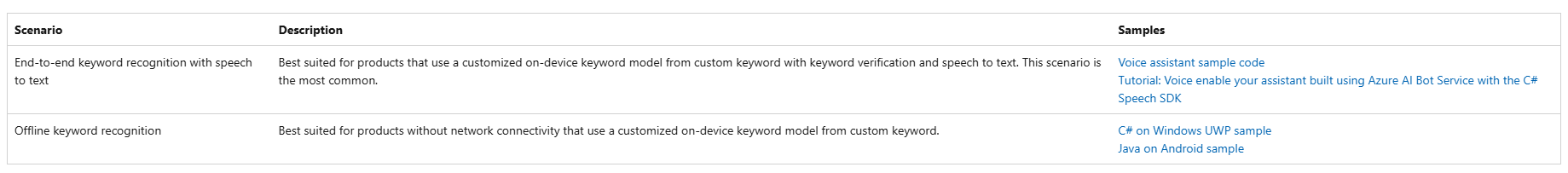
The Speech SDK enables seamless use of on-device models generated by using custom keyword with keyword verification and speech to text. It transparently handles:

* Audio gating to keyword verification and speech recognition based on the outcome of an on-device model.
* Communicating the keyword to keyword verification.
* Communicating any more metadata to the cloud for orchestrating the end-to-end scenario.

You don't need to explicitly specify any configuration parameters. All necessary information is automatically extracted from the on-device model generated by custom keyword.

**Speech SDK integration and scenarios**

The Speech SDK enables easy use of personalized on-device keyword recognition models generated with custom keyword and keyword verification. To ensure that your product needs can be met, the SDK supports the following two scenarios:



**Create a custom keyword**

**Prerequisites**

* An Azure subscription. You can [create one for free](https://azure.microsoft.com/free/cognitive-services).
* [Create a Speech resource](https://portal.azure.com/#create/Microsoft.CognitiveServicesSpeechServices) in the Azure portal.
* Get the Speech resource key and region. After your Speech resource is deployed, select **Go to resource** to view and manage keys.

**Create a keyword in Speech Studio**

1. Go to the [Speech Studio](https://aka.ms/sdsdk-speechportal) and **Sign in**. If you don't have a speech subscription, go to [**Create Speech Services**](https://portal.azure.com/#create/Microsoft.CognitiveServicesSpeechServices).

A screenshot of a computer

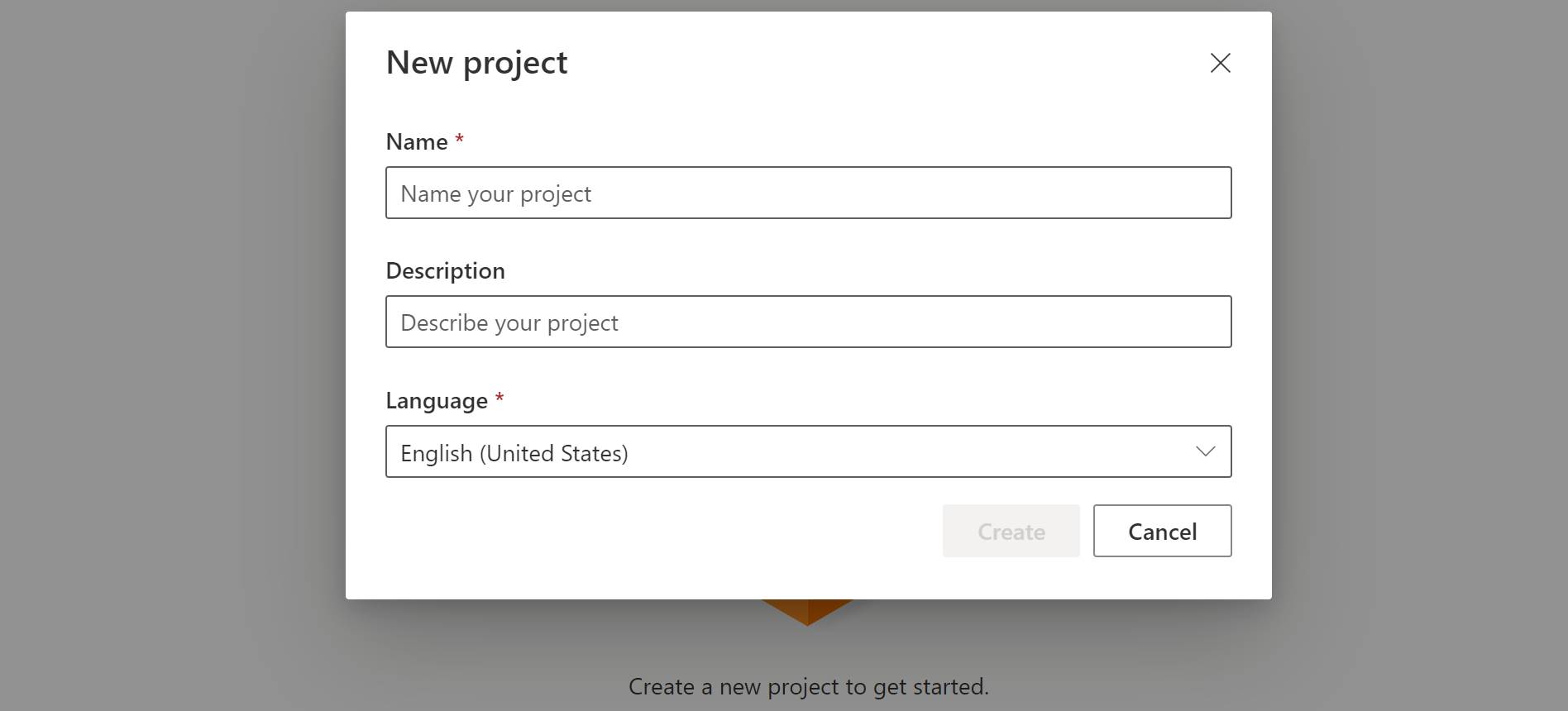
Description automatically generated

1. On the [Custom Keyword](https://aka.ms/sdsdk-wakewordportal) page, select **Create a new project**.

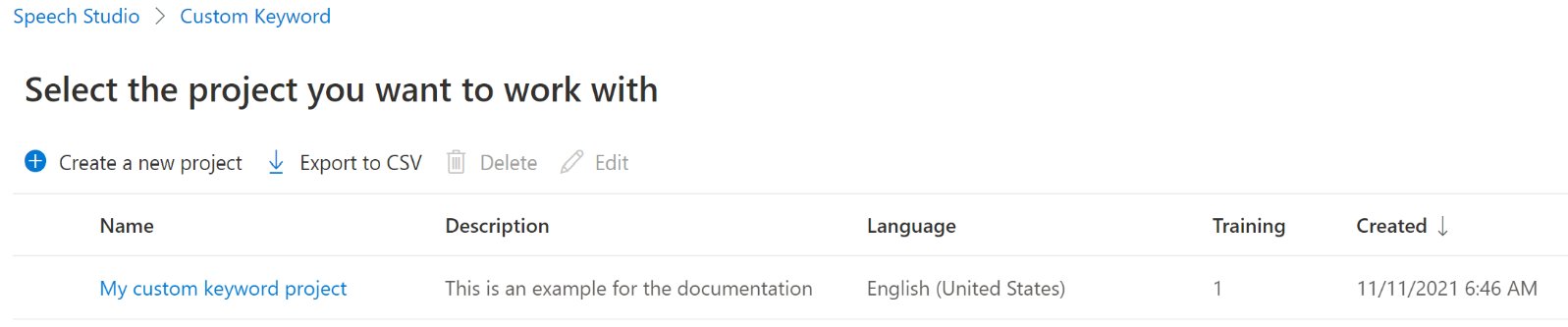
A screenshot of a computer

Description automatically generated

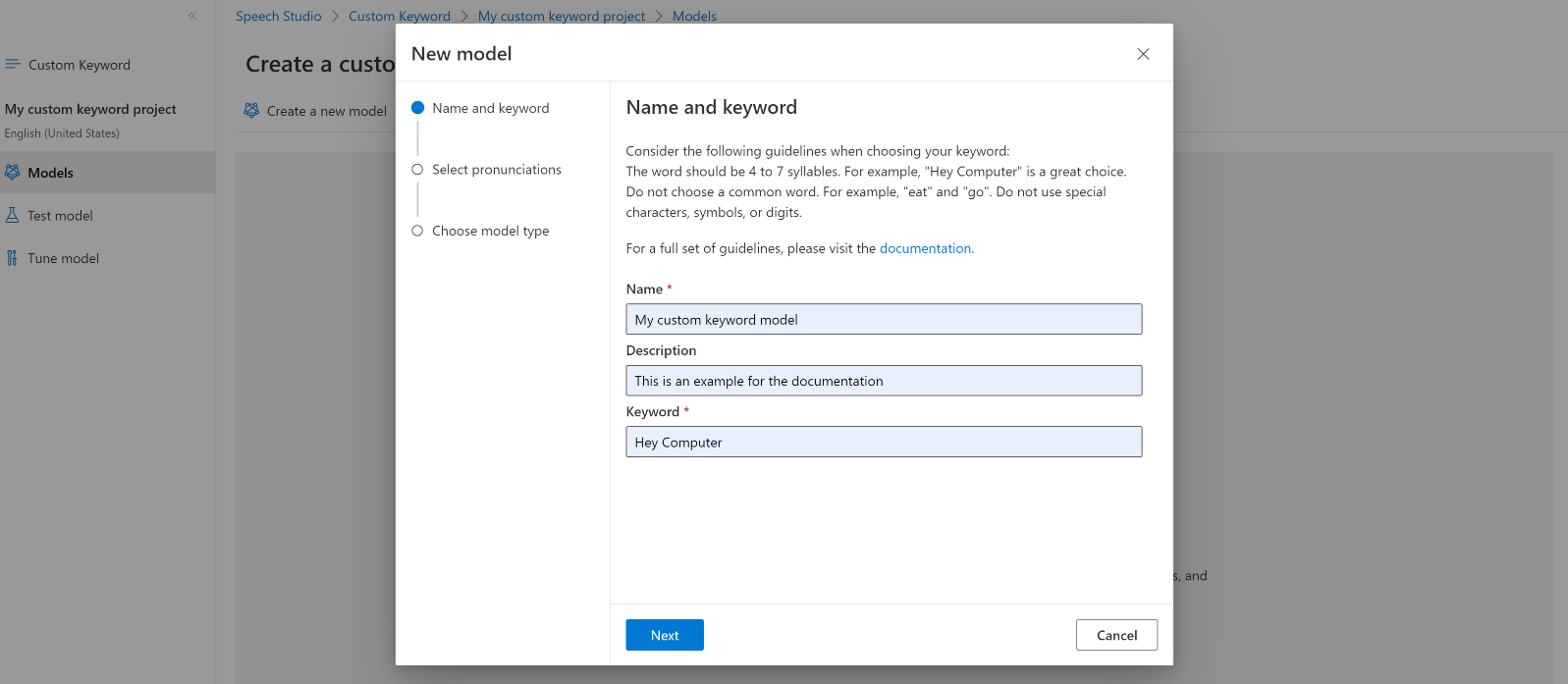
1. Enter a **Name**, **Description**, and **Language** for your custom keyword project. You can only choose one language per project, and support is currently limited to English (United States) and Chinese (Mandarin, Simplified).



1. Select your project's name from the list.



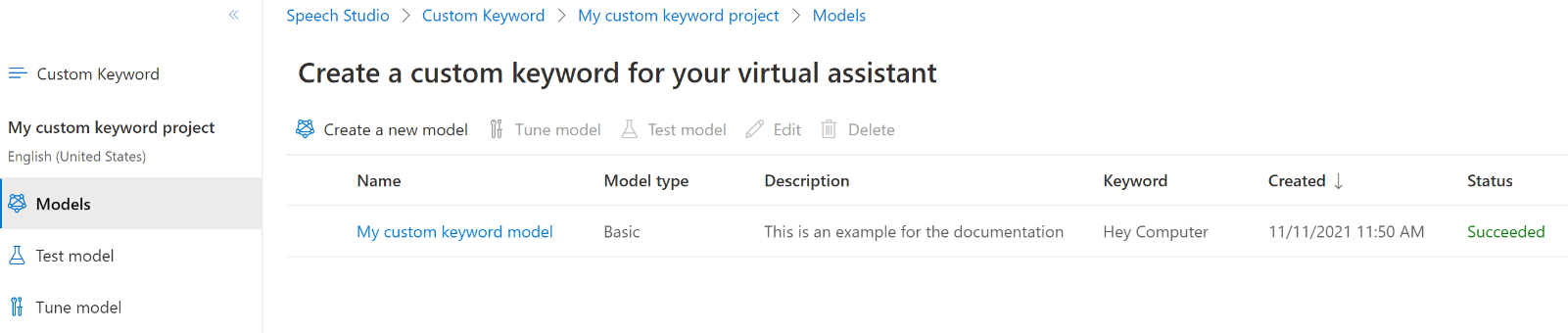
1. To create a custom keyword for your virtual assistant, select **Create a new model**.
2. Enter a **Name** for the model, **Description**, and **Keyword** of your choice, then select **Next**. See the [guidelines](https://learn.microsoft.com/en-us/azure/ai-services/speech-service/keyword-recognition-guidelines#choosing-an-effective-keyword) on choosing an effective keyword.



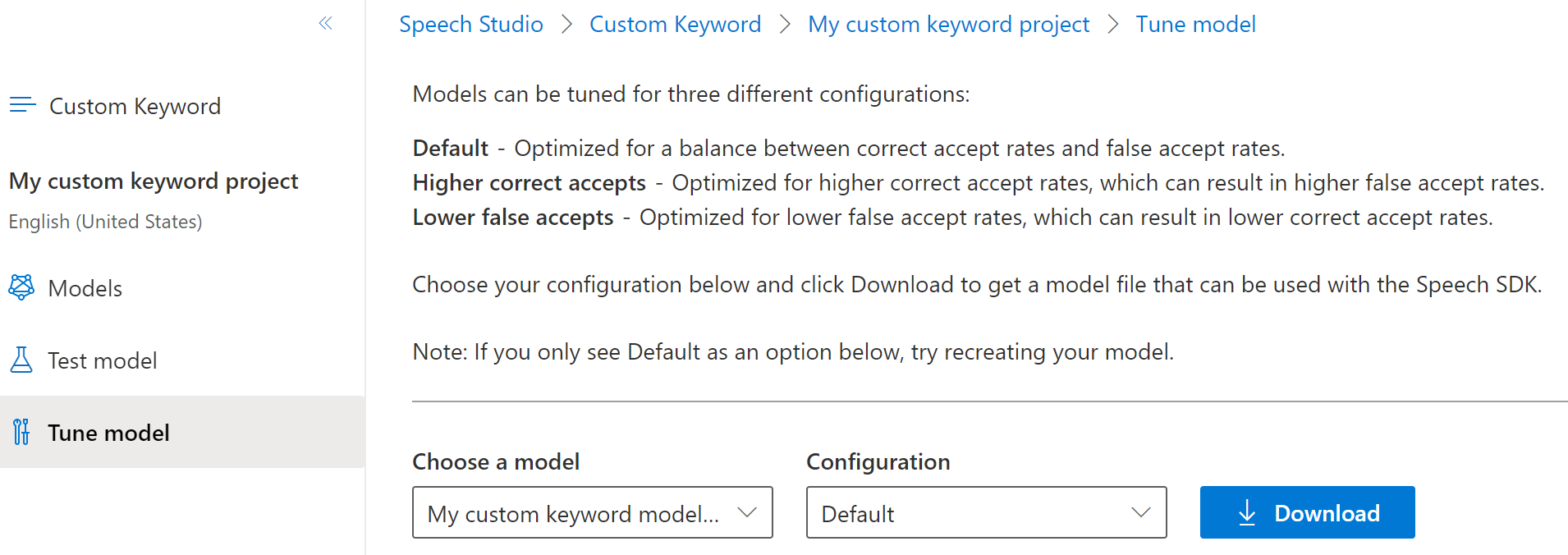
1. The portal creates candidate pronunciations for your keyword. Listen to each candidate by selecting the play buttons and remove the checks next to any pronunciations that are incorrect.Select all pronunciations that correspond to how you expect your users to say the keyword and then select **Next** to begin generating the keyword model.



1. Select a model type, then select **Create**. You can view a list of regions that support the **Advanced** model type in the [Keyword recognition region support](https://learn.microsoft.com/en-us/azure/ai-services/speech-service/regions#speech-service) documentation.
2. It might take up to 30 minutes for the model to be generated. The keyword list changes from **Processing** to **Succeeded** when the model is complete.



1. From the collapsible menu on the left, select **Tune** for options to tune and download your model. The downloaded file is a .zip archive. Extract the archive, and you see a file with the .table extension. You use the .table file with the SDK, so make sure to note its path.



**Use a keyword model with the Speech SDK**

First, load your keyword model file using the **FromFile**() static function, which returns a **KeywordRecognitionModel**. Use the path to the .table file you downloaded from Speech Studio. Additionally, you create an **AudioConfig** using the default microphone, then instantiate a new KeywordRecognizer using the audio configuration.

A screenshot of a computer program

Description automatically generated

Next, running keyword recognition is done with one call to **RecognizeOnceAsync**() by passing your model object. This method starts a keyword recognition session that lasts until the keyword is recognized. Thus, you generally use this design pattern in multi-threaded applications, or in use cases where you might be waiting for a wake-word indefinitely.

A close-up of a white box

Description automatically generated

**Continuous recognition**

Other classes in the Speech SDK support continuous recognition (for both speech and intent recognition) with keyword recognition. The SDK allows you to use the same code you would normally use for continuous recognition, with the ability to reference a .table file for your keyword model.

For speech to text, follow the same design pattern shown in the [recognize speech guide](https://learn.microsoft.com/en-us/azure/ai-services/speech-service/how-to-recognize-speech?pivots=programming-language-csharp#continuous-recognition) to set up continuous recognition. Then, replace the call to recognizer.StartContinuousRecognitionAsync() with recognizer.StartKeywordRecognitionAsync(KeywordRecognitionModel), and pass your KeywordRecognitionModel object. To stop continuous recognition with keyword recognition, use recognizer.StopKeywordRecognitionAsync() instead of recognizer.StopContinuousRecognitionAsync().

Intent recognition uses an identical pattern with the [StartKeywordRecognitionAsync](https://learn.microsoft.com/en-us/dotnet/api/microsoft.cognitiveservices.speech.intent.intentrecognizer.startkeywordrecognitionasync#Microsoft_CognitiveServices_Speech_Intent_IntentRecognizer_StartKeywordRecognitionAsync_Microsoft_CognitiveServices_Speech_KeywordRecognitionModel_) and [StopKeywordRecognitionAsync](https://learn.microsoft.com/en-us/dotnet/api/microsoft.cognitiveservices.speech.intent.intentrecognizer.stopkeywordrecognitionasync#Microsoft_CognitiveServices_Speech_Intent_IntentRecognizer_StopKeywordRecognitionAsync) functions.

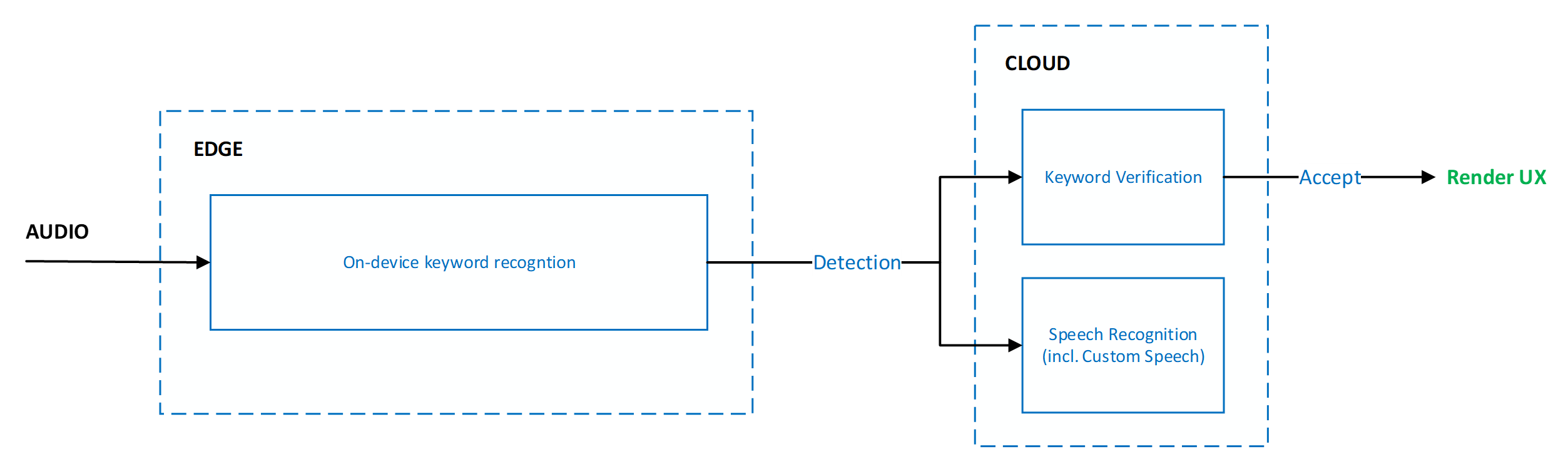
**Choosing an effective keyword**

Creating an effective keyword is vital to ensuring your product responds consistently and accurately. Consider the following guidelines when you choose a keyword.

* It should take no longer than two seconds to say.
* Words of 4 to 7 syllables work best. For example, "Hey, Computer" is a good keyword. Just "Hey" is a poor one.
* Keywords should follow common pronunciation rules specific to the native language of your end-users.
* A unique or even a made-up word that follows common pronunciation rules might reduce false positives. For example, "computerama" might be a good keyword.
* Don't choose a common word. For example, "eat" and "go" are words that people say frequently in ordinary conversation. They might lead to higher than desired false accept rates for your product.
* Avoid using a keyword that might have alternative pronunciations. Users would have to know the "right" pronunciation to get their product to voice activate. For example, "509" can be pronounced "five zero nine," "five oh nine," or "five hundred and nine." "R.E.I." can be pronounced "r-e-i" or "ray." "Live" can be pronounced "/līv/" or "/liv/".
* Don't use special characters, symbols, or digits. For example, "Go#" and "20 + cats" could be problematic keywords. However, "go sharp" or "twenty plus cats" might work. You can still use the symbols in your branding and use marketing and documentation to reinforce the proper pronunciation.

**Users experience recommendations with keyword verification**

With a multi-stage keyword recognition scenario where [keyword verification](https://learn.microsoft.com/en-us/azure/ai-services/speech-service/keyword-recognition-overview#keyword-verification) is used, applications can choose when the end-user is notified of a keyword detection. The recommendation for rendering any visual or audible indicator is to rely upon on responses from the keyword verification service:



This ensures the optimal experience in terms of accuracy to minimize the user-perceived effect of false accepts but incurs extra latency.

For applications that require latency optimization, applications can provide light and unobtrusive indicators to the end-user based on the on-device keyword recognition. For example, lighting an LED pattern or pulsing an icon. The indicators can continue to exist if keyword verification accepts a keyword, or can be dismissed if the keyword is rejected.

