Chatbots that work

Conversational AI

Andrew R. Freed

MEAP

MANNING

MEAP Edition
Manning Early Access Program
Conversational AI
Chatbots that work
Version 5

For more information on this and other Manning titles go to
manning.com

welcome

Dear reader,

Thank you for purchasing the MEAP for *Conversational AI*.

I've been privileged enough to read a lot of useful content in my career as a software engineer. In many ways I feel that I'm standing on the shoulders of giants. It's important to me to give back by producing content of my own, especially through this book. I hope you step on my shoulders as you learn from this book!

Before you start this book, you should be comfortable in reading and writing decision trees and process flows. You should be comfortable with branching control logic in the form of "if statements".

Virtual assistants use machine learning under the covers; but you do not need a PhD in mathematics to understand this book. I take great pains to make machine learning approachable.

It takes a dream team to build an effective virtual assistant. In this book you'll learn why it takes several different players to build a virtual assistant and you'll see what each member of the team needs to do. Even if you work in a "silo", you'll get a greater appreciation for what your teammates do.

I've had a lot of fun building virtual assistants in my career and I'm excited to share my experience with you. I encourage you to post any feedback or questions (good or bad!) in the liveBook Discussion forum. Your feedback will help me write the best possible book for you!

Andrew R. Freed

brief contents

1

Introduction to virtual assistants

This chapter covers:

- Listing the types of virtual assistants and their platforms
- Classifying a virtual assistant as a conversational assistant, a command interpreter, or an event classifier
- Recognizing virtual assistants that you already interact with
- Differentiating between questions that have a simple response versus those that require a process flow
- Describing what happens when you mark an email as spam and when you block an email sender.

Virtual assistants are an exciting new technology being used in an increasing number of places and use cases. The odds are good that you have interacted with a virtual assistant. From the assistants on our phones (Hey Siri!), to automated customer service agents, to email filtering systems, virtual assistant technology is widespread and growing in use.

In this book, you will learn about how and why virtual assistants' work. You will learn the types of use cases where virtual assistant technology is appropriate and how to effectively apply the technology. You will learn how to develop, train, test, and improve your virtual assistant. Finally, you will learn about advanced topics including enabling a voice channel for your virtual assistant and how to deeply analyze your virtual assistant's performance.

Who is this book for? Who is this book NOT for?

This book is written for someone interested in developing virtual assistants. We will start with broad coverage of multiple virtual assistant types and how they are used, and then will take a deep dive into all aspects of creating a virtual assistant including design, development, training, testing, and measurement.

If you have already developed several virtual assistants, you'll probably want to skip ahead to specific chapters later in the book. If you are not a developer, you can read the first couple of chapters and skim or skip the rest.

1.1 Introduction to Virtual Assistants and Their Platforms

Why are virtual assistants popular? Let's examine just one industry that frequently uses virtual assistants – the customer service industry. A new call or case at a customer service center averages between 8 and 45 minutes depending on the product or service type. Customer service call centers spend up to $4000 for every agent they hire - and even more money in training costs - while experiencing 30-45% employee turnover. This leads to an estimated annual loss of 62 billion dollars' worth of sales annually in the US alone[1]. Virtual assistants are here to help with these problems and more.

You've probably had several recent interactions with different kinds of virtual assistants:

- Retailers use virtual assistants to power chat interfaces for customer service and guided buying.
- Smartphones and connected homes that are controlled by voice (e.g. "Alexa, turn on the light!").
- Email software that automatically sorts your mail into folders like Important and Spam.

Virtual assistants are pervasive and are used in a wide variety of ways. How many of the virtual assistants in Table 1 have you interacted with? The rest of this section will dive into the mainstay platforms and virtual assistant types in this ecosystem.

[1]Source: IBM Customer Care Sales Deck – 2019

Table 1 Examples of virtual assistants and their platforms

Virtual Assistant Type	Uses	Skills you'll need to build one	Technology focus	Example Platforms
Conversational Assistant (sometimes called "chatbot")	• Customer service • Guided buying experience • New employee training	• Designing and coding process flows with one or many steps • Using conversational state to determine the next step • Writing dialog • Classifying utterances into intents • Extracting entities from utterances to support the intent • Writing code to call external APIs	Conversation and dialog flow	• IBM Watson Assistant[2] • Microsoft Azure Bot Service[3] • Rasa[4]
Command Interpreter	• Natural language or voice interface to devices	• Classifying statements into commands • Extracting supporting parameters from a command statement • Writing code to call external APIs	Classification and calling APIs	• Apple Siri[5] • Amazon Alexa[6]
Event Classifier	• Sorting email into folders • Routing messages (like emails) to an appropriate handler	• Classifying messages based on message content and metadata • Extracting many entities that support or augment the classification	Classification and entity identification	• Google Gmail[7] • Natural Language Understanding services

There are a wide variety of platforms that help you build virtual assistants. Most of the platforms are usable in a variety of virtual assistant scenarios. There are a few things you should consider as you choose a platform:

- **Ease of use**: Some platforms are intended for business users and some for software developers
- **APIs and integration**: Does the platform expose APIs, or have pre-built integrations

[2]https://www.ibm.com/cloud/watson-assistant/
[3]https://azure.microsoft.com/en-us/services/bot-service/
4https://rasa.com/
5https://www.apple.com/siri/
6https://developer.amazon.com/en-US/alexa
7https://www.google.com/gmail/

to 3rd party interfaces and tools? A virtual assistant is usually integrated into a larger solution

- **Runtime environment**: Some platforms run only in the cloud, some only on-premise, and some both.
- **Open or closed source**: Many virtual assistant platforms do not make their source code available.

Let's look at a couple of assistants in more detail.

1.1.1 Types of virtual assistants

Virtual assistants come in many shapes and sizes. When you hear "virtual assistant" you may think "chat bot", but there are multiple applications of virtual assistant technology. Virtual assistants can be used to have a full conversation dialog with a user, use dialog to execute a single command, or work behind the scenes without any dialog at all.

There are three categories of virtual assistants:

- **Conversational assistants** are full systems using full conversational dialog to accomplish one or more tasks. (You may be used to calling these "chat bots")
- **Command interpreters** use enough dialog to interpret and execute a single command. (You probably use one of these on your phone)
- **Event classifiers** don't use dialog at all, they just read a message (like an email) and perform an action based on the type of message.

When a user sends textual input to a virtual assistant that input is generally understood to contain an intent (what the user wants to achieve) and optionally some parameters supporting that intent (we'll call these parameters "entities").

A generalized virtual assistant architecture is shown in Figure 1.

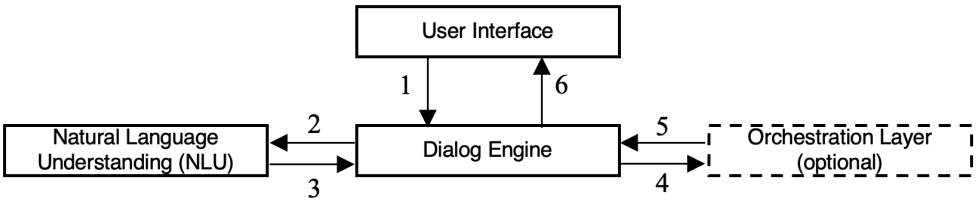


Figure 1 Generalized virtual assistant architecture and control flow

This architecture includes four primary components.

- **Interface**: The way end-users interact with the assistant. This can be a textual or voice interface and is the only part of the virtual assistant visible to the user.
- **Dialog Engine**: Manages dialog state and coordinates building the assistant's response to the user.
- **Natural Language Understanding (NLU)**: This component is invoked by the dialog engine to extract meaning from a user's natural language input. The NLU generally extracts an "intent" as well as other information supporting an intent.

- **Orchestrator**: (Optional) Coordinates calls to APIs to drive business processes and provide dynamic response data.

Let's examine how these components can work in a single turn of dialog. When I set an alarm on my phone, a command interpreter is my virtual assistant, and it executes a flow like the following:

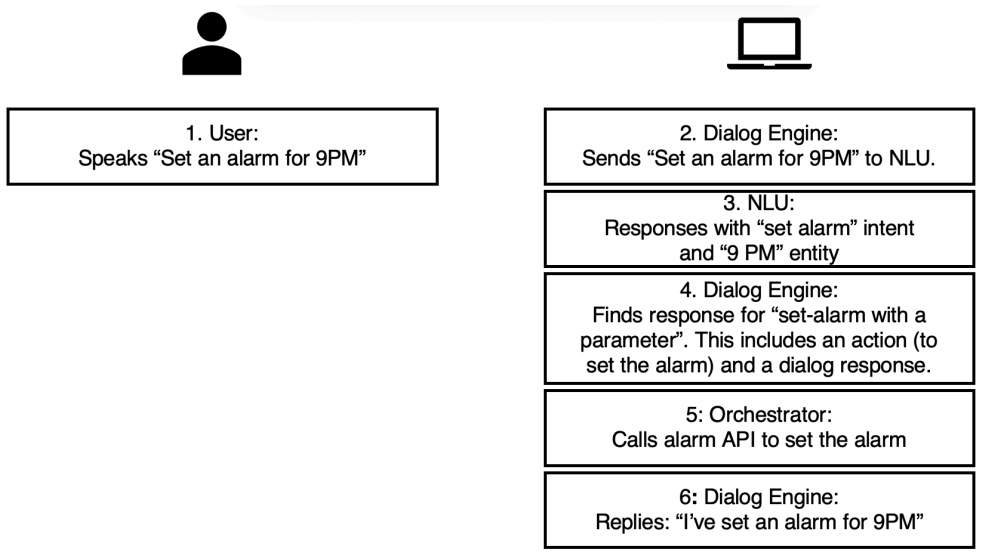


Figure 2 Breaking down a conversational interaction with a virtual assistant

In Figure 2 we see how the components interact in a conversation with a single turn.

- The user starts the conversation asking for something: "Set an alarm for 9PM."
- The user interface passes this text to the Dialog Engine which first asks the classifier to find the intent. Natural language understanding identifies a "set alarm" intent.
- The natural language understanding also detects an entity – the phrase "9 PM" represents a time.
- The dialog engine looks up the appropriate response for the "set alarm" intent when a time parameter is present. The response has two parts: an action (performed by the orchestrator) and a textual response.
- The orchestrator calls an API exposed by an alarm service to set an alarm for 9PM.
- The dialog engine responds to the user via the user interface: "I’ve set an alarm for 9PM.”

Terminology alert!

The terms in this section are meant to be generic across virtual assistant providers. Depending on your provider, you may see slightly different terminology.

Interface: Many virtual assistant providers do not include a user interface, exposing their capabilities only through Application Programming Interfaces (APIs). Your provider might refer to Integrations, Connectors, or Channels, to supply a user interface.

Natural Language Understanding (NLU): Nearly every virtual assistant includes a natural language understanding component. Sometimes this component is referred to as Natural Language Understanding (NLU) or Cognitive Language Understanding. The NLU component usually includes a classifier and an entity (or parameter) detector.

Dialog Engine: This component has the least consistent terminology across virtual assistant platforms. Each platform lets you associate responses to conversational conditions. Some platforms expose this only as code and some through a visual editor.

Orchestration: Virtual assistants sometimes let you code backend orchestration directly into the assistant, commonly referred to as webhooks. You can generally write an orchestrator that interfaces with a virtual assistant through an API. Finally, many platforms have built-in orchestration interfaces to third parties; these may be referred to as integrations or connectors. Sometimes the platform will use the same terminology (i.e. "integrations") to refer to front-end and back-end components!

CONVERSATIONAL ASSISTANT

Conversational assistants are the type of agent that most frequently comes to mind when you hear the words "virtual assistant". A conversational assistant has a conversational interface and services a variety of requests. Those requests may be satisfied with a simple questionand-answer format or may require an associated conversational flow.

In May 2020 many private and public entities built conversational assistants to handle questions related to the novel coronavirus. These assistants fielded most responses via a simple question and response format. However, many of these assistants also included a "symptom checker" that walked a user through a diagnostic triage process. These assistants were critical in quickly disseminating the latest expert and official advice to constituent populations.

Virtual assistants can be trained to answer a wide variety of questions. The answers returned by the assistant can be static text or contain information from external sources. In Figure 3 we see a user greeted by a virtual assistant and asking a simple question: "What is the coronavirus?". The virtual assistant returns with an answer: the description of the coronavirus. At this point the user's request is considered complete: a question has been asked and an answer provided.

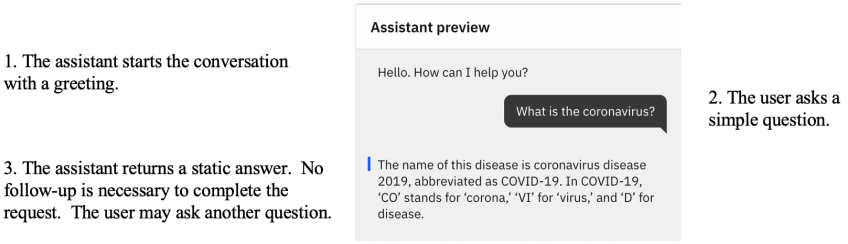


Figure 3 A simple question and answer scenario with a static answer. Answer from Center for Disease Control at https://www.cdc.gov/coronavirus/2019-ncov/faq.html.

Figure 3 demonstrated a simple question and answer response. The question "What is the coronavirus" has a simple response, in this case a definition of the coronavirus. Everyone who asks that question gets the same answer, and the answer rarely changes. This question was answered using just the interface, NLU, and dialog engine. We did not need to make an API call, so no orchestrator was required.

Many coronavirus assistants also answered questions like "How many COVID cases are there?". In this case, the answer is changing daily. The number of cases cannot be hardcoded into the dialog engine (you would have to update it every day!). The answer is still a simple response ("There have been *x* cases") but this will require an orchestrator to query a case count API to return a complete answer.

Virtual Assistants in Action!

The Center for Disease Control has its own virtual assistant for people to check their symptoms to see if they may have the coronavirus or if they should get tested. Try it out at https://www.cdc.gov/coronavirus/2019-ncov/symptoms-testing/symptoms.html

In Figure 4 we see the user has asked a more complex question: "Do I have the coronavirus?" This is a difficult question to answer. While we could return a static result, it would be paragraphs or pages in length and would require a lot of work on the user's part to interpret. To deliver a satisfactory answer we need additional information from the user. The assistant asks a series of follow-up questions, in this case asking about fever and itchy eyes. After a series of back-and-forth messages, the assistant will make a final recommendation including possibilities like self-isolation or seeking immediate medical assistance.

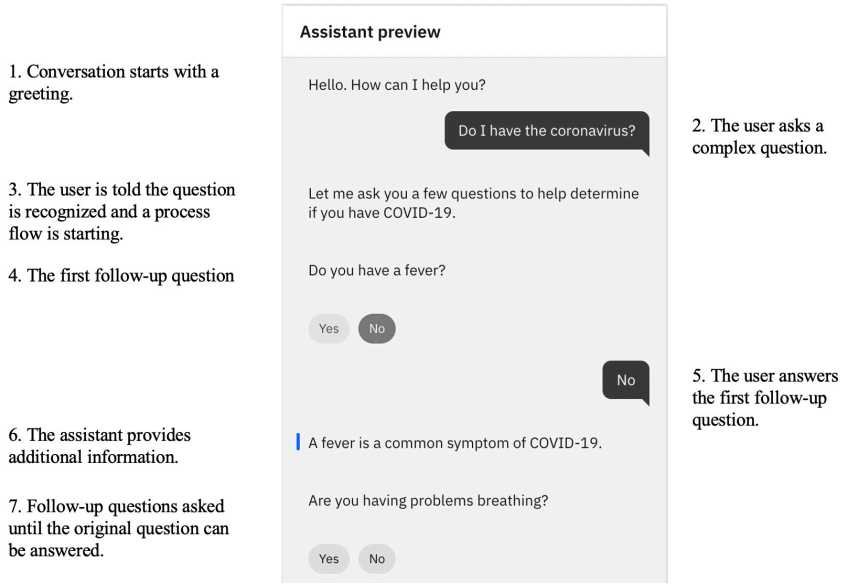


Figure 4 A new conversation where the user asks a question that requires a process flow before providing the final answer.

A virtual assistant is frequently expected to service multiple requests in a single conversational session whether these requests follow a simple question-and-answer format or whether they involve more detailed process flows. When designing a process flow, be sure to sketch it out in a flow chart as in Figure 5.

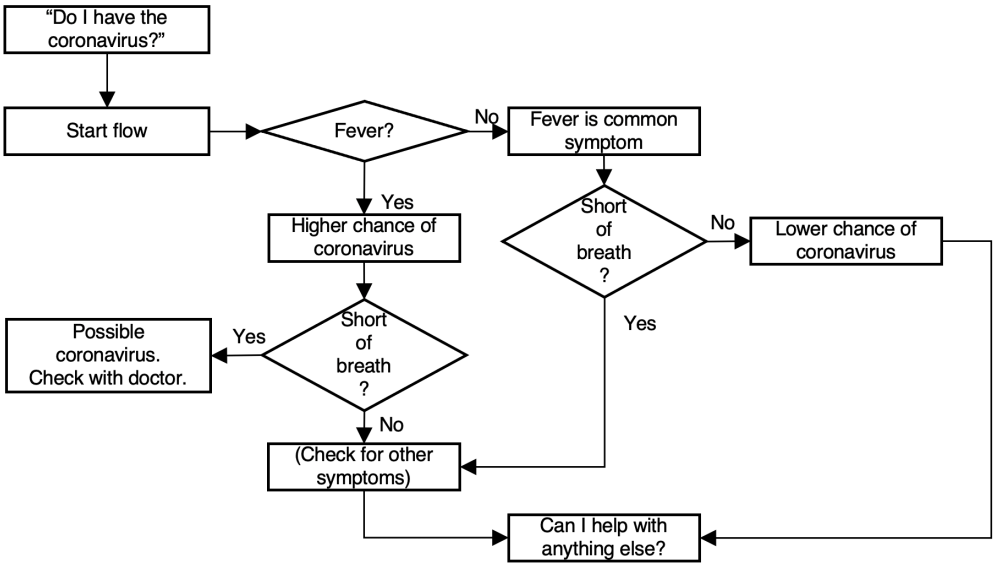


Figure 5 Exemplary process flow for a coronavirus assistant inferred from the table at https://www.cdc.gov/coronavirus/2019-ncov/community/schools-childcare/symptom-screening.html. DO NOT TAKE MEDICAL ADVICE FROM THIS BOOK.

A virtual assistant requires the most up-front design work of the three categories due to the amount of work required in gathering and writing the responses. I will describe those design aspects in Chapter 3.

COMMAND INTERPRETER

A simpler use of virtual assistant technology is in the Command Interpreter pattern. This is prevalent in "smart" devices: your phone, your TV remote, your appliances. The command interpreter deals with a limited vocabulary - enough to invoke the commands it supports - and the interaction is finished as soon as the information needed to execute the task is collected. A smart television remote is tuned to recognize words like "channel" and "volume" but won't recognize a command like "Set air temperature to 76 degrees". Table 2 shows example command interpreters and commands that are in their vocabulary.

Table 2 Example command interpreters and supported commands

Command Interpreter Type	Example commands
Smartphone	"Set an alarm for 9PM" "Call Mom" "Open Facebook"
Voice-powered television remote	"Increase volume" "Turn to channel 3"
Smart home controller	"Turn on the light" "Set air temperature to 76 degrees"

My favorite command on my phone is the "set alarm" command which has two parameter slots: the alarm time and the alarm reason. On my phone, the alarm time is a required parameter and the alarm reason is optional. In the first example below, I've mapped out the simplest "set alarm" invocation: "Set an alarm". The interpreter knows that setting an alarm without an alarm time is impossible, so it prompts me for the alarm time.

The command interpreters you are most familiar with require activation either via a physical button or a "wake word". Several wake words you may be familiar with include "Hey Siri", "Ok Google", and "Alexa". Devices that wake via a button don't start listening until you press that button. Devices that are always listening don't jump into action until you use a wake word.

Figure 6 and Figure 7 demonstrate how several command interpreters parse your input into an actionable command. Try experimenting with your devices to see how many different ways you can execute a command, either by changing the words ("create alarm" vs "set an alarm") or changing the order of the information given.

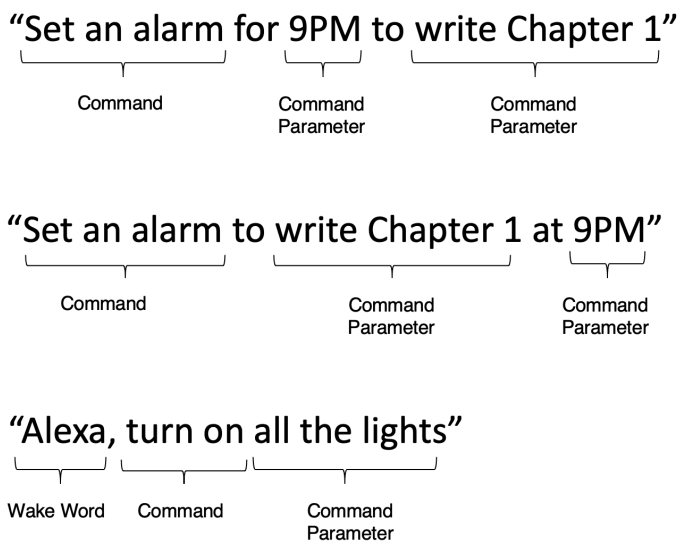


Figure 6 Exploring the parts of a command

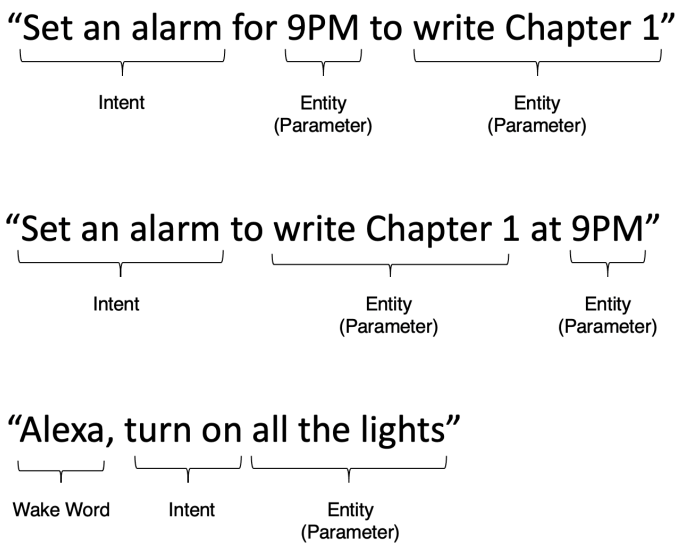


Figure 7 Alternate terminology for the command interpreter pattern

The command interpreter's job is to identify a command (ex: “set alarm”) and fill all of the parameter "slots" (ex: time of alarm) for that command. When the command interpreter is invoked it will ask as many follow-up questions as needed to fill all of the required slots -

no more questions and no less. This is similar to how virtual assistants work except that command interpreters only service a single request - the entire conversation is finished when the required parameters are gathered, and the initial command is executed. Figure 8 shows a command interpreter using slots to set an alarm.

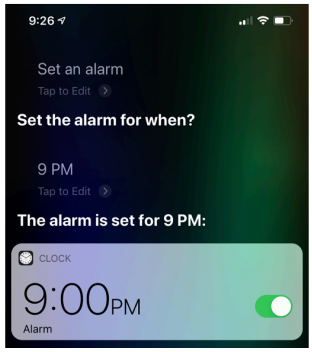
Do virtual assistants always have "commands"?

Terminology differs. Generally, in this book I will refer to intents and entities; nearly every virtual assistant platform uses the "intent" terminology. Some platforms use "action" and "intent" interchangeably.

In the case of command interpreters, commands are intents and parameters are entities.

1. The user invokes a command.

3. The user supplies the missing parameter.



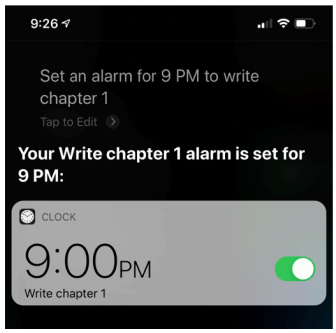
2. The command is recognized but is missing a required parameter.

4. The command is executed, and the conversation is over.

Figure 8 A command can require follow-up questions from the command interpreter

Command interpreters can also be coded to receive multiple parameters at once as long as there is a way to distinguish them. In the next example, I provide all of the alarm parameters in a single statement: "Set an alarm for 9PM to write Chapter 1." The phone gladly obliges this request! Figure 9 shows how the command interpreter reacts when all parameters are passed at once.

1. The user invokes a command with two parameters:
   - Alarm time
   - Alarm label



2. All required parameters are present. The command is executed, and the conversation is over.

Figure 9 When all required parameters are passed to the command interpreter, it completes the command

and ends the conversation.

When developing a command interpreter, you will need to be able to separate the parameters from the command. In the case of setting an alarm this is not too difficult: the command will be something like "set an alarm", the alarm time will be text that parses to a time, and any remaining text (minus filler words like "for", "to", "at", etc.) will be the reason.

EVENT CLASSIFIER

The final category of virtual assistant technology is the Event Classifier pattern. In this pattern, there is frequently no dialog at all! You are probably most familiar with this pattern through your email client. This pattern is most typically used in email routing and email assistants however this has also been used for routing support tickets to appropriate queues. Like the Command Interpreter pattern this category can also use parameter slots to ensure enough information is provided.

The event classifier pattern has been part of most email platforms since the rise of spam in the late 1990s and early 2000s. Some providers make it almost invisible to you (your internet service provider likely filters out spam before you even have a chance to download) and some providers let you peek a bit behind the curtain. The Gmail client runs an event classifier to sort your mail into separate mailboxes.

Event classifiers use a mixture of rules and machine learning techniques to decide how mail should be routed. In Gmail's case, the classifier decides which inbox to choose. Gmail offers several ways for the end-user to personalize or train the classifier through a mix of rules and training options. I selected an email in my Promotions folder and was offered the following training options:

- "Add *sender* to Contacts"
- "Block sender"
- "Report spam"
- "Report phishing"
- "Filter messages like these"

Figure 10 shows how classification logic can work inside an email client/

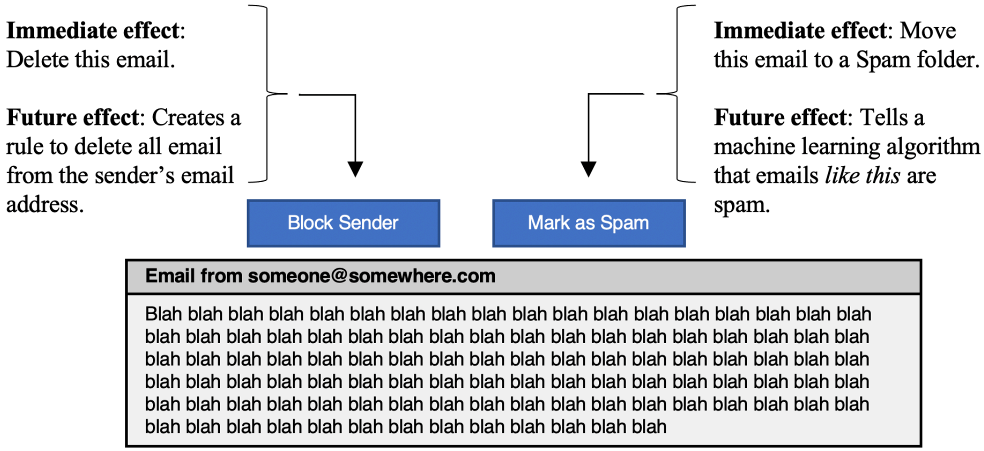


Figure 10 Actions in your email client can personalize the classification of your email!

The first two options are exposing rules: putting a sender in your contacts list means they should always go to the Primary inbox. Blocking a sender prevents their messages from appearing in your Primary inbox. The last three options are likely providing training to a machine learning algorithm. Additionally, you can drag a message from one inbox to another. When I moved a message from Promotions to Primary, I was prompted: "Conversation moved to Primary. Do this for future messages from sender?" Gmail is asking whether the rules and training for the event classifier should be updated.

Behind the scenes, additional rules are applied. Did the email originate from an IP address known to send spam? Did thousands of other Gmail users get the exact same message? Is the email filled with nonsensical words (v1agra)? Further, Gmail is constantly updating its training and rules based on the actions you and other users take.

Event classifiers can also be used in bug report or trouble ticket submission forms. A software product support team could use an event classifier on their website, tied to a form that lets users request help. Users can be instructed to describe their problem in natural language and the event classifier can decide how the problem should be handled.

Just like the virtual assistant and command interpreter patterns, the classifier needs to find an overall intent in the message. This will be used to direct the message – there are often different teams supporting each different kind of issue (installation, crashes, general questions). Just like parameters in the command interpreter pattern, there are also supporting details that each issue type requires (like a product version and operating system) and the event classifier can request these parameters if they are not provided.

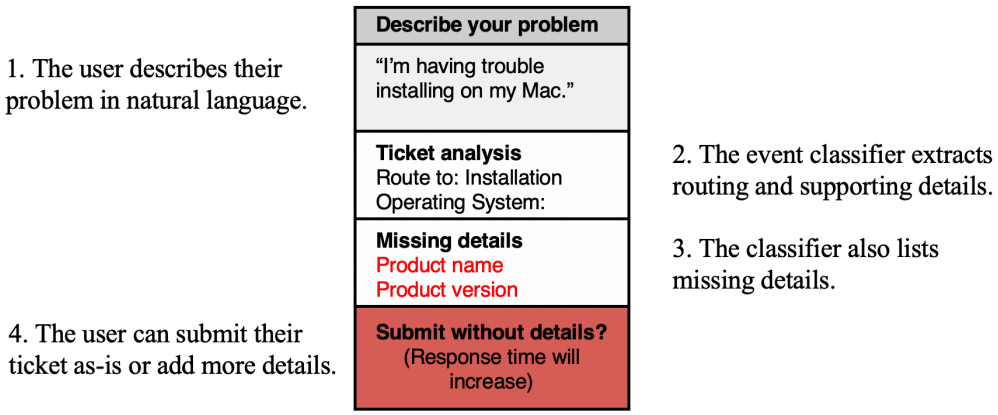


Figure 11 An event classifier helps classify support requests. Even better, it tells users what information will be needed to get a faster answer.

Figure 11 shows a form where a user has entered a vague description. The description is enough to route the request to the Installation team, however that team will need more details to solve the problem quickly. The user can proceed as-is or can update the description. In Figure 12 the user takes the opportunity to improve the problem description and the event classifier tells them that all the required information is present.

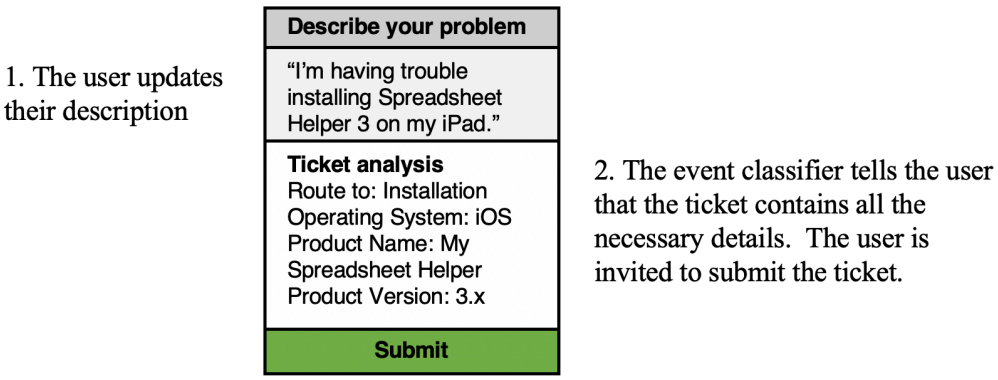


Figure 12 The user updates their problem description and the event classifier finds all the required parameters.

Why not just use a form with dropdowns?

You may be inclined to present a form to the user, with all of the required and options available as dropdown menu options. Many support forms work exactly that way. There are a few challenges with this approach:

1. The number of dropdowns can be intimidating to a user
2. Different request types might require different parameters, increasing the size and complexity of the form
3. The user may not know all of the right options to select. (In Figure 12, the event classifier inferred `iOS` operating system from `iPad` in the description.

A simple form with a single natural language field is approachable, and the event classifier can nudge the user to provide additional useful information.

Classifiers are powerful sorting technology. All virtual assistants have a classifier at their heart whether that classifier uses rules, machine learning, or both (we will discuss more in future chapters).

1.1.2 A Snapshot of Virtual Assistant Platforms

There are several different virtual assistant platforms. Most of these platforms are suitable for building multiple kinds of virtual assistants.

MICROSOFT

Microsoft offers a full catalog of software services that you can use to build your own solution. Their solutions are very friendly to developers. Microsoft offers separate Azure Cognitive Services: Language Understanding Intelligent Service (for natural language understanding) and Bot Framework (for dialog and response). Microsoft's services are suitable for building all three virtual assistant types and integrate into a variety of external interfaces.

For instance, **Wolford**[8] deployed a guided self-service shopping assistant powered by Microsoft Azure Bot Service[9] to consumers in 14 different countries. The assistant answers common questions from shoppers and provides personalized "Style Me" recommendations from Wolford's product catalog.

AMAZON

Amazon also has a full catalog of software services. Amazon Lex is the primary service used for building virtual assistants but integrates easily with Amazon's other cloud-based services as well as external interfaces. Amazon Lex uses the same foundation as Alexa.

Amazon's **Alexa** is available on over 100 million devices worldwide, configurable with over 100,000 Alexa skills. Alexa skills are usually command interpreters, but you can use Lex to build conversational assistants as well. Amazon offers several other services for building your own event classifiers.

8https://customers.microsoft.com/en-US/story/wolford-consumer-goods-azure
9https://azure.microsoft.com/en-us/services/bot-service/

GOOGLE

Google is another platform with a variety of software services in its catalog and many pre-built integration points. Google's DialogFlow is the primary service used for virtual assistants, and they too offer several other services for building your own event classifiers.

Google's **Gmail** is a quintessential event classifier, with 1.5 billion users worldwide. Gmail sorts incoming mail into specific categories as well as detecting spam. Google's **Nest** is a configurable command interpreter used in many homes and businesses.

RASA

Rasa is the only open-source platform on our list, giving you full control over the software you deploy. The other vendors on this list have you control the classifier by changing training data, but Rasa will let you change the entire classification process.

Rasa is deployed at **Align**[10] in a Smart Newsletter. Align uses Rasa's virtual assistant as a concierge. The assistant increase engagements with Align's newsletter and decreases friction in processes such as webinar registration.

IBM

The last platform on our list, IBM also offers a platform with easily integrated services. Watson Assistant is IBM's virtual assistant platform, and it is suitable for building all three types of virtual assistants.

Humana[11] fields thousands of routine health insurance queries per day with a voice-enabled self-service virtual assistant. Watson Assistant serves these callers at one-third of the cost of Humana's previous system while doubling the call containment rate.

1.2 Primary use cases for Virtual Assistant technology

There are three primary use cases for virtual assistant technology aligning with the technical categories described above. These include self-service virtual assistant, agent assist, and classification and routing. These use cases are distinguished by who interacts with the virtual assistant and the way they interact.

1.2.1 Self-service virtual assistant

The self-service virtual assistant pattern is the most familiar pattern. In this pattern, a user interacts directly with the virtual assistant. The virtual assistant is generally, but not always, set up to handle inquiries otherwise answered by human agents.

This use case pattern is frequently used in:

- **Customer Service**: Resolving frequently asked questions without a wait or hold time
- **Conversational Commerce**: Guiding a consumer through an exploration and purchasing experience
- **Employee Productivity**: Help employees find enterprise guidance quickly
- **Home/Personal Assistant**: Smart devices perform tasks through a conversational interface

10https://rasa.io/pushing-send/case-study-align/
11https://www.ibm.com/watson/stories/humana/

When you use the assistant on your smartphone you are interacting with a virtual assistant. This style of virtual assistant is increasingly common in automated customer support. Touch-tone voice agents ("Press 1 for Billing, press 2 to Open an Account") like you may interact with from your bank or favorite retailer are technically virtual assistants. In this book, we will focus on virtual assistants that use a conversational interface rather than a button-driven or keyword-driven interface. Figure 13 shows how self-service virtual assistants are generally deployed.

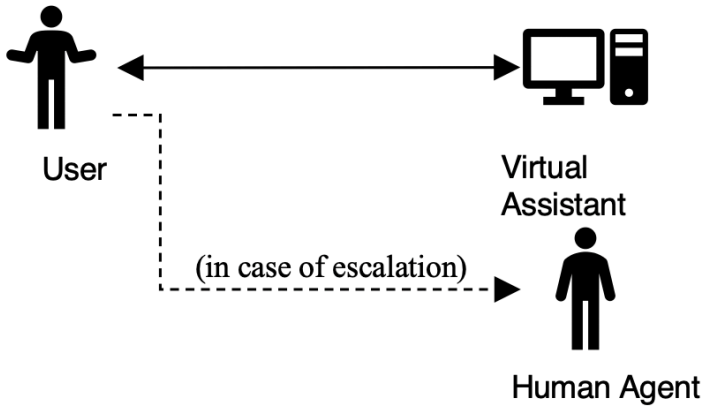


Figure 13 Model for how users interact with support agents

The actor diagram is simple in a virtual assistant. A user interacts directly with a virtual assistant which does its best to handle any requests. Optionally the virtual assistant may detect that it is unable to satisfy the user's request and can escalate the conversation to a human agent. The virtual assistant pattern can work in web (text) and voice channels.

1.2.2 Agent assist

Agent assist is a twist on the self-service pattern. In this pattern, the user interacts directly with a human agent. That human agent can augment their intelligence by using a virtual assistant. The virtual agent is like an expert "in the ear" of the human agent.

Within the agent assist pattern, there are two significant variations. In the first variation input from the end-user is sent directly to the virtual assistant and the virtual assistant sends output to the human agent. The first variation is shown in Figure 14.

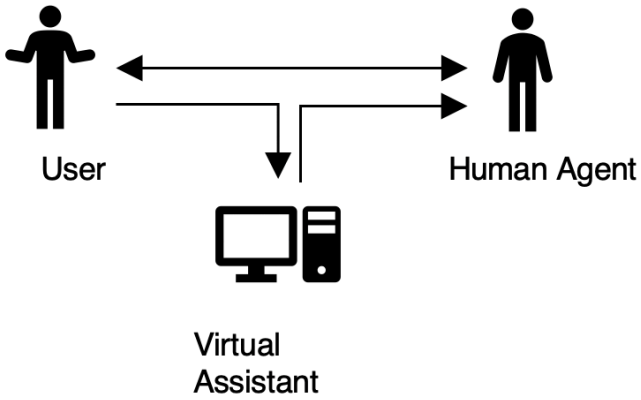


Figure 14 First model for interactions in an Agent Assist scenario. The user interacts only with the agent, but the assistant is listening to the conversation and advising the agent.

The agent assist pattern reduces the burden on the human agent by seamlessly providing them with additional information. The human agent can decide whether or not to include information from this response to the end-user. The downside of this approach is that it may require more virtual assistant training: the end-user is likely to include additional, "irrelevant" context when speaking to another human.

The other significant variation is for the human agent to use the virtual agent only when needed. This pattern is shown in Figure 15. In this variation, the agent forms a question, possibly well-structured, to the virtual assistant and receives a response. The agent can incorporate the information in a response to the end-user. This requires active participation from the agent - they have to type in a question to the virtual assistant - but requires less virtual assistant design and implementation work.

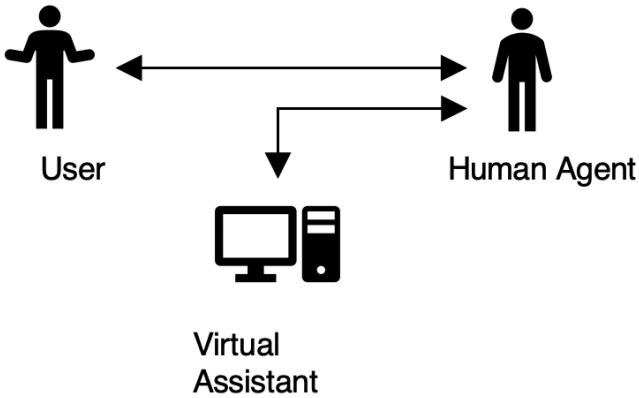


Figure 15 Alternate agent assist pattern. The assistant is invoked by the agent only when required.

Both scenarios are commonly used in training and supporting new customer service agents. Customer service departments frequently have high turnover and the agent-assist pattern is a great way to raise the average agent ability. Agent assist can also work in web (text) and voice channels.

1.2.3 Classification and routing

The final use case is to use a virtual assistant to classify data that's already been received. This classification pattern is commonly run on emails but is suitable for many other input types including submitted forms and social media posts and comments. A classifier can quickly sort through a large volume of incoming requests and the resulting classification informs how the requests should be routed. Figure 16 shows an example of the classification and routing pattern.

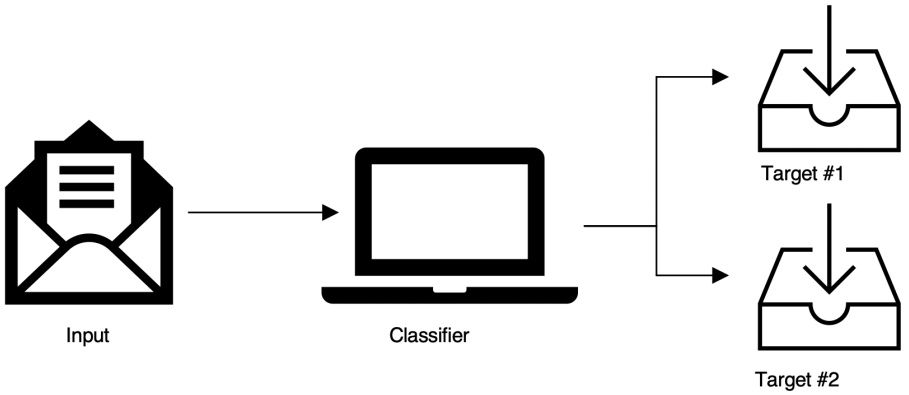


Figure 16 Example classifier and routing pattern.

A variation on this use case is to run the virtual assistant before a form is submitted. The virtual assistant can run on free text in a form and detect what critical information is and is not provided. For example, in a bug report form, the assistant can detect whether you have provided information on your operating system, product version, and what general problem type you are experiencing.

1.3 Follow Along with this Book

The techniques in this book will apply to most virtual assistant platforms. I will reference IBM Watson Assistant[12], a virtual assistant platform, which is suitable for implementing each of the intelligent architectures outlined in this book. You need not use Watson Assistant, however, to implement your assistant. The book's examples are helpful guides in the spirit of all virtual assistant platforms. Feel free to use other ones that I discussed earlier such as Microsoft Azure Bot Service, Amazon Lex, Google DialogFlow, Rasa, or any other platform you prefer.

Watson is designed to run wherever you need it whether that's on a public cloud or on your premises. The interface is suitable for either technical or non-technical users. You will be able to build your own virtual assistant on IBM Watson Assistant using a Lite (free) account.

1.3.1 What you need to create your assistant

If you are following along in Watson Assistant, you will need the following:

- **IBM Cloud account**: You can register for a free IBM Cloud account at https://cloud.ibm.com.
- **Watson Assistant instance**: With your IBM Cloud ID you can create a Watson Assistant instance at https://cloud.ibm.com/catalog. A Lite plan is available free of

[12] https://www.ibm.com/cloud/watson-assistant/

charge and is sufficient to run all of the examples in this book.

Terminology alert!

There are many virtual assistant platforms hosted on vendor clouds. They generally include the following concepts:

Account: You have to register with the provider, at minimum with an email address or account identifier.

Try before you buy: At the time of this writing all major platforms provide some sort of free, lite, or trial option. Be warned: some platforms require you to register a credit card with your account, even if you only use free services. If your provider requires a credit card when you sign up, be sure to read the fine print.

Instance: This refers to your personal "copy" of the virtual assistant software. This may also be called a service, service instance, or resource depending on your platform. You may also see reference to "resource groups" or "service groups" – these are intended to help enterprises keep many separate instances segregated. If you're presented an option on something like a "resource group" you can safely take the default.

If you are not using a cloud-based platform you may not see any of these terms.

If you are using a different virtual assistant provider, look through their documentation for an option on using their software for free.

1.3.2 Useful spreadsheet software

Regardless of platform, you will find it handy to have **spreadsheet software**. You can use any software that is capable of reading a comma-separated values (CSV) file. Spreadsheet software commonly works with this format.

Virtual assistants are commonly trained with two-column CSV files. Each row of the file is used to train the assistant, with the first column being an input and the second column being the expected output. We'll get deeper into the specifics later.

1.3.3 Recommended Programming Language and Code Repository

There will be a small number of code examples for testing virtual assistants. To run these examples, you will need:

- **Python**: An environment to run Python scripts and Jupyter notebooks. Python can be downloaded from https://www.python.org/downloads/.
- **Git**: A git client is needed to download code samples from the Internet.

The code samples are available at the book's GitHub repository: https://github.com/andrewrfreed/CreatingVirtualAssistants

When the code assumes a particular virtual assistant platform, I will describe what the code is doing and how to achieve the same results in other virtual assistant platforms.

1.4 Summary

- Virtual assistants are suited to solve a variety of tasks, most commonly through a conversational interface and with varying amounts of dialog. Some virtual assistants have no dialog at all (event classifiers). Some assistants only use enough dialog to get the information needed to execute a single command.
- Virtual assistants can be used by consumers for self-service tasks or to augment the intelligence of a human agent.
- Some commands and questions can be answered immediately by a virtual assistant, and some require follow-up questions to gather additional detail.
- When you mark an email as spam, you are training a virtual assistant!

2

Building your first virtual assistant

This chapter covers:

- **Identifying the intent and entity in a single user utterance**
- **Implementing a question-and-answer dialog for a recognized intent. Then, add contextual information to that answer when an entity is recognized.**
- **Implementing a multiple-question process flow to satisfy a user request.**

In the previous chapter, we learned about what virtual assistants are, why people use them, and some examples of virtual assistants that we encounter every day. In this chapter, you will learn how they work, and you will build an assistant from the ground up. We will work through a case study for a retail company looking to build their first virtual assistant.

The first thing assistants need to do is recognize what user's utterances actually mean. A conversation starts with a user stating an intent and any related entities. The intent will help the assistant understand what to do next – whether to respond with an answer or to start a new dialog flow. We will first see in detail how the assistant breaks down a user utterance into intents and entities.

The simplest response is for the assistant to respond to each intent with a single "answer" response. We will configure an assistant to recognize a variety of different intents and to give a unique response for each. We will further explore how to provide additional information in a response based on the entities detected in the user's utterance.

Some user intents require a multi-step process flow that includes one or more follow-up questions. We will also explore methods for building these process flows in a virtual assistant. We will also implement a "two strikes" rule so that if the assistant consistently doesn't understand the user, the user will be given a different resolution option.

By the end of this chapter, you will be able to build your first virtual assistant. You will be able to implement simple question and answer responses. You will be able to tailor a response based on contextual details in the user's utterance. You will also be able to

implement multi-step process flows that guide a user through a series of steps, ending with a resolution for the user. Armed with this knowledge you will be ready to design the process flows for your next virtual assistant.

2.1 Building a virtual assistant for FICTITIOUS INC

FICTITIOUS INC is a growing retail chain looking to reduce the burden on their support representatives. They want to build a self-service virtual assistant to handle their most frequent customer questions. They primarily receive questions on store locations and store hours, scheduling appointments, and job applications. In this chapter, we will start building their assistant.

FICTITIOUS INC's use case is a typical retail scenario. Customer service departments are often overloaded with routine inquiries that can easily be automated. FICTITIOUS INC will offer a virtual assistant to their users as a fast way to get answers, without needing to wait for a human to answer questions. When their virtual assistant cannot answer the user's questions, users will be redirected to the existing human-based customer service processes.

FICTITIOUS INC prefers to leverage pre-trained content to reduce their virtual assistant development time, though this is not a hard requirement. They have made it clear that they will use any technology that works, giving you free range to pick the virtual assistant platform of your choice. They only require that the platform includes a "try before you buy" option. In this chapter, I will demonstrate the FICTITIOUS INC scenario using IBM Watson Assistant.

2.1.1 Creating the assistant

FICTITIOUS INC first needs to get some virtual assistant software. For this demo I registered an account on the IBM Cloud[1], and created a personal (free) instance of Watson Assistant by importing it from the IBM Cloud Catalog[2]. I'm asked to configure a service instance name (any name will do) and select a geographic region (the closest to your users, the best).

If you are using a different cloud-based platform you will encounter a similar workflow. You generally need to sign up (possibly with a credit card) and can find a free/lite/trial option. If you are not using a cloud-based platform you will need to download and install the virtual assistant software and any dependencies it requires (be sure to check the documentation).

After this virtual assistant software is configured for your use you need to create a specific virtual assistant that uses this software. Figure 1 demonstrates the relationship between the former and latter steps.

[1] https://cloud.ibm.com
[2] https://cloud.ibm.com/catalog

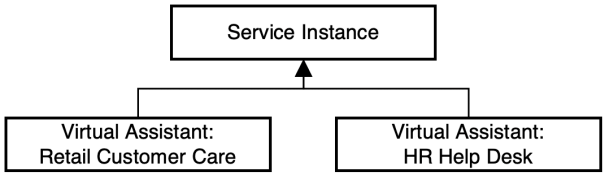


Figure 1 After configuring the Service Instance (or synonymous item) for your platform, you'll need to create a virtual assistant for your specific use case. You can create separate virtual assistants using the same service instance.

The next step is to create a virtual assistant specific to your use case. The specific way you do this varies greatly depending on your platform of choice.

Terminology alert!

You want to create a virtual assistant, but what does your platform call it? They may call it an Assistant, a Bot, an Agent, or something else entirely! Sometimes the Assistant/Bot/Agent has everything you need and sometimes you couple it with a Skill, another service, or raw code.

The different naming schemes are sometimes just cosmetic and sometimes indicative of different architectural patterns. Many platforms use a layered architecture allowing you to connect to a variety of front-end user interfaces and back-end service integrations. For the FICTITIOUS INC demo, we will not worry about integrating with any other systems.

The one exception to this rule is if your virtual assistant platform separates the "virtual assistant" service from the classifier or "natural language understanding" service, you will need to integrate them to build your assistant. Check your platform's documentation when we get to the intent training section.

Since I am working with Watson Assistant I have two more components to create. (Your platform will likely use different terminology. Use Figure 2 to figure out what your virtual assistant platform requires.) The smallest possible configuration in Watson Assistant's v2 API uses these two parts:

1. **Assistant**[3] – The interface integration layer.
2. **Skill**[4] - The dialog engine.

[3] https://cloud.ibm.com/docs/assistant?topic=assistant-assistants
[4] https://cloud.ibm.com/docs/assistant?topic=assistant-skill-dialog-add

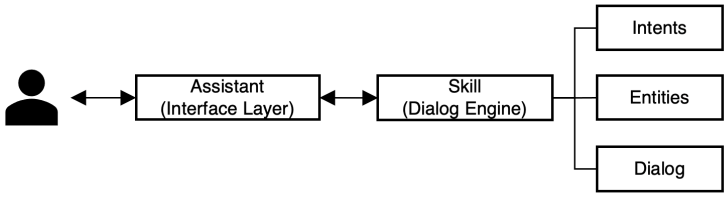


Figure 2 The Watson Assistant components used for FICTITIOUS INC. The terms Assistant and Skill may vary in your platform, but you can expect to find intents, entities, and dialog.

When I created a new Watson Assistant instance it came pre-loaded with an Assistant called "My first assistant". There was no other default configuration provided with the assistant, so I added a Dialog Skill to implement FICTITIOUS INC'S scenario. The "Add dialog skill" button creates a new, empty Dialog Skill as shown in Figure 3.

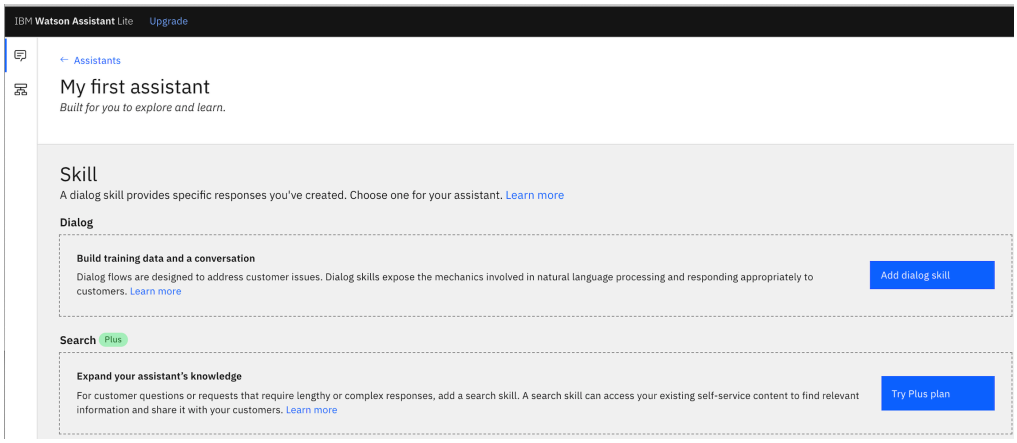


Figure 3 Adding a dialog skill to the assistant. Your platform likely has choices that differ in name but are similar in function.

When creating a Skill, you will be prompted to provide a name, description, and language. (Your platform will collect similar data.) Be sure to provide a useful name. The name and description are made available for your benefit. I created the FICTITIOUS INC Skill with the name "Customer Service Skill", description "Answer most common customer service questions", and the English language. After clicking "Create dialog skill" the Skill is linked to the assistant.

If you are using another virtual assistant platform you will have similar options. You may be prompted to pick a specific language that the assistant will converse in. Some of the virtual assistant platforms I surveyed allow you to create an assistant based on a sample provided by the platform. For following along in this chapter, I suggest creating a new, empty assistant.

With the basic virtual assistant infrastructure (Assistant and Skill in Watson Assistant, a Bot or Assistant or Agent in other platforms) in place, it's time to start implementing FICTITIOUS INC's use case. We configure three specific components inside the virtual assistant:

- Intents – this will show us all the training data made available to the assistant. This will be empty until we manually provide some intents or import them from the content catalog.
- Entities – we will use this to identify several FICTITIOUS INC store locations
- Dialog – this is where we will encode the dialog flows used in FICTITIOUS INC's assistant.

Our new assistant is almost completely empty. Let's start by adding some intents, so that the assistant can understand what the user means.

2.2 What's the user's intent?

A conversation in a virtual assistant generally starts with the following pattern:

1. The user speaks/types something. This is called the utterance.
2. The assistant determines what the utterance means, or what the user is trying to accomplish. This is called the intent.
3. The assistant formulates a response and sends it to the user. This response may be an answer or a request for additional information.

The pattern is displayed in Figure 4.

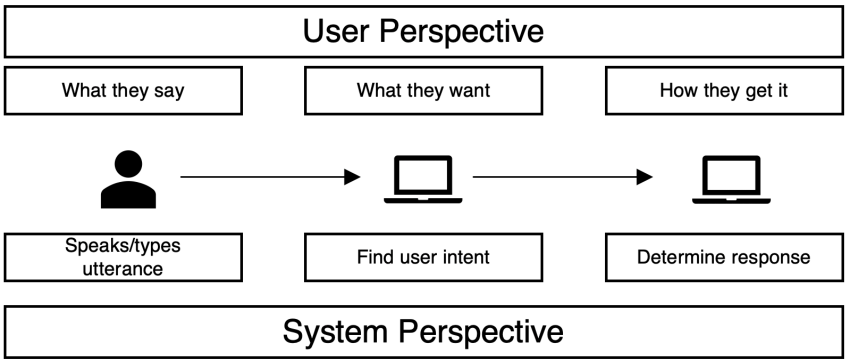


Figure 4 One turn of a conversation, from the user and system perspectives.

Terminology alert!

Every virtual assistant platform I surveyed at the time of this writing used the term "intent" in the same way. Hooray for consistency! Some platforms are starting to hide the "intent" terminology (are they afraid it scares users?), even though they use intents under the covers.

FICTITIOUS INC's users will use a nearly infinite number of utterances and these will map to a finite set of intents. For FICTITIOUS INC's assistant, we will need to create a set of intents and map dialog responses to those intents. Let's explore each of these more deeply.

2.2.1 What's an utterance?

The user primarily interacts with assistants through natural language. The user's input is called an "utterance" and the utterance is literally just what the user says (in a telephone assistant) or types (in a web/chat assistant).

Terminology alert!

Nearly every virtual assistant platform I surveyed used the term "utterance". A few platforms referred to them as "examples" or "training phrases". Search your platform's documentation for the section on training intents and you'll quickly find how they refer to "utterances".

Example utterances include:

- "Where are you located?"
- "What time do you open?"
- "How can I reset my password?"
- "Tell me a joke"

Can users provide input to a virtual assistant besides text?

Yes! There are other ways for users to provide input. In a web channel, assistants the user may also provide input by clicking on a button. In a voice channel, assistants the user may enter numeric values on their keyboard with Dual-Tone Multi-Frequency (DTMF) signals. Virtual assistants still generally treat this input as textual.

2.2.2 What's a response?

The end goal for a virtual assistant is to respond appropriately to a user's utterance. A *response* is simply whatever the assistant returns to the user. A set of example utterances that FICTITIOUS INC's users might say, along with likely FICTITIOUS INC responses, are shown in Table 1.

Table 1 Example dialog utterances and responses

Example user utterance	Example system response
"Where are you located?"	"We have two locations near you: 123 Maple St and 456 Elm Drive."
"What time do you open?"	"Our hours are 9am to 8pm on weekdays and 10am to 10pm on weekends."
"How can I reset my password?"	"What's your user ID?"
"Tell me a joke"	"I'm not trained to handle that. I can help you with several other common IT requests like password resets."

Terminology alert!

Nearly every virtual assistant platform I surveyed used the term "response". I also saw "dialog response", "action", "message", "prompt", and "statement".

If you consider the paradigm "if this, then that", the "that" is the response.

Table 1 shows what the user sees, but internally the assistant does a little more work to determine that response. The assistant uses natural language understanding (NLU) to extract an intent from the user's utterance and uses that to further drive the conversation. Let's look at how an NLU module does that.

Can virtual assistants respond with output other than text?

Yes! There are multiple ways to provide output in a virtual assistant. Web assistants can use buttons, images, hyperlinks, and other rich media in their responses. Voice assistants can add music or other audio in a response.

2.2.3 How does the assistant understand what the user means?

The assistant makes use of a natural language understanding engine that finds meaning in the user's utterance. The natural language understanding component has two parts:

- Intent classifier: Finds meaning that applies to the entire utterance. The meaning is generally expressed as a (verb-based) intent and is used extracted via machine learning.
- Entity extractor: Finds zero or more subsets of the utterance with a specific meaning, such as dates, locations, or other objects. The entities are usually noun-based and may be extracted by machine learning, hardcoded rules, or both.

The natural language understanding component in your platform may offer separate training options for the classifier and the entity extractor. Table 2 summarizes when you use intents and entities.

Table 2 Key differentiation between intents and entities. Both are extracted via a natural language understanding component.

Intent	Entity
Verb-based	Noun-based
Considers the whole utterance	Extracted from a word or segment in the utterance
Usually uses machine learning	Uses rules and/or machine learning

WHY MACHINE LEARNING?

A classifier uses machine learning because learning to understand text by example is a more scalable approach than implementing rules. Utterances come in natural language which has nearly infinite variations. It is difficult to code enough rules to handle all possible variations in text, but a machine learning classifier can scale quickly. Table 3 shows several possible ways that a user may state they need their password reset.

Table 3 Example utterances from a user whose intent is to reset their password. Set A is handled easily with rules, but Set B is more easily handled with a machine learning classifier.

Set A	Set B
• Reset my password • Please help me reset my account password • My password needs to be reset	• I forgot my password • Reset my login information • I can't sign into the intranet • I'm locked out of my email account

A first and naive approach to intent detection would be to use keywords to apply structure to unstructured text. Table 3's Set A is easily handled with a rule: any utterance containing "reset" and "password" should be treated as a password reset request. Table 3's Set B shows why a *rules-based* approach grows unwieldy quickly. There are nearly infinite ways to express most intents in natural language and it is too hard to write rules to cover them all.

The classifier uses machine learning techniques at its heart. We'll see how the classifier learns to identify a user's request later, so for now we can assume we have one trained and ready to use. The classifier attempts to derive intent from every input phrase and passes this intent to the dialog engine. The dialog engine is then able to use structured rules to determine the next dialog step.

WHAT'S AN INTENT?

An intent is a normalization of what the user means by their utterance, or what they want to do. An intent is generally centered around a key verb. Note this verb may not be explicitly included in the utterance! By convention, I will reference intents with their name preceded by a pound (#) sign, such as `#reset_password`. This is the convention used by most of the popular assistant platforms. Table 4 shows sample utterances from FICTITIOUS INC's users along with the identified intent, and the verb(s) at the center of that intent.

Table 4 Sample utterances with associated intent and implied verb

Utterance	Intent	Implied Verb(s) of the Intent
"Where are you located?"	#Store_Location	Locate
"What time do you open?"	#Store_Hours	Operate/Open/Close
"How can I reset my password?"	#Reset_password	Reset/Unlock
"Tell me a joke"	#Chitchat	n/a

An intent represents a logical grouping of similar utterances that all have the same user meaning, i.e. user intent. As in the previous section "I need to reset my password" and "I'm locked out of my account" both indicate locking/resetting problems. A classifier is more powerful than a simple keyword matcher and uses contextual clues in the natural language to derive the user's intent.

Classifiers are trained with a series of examples and their corresponding intents.

It is possible that a user utterance will not map to any of the intents the classifier is trained on. In this case, the classifier will respond with some indication that the utterance is poorly understood, either by not returning an intent or returning an error condition. In Watson Assistant's development interface this scenario is flagged as `#Irrelevant` and in the API no intents are returned. Your virtual assistant platform of choice will do something similar when a user utterance is not well understood.

A classifier needs careful training for optimal performance. This training will be covered more thoroughly in the following chapters. For now, just consider that intents are trained with a variety of example utterances with similar meaning and that intents should be clearly differentiated from one another. In the end, the classifier will evaluate the user utterance for an intent, and send that intent back to the dialog engine.

WHAT'S AN ENTITY?

Entities are noun-based terms or phrases. An intent applies to the entire user utterance, but an entity applies to part of the utterance, often just a single noun or noun phrase. By convention virtual assistant platforms reference entities with an "at" (@) sign preceding their name. Table 5 shows some example entities FICTITIOUS INC can use in their assistant.

Table 5 Example entities for FICTITIOUS INC.

Entity Type	Entity Value (canonical form)	Entity synonyms (surface forms)
@store	Elm	Elm Dr, Elm Drive
@store	Maple	Maple St, Maple Street
@contact_method	phone	mobile, SMS, smartphone
@contact_method	email	webmail

The simplest entity pattern is to use a data structure called a dictionary - an enumerated list of all possible words and phrases you want to treat the same way.

Terminology alert!

Most virtual assistant platforms use the term "entity". Occasionally I saw "parameter" or "slot type".

Entities are typically defined in one of three ways:

1. A pre-defined, static list (sometimes called a "dictionary" or "lookup table")
2. Regular expression patterns
3. Trained by example

Virtual assistants often come with built-in "system entities" to detect things like dates and countries.

If all of the entity variations are known ahead of time the dictionary-based approach works very well. FICTITIOUS INC's virtual assistant will use dictionary-based entities for their customer service use case.

What's a dictionary?

The traditional usage of a dictionary is a book, or another repository, with a list of words and information about those words: definition, pronunciation, part of speech, and sometimes other features like synonyms. In this context, a synonym for a given word is meant to be a similar but different word.

In the context of virtual assistants, a dictionary is simply a list of known words, each word having an associated type. Synonyms in this kind of dictionary are completely interchangeable.

In Table 5 `@store` was a type. "Elm", "Elm Drive", and "Elm Dr" can all treated interchangeably. Many virtual assistants have spelling correction, or a feature like "fuzzy matching", which allows misspellings like "Eml Drvie" to be treated as if they too were in the dictionary.

COMBINING INTENTS AND ENTITIES

Virtual assistant platforms use entities to provide additional contextual information, besides just the intent, that can be used to provide a response. When utterances share an intent but differ only by the nouns, entities are the best way to determine a response. Table 6 contrasts two assistants, one that does not use entities and another that does.

Table 6 Contrasting two virtual assistants, one that does not use entities and one that does. The assistant without entities will be much harder to train.

Utterance	Without entities	With entities
Where are your stores?	`#store_location_all`	`#store_location`
Where is your Elm store?	`#store_location_elm`	`#store_location, @store:Elm`
Where is your Maple store?	`#store_location_maple`	`#store_location, @store:Maple`

You might try to treat "Where is your Elm store" and "Where is your Maple store" as different intents. But machine learning has difficulty distinguishing multiple intents that use nearly identical utterances. We will explore the reasoning in Chapter 7. For now, we can assume that using entities will help us build a more accurate virtual assistant.

Use entities when multiple questions have the same intent but need different responses.

2.2.4 Adding intents to FICTITIOUS INC's assistant

FICTITIOUS INC's new assistant is almost completely empty. Critically, it does not include any intents, therefore it will not understand any user responses. If you are using Watson Assistant you can import intents from the Watson Assistant Content Catalog[5] – this will provide some initial training data for the general customer service intents FICTITIOUS INC is interested in.

Not using Watson Assistant? No problem!

If your platform does not have sample intents for a retail scenario, you can create your own training data. In this chapter, we will use the following intents, and you can create a few example utterances for each such as noted in this sidebar. Add a few utterances of your own to increase the training quality.

`Customer_Care_Store_Location`: "Where are you" and "How do I find your store"

`Customer_Care_Store_Hours`: "What times are you open" and "When do you close tonight"

`Customer_Care_Employment_Inquiry`: "I need a job application" and "Can I work for you"

`Customer_Care_Appointments`: "I need to schedule an appointment" and "When can I see a manager"

After clicking the Content Catalog menu item, I selected the Customer Care category and clicked "Add to skill". The assistant is now populated with 18 different customer service intents, and each intent has 20 different examples. This is a robust set of training data to start FICTITIOUS INC's assistant. For example, the assistant will be able to recognize store location questions like "Where is your store?" via `#Customer_Care_Store_Location` intent.

As soon as the Customer Care content is loaded into the assistant you need to train the natural language understanding component. Your platform may train automatically (Watson

[5] https://cloud.ibm.com/docs/assistant?topic=assistant-catalog. Your virtual assistant platform may also offer sample training data; check your documentation.

Assistant is one such platform) or you may need to explicitly invoke a training action (check your platform's documentation). Regardless of platform, the training process should only take a few moments.

Virtual assistant platforms generally include a testing interface. In Watson Assistant the interface is called "Try it out". I've also seen the test interface referred to as a simulator or developer console. Some platforms do not provide a graphical testing interface but expose APIs you can use for testing. The testing interface will usually include information that helps you debug your assistant but that you won't expose to users (such as the name and confidence of any intents found). After your assistant is finished training, you can explore how well it recognizes intents by using your platform's testing interface.

Test your assistant with questions that were not included in your training data. (Use your imagination!) I asked my assistant "What time do you open?" and "Where is the nearest store?". As shown in Figure 5, the assistant recognizes the intent behind these statements (`#Customer_Care_Store_Hours` and `#Customer_Care_Store_Location`, respectively) but responds to the user as if the assistant didn't actually understand.

Testing tip: See how your assistant responds to phrases you did not explicitly train on!

1. Test question from user
2. Intent recognized by assistant
3. Dialog response from assistant

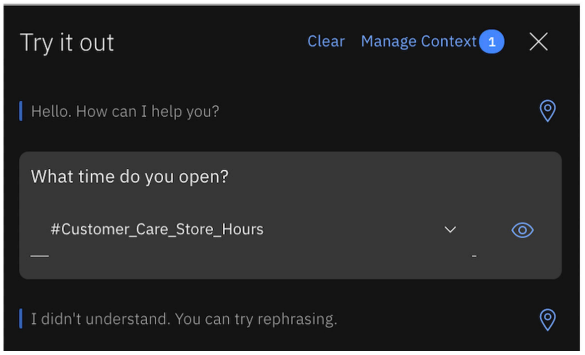


Figure 5 A testing interface shows some information that you won't expose to users (like the name of the intent). Here I can see that the new assistant recognizes my intent. I can now infer that the "I didn't understand" response is because no dialog condition is associated with that intent.

The assistant understands several different intents but does not know how to respond to any of them. Figure 6 summarizes what our assistant can (and can't!) do.

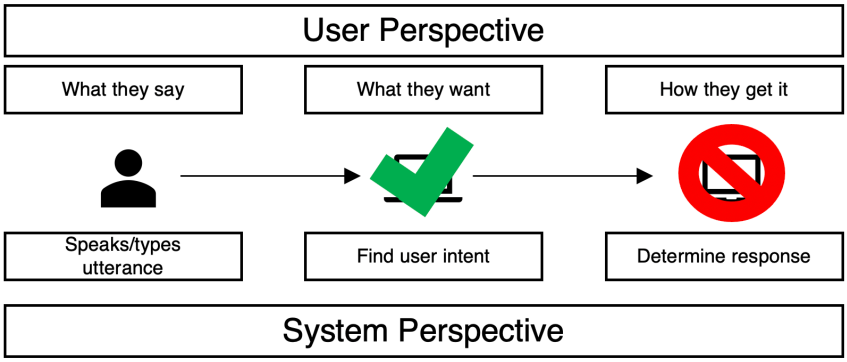


Figure 6 When you add intent training data, but do not configure dialog responses, you are here.

We've made nice progress but we're only halfway there. Let's teach the assistant how to respond to the user in a meaningful way.

Why are intents and responses separated?

In many virtual assistant platforms finding the intent is only one part of generating a response. Training an intent classifier and building dialog rules to generate responses are two separate skills.

Nonetheless, there is a growing trend among virtual assistant providers to combine these steps. You may be using a platform that abstracts away intents. The training interface may ask you to just provide the "this" (utterance) and "that" (response) in an "if *this*, then *that*" pattern.

2.3 Responding to the user

Just like a brand-new assistant needs to be loaded with intents, the assistant also needs to be populated with dialog rules. It's not sufficient to recognize the user intent – you need to respond appropriately to that intent. Virtual assistants use conditional logic to determine which response to give to the user. The simplest conditional logic is "if the user's intent is *this*, respond with *that*".

Terminology alert!

Dear reader, dialog logic is where there is the most variation between virtual assistant vendors! They all use some form of conditional logic, but the specifics of this logic vary.

One major difference is the interface paradigm: some use a graphical interface, and some require you to write code using their libraries, to implement dialog logic.

Most virtual assistants have an event loop at their core. Most interactions with the assistant follow an "if this, then that" pattern over and over until a conversation ends. Virtual assistant platforms have very similar conditional logic primitives even if they use different names for them.

In this section, I will use lots of screenshots since "a picture is worth a thousand words". I am confident you can map each of the examples in this section to your virtual assistant platform of choice.

Your virtual assistant platform provides some method to implement dialog logic, either through a graphical editor or through pure code. A common way to reason about dialog logic is to think of the dialog as a decision tree. (This harkens to call centers that train employees on scripted "call trees".) In this methodology, every decision point in the dialog could be called a "dialog node". You'll be creating several dialog nodes as you implement all of the conversational paths supported by your assistant.

Before creating new dialog response nodes for FICTITIOUS INC let's explore the dialog nodes they got by default. Virtual assistant platforms will generally include a few dialog nodes, or dialog rules, to help you get started in your assistant. Figure 7 shows an example set of default dialog nodes.

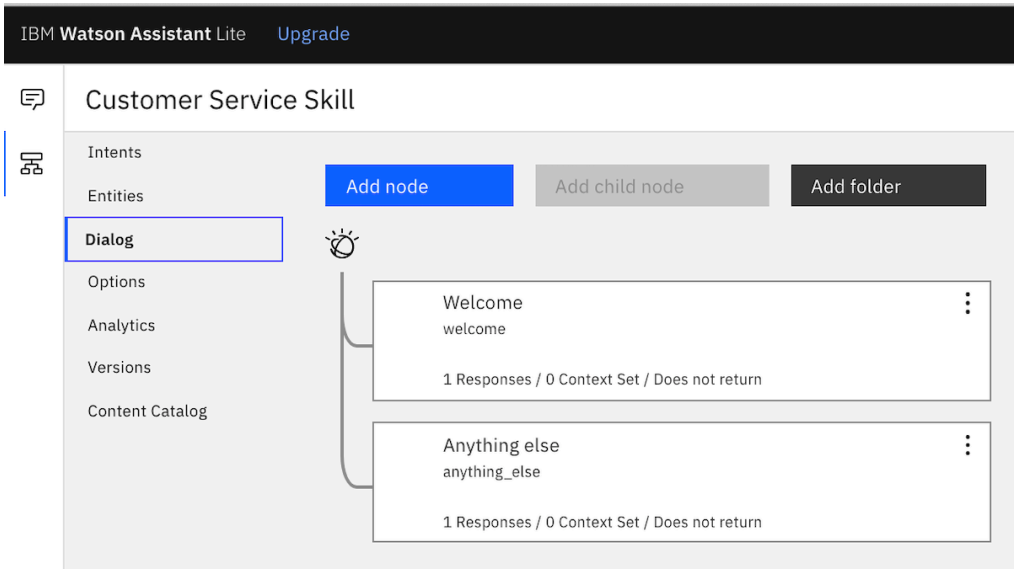


Figure 7 Dialog nodes created automatically by Watson Assistant. Your virtual assistant platform of choice will come with some default conditions and/or dialog nodes.

Each of the dialog nodes in Figure 7 includes a special condition that allows it to execute. The first is the "welcome" node – this is the node that contributed the "Hello. How can I help you?" greeting. The second is the "anything else" node which provided all the other dialog responses like "I didn't understand. You can try rephrasing."

The two nodes and their conditions are described below in Table 7.

Table 7 Pre-created dialog nodes and associated conditions in Watson Assistant. Other platforms have similar conditions, intents, events, and/or dialog nodes that handle these very common situations!

Dialog node title	Node condition	Description
Welcome	welcome	This represents your bot's greeting to users. The welcome condition is only true on a user's very first interaction with the assistant. In most virtual assistant instances, the assistant "speaks first" as a response to the user opening a chat window.
Anything else	anything_else	This represents a fallback or catch-all condition. The anything_else condition is always true and thus can be used to catch utterances that the bot does not understand.

Way back in Figure 5 we saw an assistant that recognized intents, but it responded with "I didn't understand". Now we know why. An assistant must be configured both to recognize intents and respond to conditions. If the assistant is not configured for both, it will fall back to some default behavior. Our newly created assistant has only a message that it uses to start a conversation (welcome) and a message it uses for every other response (anything else).

Terminology alert!

Your virtual assistant platform comes with some built-in conditions, events, or triggers that you can use.

"welcome" may be expressed as a "conversation start event" or a "welcome intent" – it just means a new conversation has started

"anything else" may be referred to as the "fallback intent", "default intent", "default condition", or "catch-all".

These two conditions are so prevalent in virtual assistants that your platform's tutorial is sure to include them!

FICTITIOUS INC needs rules coded into the assistant, telling it how to respond when more interesting conditions are met, such as when the `#Customer_Care_Store_Hours` intent is recognized in a user question. The assistant should respond to any `#Customer_Care_Store_Hours` question with some sort of answer about when stores are open!

2.3.1 Simple question and answer responses

The simplest possible dialog response is to provide a single, static answer based on the recognized intent. FICTITIOUS INC most commonly receives questions about store hours,

store locations, and how to apply for a job. Sample responses for these conditions will be coded into FICTITIOUS INC's assistant as shown in Table 8.

Table 8 The responses we'll code into the FICTITIOUS INC assistant when certain intents are recognized

When the assistant recognizes this intent...	It will give the following response
`#Customer_Care_Store_Hours`	**All of our local stores are open from 9am to 9pm.**
`#Customer_Care_Store_Location`	**We have two locations near you: 123 Maple St and 456 Elm Drive.**
`#Customer_Care_Employment_Inquiry`	**We accept job applications at FictitiousInc.com/jobs. Feel free to apply today!**

After coding these responses like Table 8 into the assistant, anyone who asks a question to the assistant will get a more useful response!

Figure 8 shows how to configure the first dialog rule for FICTITIOUS INC. If the assistant recognizes the user input as `#Customer_Care_Store_Hours`, it will respond with "All of our local stores are open from 9am to 9pm" and then wait for the user to provide additional input.

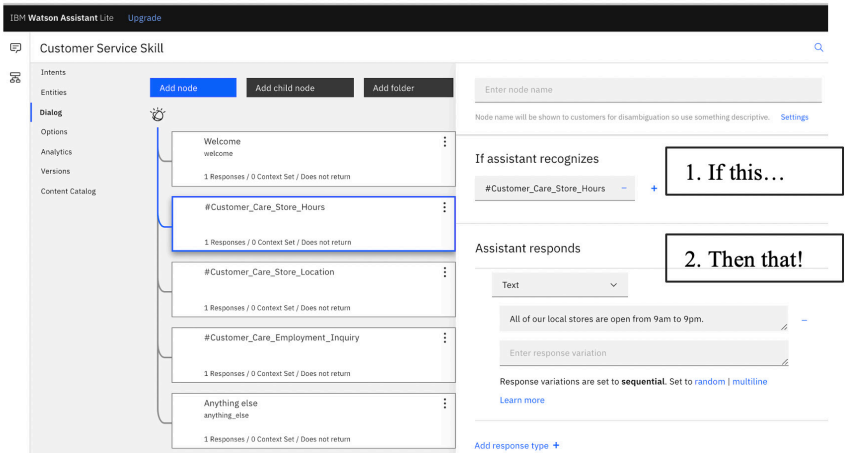


Figure 8 A simple dialog rule takes the form "if this *condition*, then give this *response*". Your virtual assistant platform may have a different interface, but you will surely be able to connect conditions to responses.

Figure 9 shows a test of the dialog configured as per Table 8. Note that we have only provided answers to three of the intents - for the other remaining Customer Care intents, even if the intent is recognized the assistant will use the catch-all "anything else" dialog node to respond. Each intent-specific response needs to be coded into the dialog as shown in Figure 8.

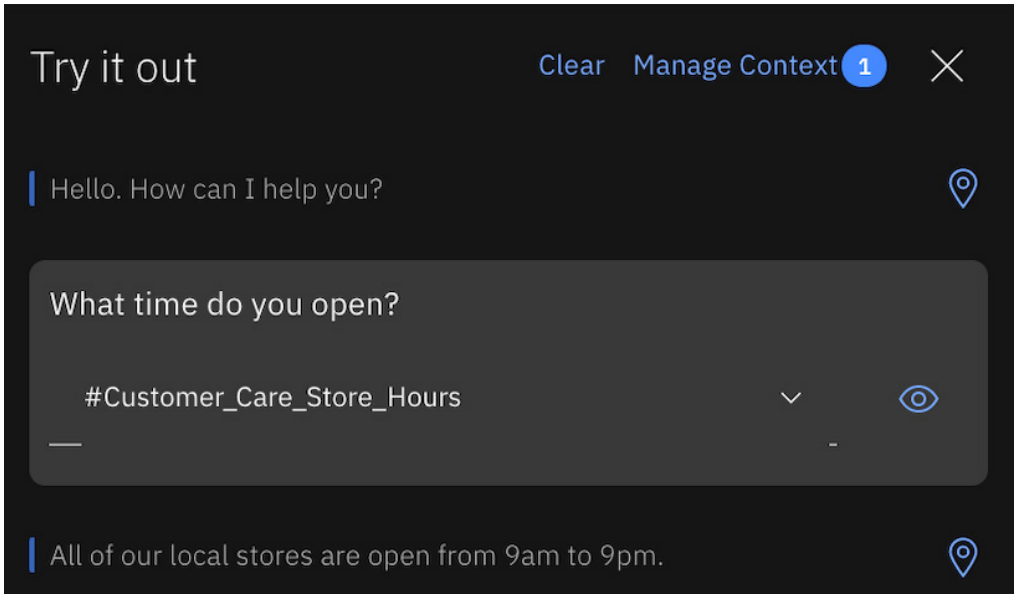


Figure 9 Testing after we have associated intent conditions with dialog responses. The testing interface now shows both the correct intent and the correct dialog response. A successful test!

The assistant is now more useful to users, at least when it comes to three question types (store hours, store locations, and employment inquiries). However, FICTITIOUS INC has two retail locations, one at Maple St and one at Elm Dr. If users ask about a specific store they should get a store-specific response. The current implementation gives the same response for an intent no matter how detailed the user's question gets. The assistant needs to understand more detailed context from the conversation

2.3.2 Contextualizing a response by using entities

Giving a store-specific answer has two parts: first detecting that a specific location was mentioned, and second replying with a specific condition. The specific location will be detected by training the assistant to recognize entities; the entity will be used to give a specific response.

DETECTING AN ENTITY

Since the number of FICTITIOUS INC stores is both limited and well-defined, the simplest solution to use a static dictionary-based entity. We open the Entities tab and create a new entity called `@store_location` with two possible values: "elm" and "maple". We'll include a couple of variations with the street/drive qualifier as well, though these are not strictly required. We will also enable "Fuzzy matching" so that our entity dictionary will be robust in the case of user typos and misspellings.

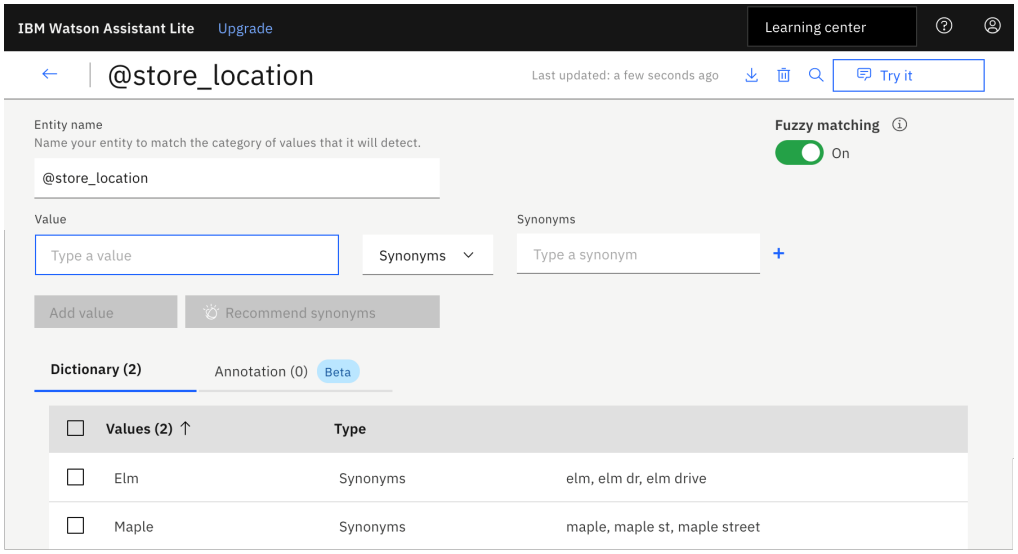


Figure 10 Example interface for defining entities. FICTITIOUS INC uses a @store_location entity with values for their two stores: Elm and Maple.

Terminology alert!

"Fuzzy matching" is a pretty common term! Each of the virtual assistant platforms I surveyed had fuzzy matching. Use fuzzy matching so that your assistant is resilient against typos and misspellings.

USING AN ENTITY TO GIVE A SPECIFIC RESPONSE

Next, we have to update the dialog conditions to check for a mention of one of our `@store_location` values in the user's utterance. We will need to update the `#Customer_Care_Store_Location` node to check for a `@store_location` and give a targeted response if present, and a generic response if not present.

Follow these steps to update the dialog:

1. Find the `#Customer_Care_Store_Location` condition.
2. Within that conditional block, create child conditions for `@store_location:Elm`, `@store_location:Maple`, and another fallback or "anything else" node (for when `#Customer_Care_Store_Location` is detected but no `@store_location` is present)
3. Associate a dialog response to each of the new conditions.

1. First dialog node detects the intent.

2. Defer a response until the child nodes are evaluated for entities.

3. If the user asked about Elm, tell them about the Elm location.

4. If the user asked about Maple, tell them about the Maple location.

5. Otherwise, give the generic #Customer_Care_Store_Location response

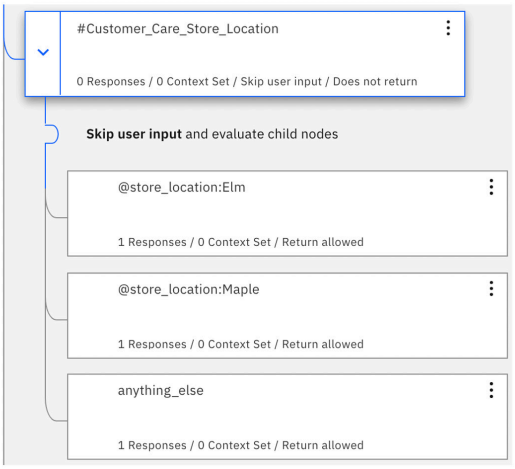


Figure 11 Dialog rules implemented for detecting a specific store location. Three different answers will be given to a "store location" question based on the specific store, or lack or specific store, in the user's utterance. You can also implement this by putting multiple conditions in each dialog, joined by an "and"

This may look different in your virtual assistant platform, but you can still do it!

"Skip user input and evaluate child nodes" is my personal favorite way to implement this pattern. I like having one condition per dialog node which makes each node easier to read. The last three nodes in Figure 11 effectively have two conditions by being children of the first node.

You could alternately write three nodes with these conditions:

```
#Customer_Care_Store_Location and @store_location:Elm
#Customer_Care_Store_Location and @store_location:Maple
#Customer_Care_Store_Location
```

It's a matter of what your platform supports and your personal preference!

In Table 9 you can see that the assistant detects the three separate variations of the store location intent based on the contextual clues.

Table 9 How the assistant interprets three different "store location" utterances when entities are used

User utterance	Assistant detects	Assistant responds
Where is the Elm store?	`#Customer_Care_Store_Location` `@store_location:Elm`	That store is located at 456 Elm Drive.
What's the address for the Mapel store?	`#Customer_Care_Store_Location` `@store_location:Maple`	That store is located at 123 Maple St.
Where are your local stores?	`#Customer_Care_Store_Location`	We have two locations near you: 123 Maple St and 456 Elm Drive.

Figure 12 shows one variation of the store location questions in Watson's "Try it out" interface.

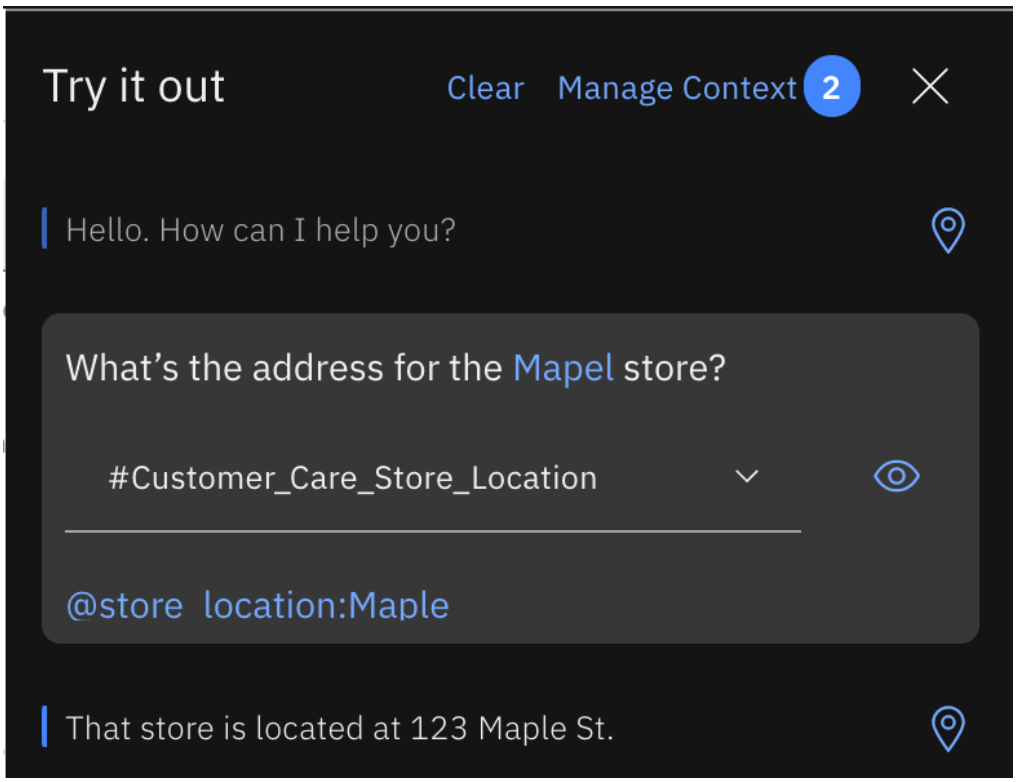


Figure 12 The assistant has been trained to detect specific stores via entities and gives targeted responses. The assistant is even adaptive to misspellings ("Mapel") thanks to fuzzy matching.

The ability to tailor a specific response based on one or more contextual clues is a powerful way to satisfy your end users!

This may look different in your virtual assistant platform, but you can still do it!

For the virtual assistant platforms I surveyed, each provided some way of inspecting entities by telling you the entity type (`@store_location`), value ("Maple"), and "covered text" ("Mapel") in the testing interface.

2.3.3 An alternate way to provide contextual responses

FICTITIOUS INC provided contextual responses to `#Customer_Care_Store_Location` questions through the use of four dialog nodes. The pattern used requires *n + 2* dialog nodes, where *n* is the number of unique entity values. FICTITIOUS INC will also provide contextual responses to the `#Customer_Care_Store_Hours` intent based on store location variations. However, for this intent, they use a different technique called "Multiple conditioned responses"[6].

Multiple conditioned responses allow you to put conditional response logic in a single dialog node. To update this dialog with multiple conditioned responses:

1. Enable multiple conditioned responses on the dialog node using the settings (triple dot) icon
2. Create conditioned responses for `@store_location:Elm`, `@store_location:Maple`, and an "anything else" node (for when no `@store_location` is present)

Coding conventions and style

Your platform may support multiple ways to implement complex conditions. Methods like multiple conditioned responses have the advantage and disadvantage of being more compact. These responses use less screen real estate, allowing you to see more of your dialog nodes on the screen at a time.

Consistency is important in software development. Consistency makes software easier to read and understand. However you choose to implement complex conditions, try to do it consistently!

[6] https://cloud.ibm.com/docs/assistant?topic=assistant-dialog-overview#dialog-overview-multiple

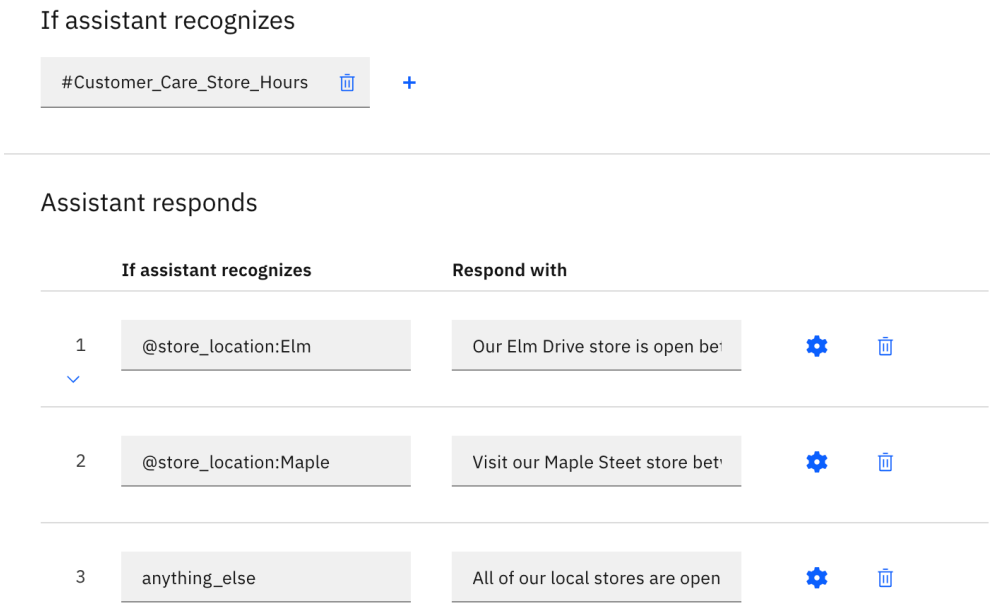


Figure 13 Giving a targeted "store hours" response based on which store (if any) the user asked about. A single dialog node is configured with three different possible responses. This is more compact on the screen but can be harder to read.

We've now seen two possible methods to improve the assistant's responses using contextual information. Either method will satisfy your end users and make them feel understood. When the user asks a specific form of a question ("where's the **Elm** store?") it's important to give a specific response ("The **Elm** store is at…")

Dictionaries seem pretty static – would I really use them to list my stores?

The preceding examples using dictionary-based entities are intended as a simplistic example. A national chain would prompt for zip code and use a store locator service called from an orchestrator to select a store, or get the zip code from the user's profile. The chain would not code hundreds or thousands of entities for each specific store.

A dictionary-based entity works best on a relatively small and finite list of possibilities.

2.3.4 Responding with a process flow

Many self-service interactions require a process flow, or some way of collecting all of the information necessary to complete a task. FICTITIOUS INC will implement a simple process flow for setting up an appointment.

When FICTITIOUS INC first configured the dialog for their assistant, they only created static responses. The response for `#Customer_Care_Employment_Inquiry` was a redirection to a website ("Apply on FictitiousInc.com/jobs now!). Instead of doing a similar redirection for `#Customer_Care_Appointments`, the assistant should actually book an appointment. Booking an appointment has a process flow – an appointment consists of a date, time, and specific store location.

FICTITIOUS INC can implement this process flow in one dialog node using a capability called "Slots". They will also use built-in entities (called System Entities) to gather date and time of the appointment.

Terminology alert!

Each virtual assistant platform I surveyed at the time of this writing had "Slots" capability though some gave it a different name like "parameters and actions". Most virtual assistant platforms fill slots with "entities" though some call them "slot resolution"

Think of "slots" as something that must be filled before a request can be worked on.
Setting an alarm has one slot (the time of the alarm)
Setting an appointment may have three slots (the date, time, and location of the appointment)

Virtual assistant platforms also usually come with "system" or "default" entities to cover basic scenarios like detecting dates. (Building your own date detection is fiendishly difficult, be glad your platform does it for you!)

Check your virtual assistant platform's documentation for how slots are implemented. Slots are a cool feature and platform providers can't help but brag about it!

Figure 14 shows how the appointment flow can look in a virtual assistant platform that uses slots. While the specific steps may vary slightly in other platforms, the Watson Assistant steps are listed here:

1. Enable Slots on the dialog node using the settings (triple dot) icon
2. Enable System Entities for sys-date and sys-time
3. Create conditions for each of the slot variables: how to detect them, where to store them, and how to prompt for them.
4. Create a text response when no slot variables are present
5. Create a response when all variables are filled in

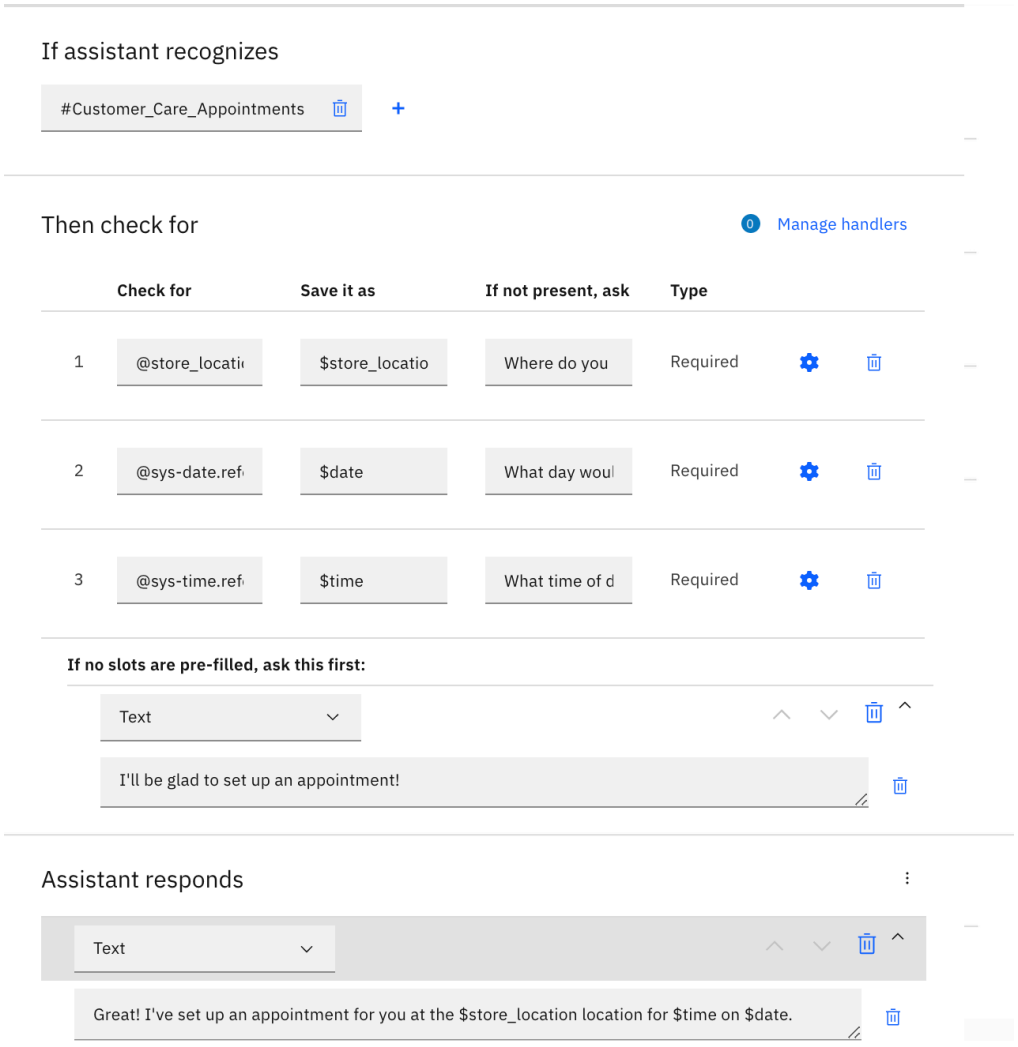


Figure 14 Configuring dialog to set up an appointment using slots. The dialog flow has slots for collecting the appointment location, date, and time. The user will be asked follow-up questions until all the slots are filled since an appointment cannot be scheduled without all three pieces of information. Your virtual assistant platform may have a different interface but probably still has slots!

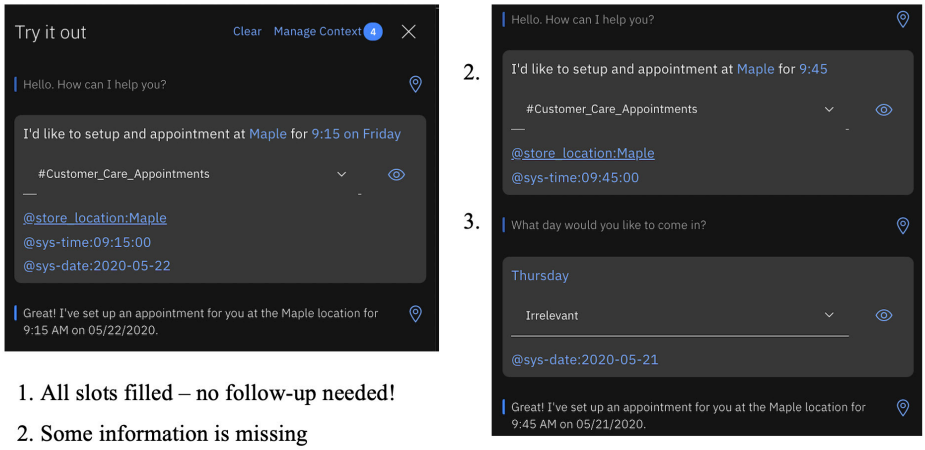
Now that the dialog engine is coded with these conditions it can respond with a message reflecting all of the appointment details, such as: "Great! I've set up an appointment for you

at the `$store_location` location for `$time` on `$date`." Your assistant will fill in the details dynamically as shown in the "Try it out" panel in Figure 14.

Technique alert!

Your platform will let you use variables in your responses, just check the syntax from your provider's documentation. Common patterns are $ plus a variable name (`$time`) or to put the variable name in braces (`{name}`). Search for your platform's documentation for "variables" or "parameters" and you should find what you need.

Figure 15 shows two different examples of conversational flows using slots. In the first example, the user supplies all three pieces of information in a single statement: "I'd like to set up an appointment at **Maple** for **9:15** on **Friday**". Since all information has been provided the system responds that the appointment has been set up. In the second the user's statement "I'd like to setup an appointment at **Maple** for **9:45**" does not include a date, so the system prompts the user "What day would you like to come in?".



1. All slots filled – no follow-up needed!
2. Some information is missing
3. Follow-up question to fill empty slot

Figure 15 Two conversational examples with slots. In the left-hand example, all information is provided in the initial statement. In the right-hand example, the assistant asks follow-up questions to fill all of the required slots. This capability is common across major virtual assistant platforms!

The example ends with a message printed out to the user: "Great! I've set up an appointment for you at the $store_location location for $time on $date." I used Spring Expression Language (SpEL) expressions to reformat the date and time to a more user-friendly representation, rather than the internal date format.

Platform variation!

Watson Assistant uses SpEL[7] for "low-code" manipulation within the virtual assistant dialog response. Your platform may require you to use "no-code", "low-code", or just plain code in your responses. Check your platform documentation on how to customize dialog responses with system variables.

Listing 1: The SpEL logic used in the Watson Assistant date formatting example

```
{
  "context": {
    "date": "<? @sys-date.reformatDateTime(\"MM/dd/yyyy\") ?>"
  }
}
```

Listing 2: The SpEL logic used in the Watson Assistant time formatting example

```
{
  "context": {
    "time": "<? @sys-time.reformatDateTime(\"h:mm a\") ?>"
  }
}
```

This appointment flow ends with a dialog response to the user, telling them about their appointment, but it does not actually book the appointment. Booking an appointment would require an API call to an external booking system, coordinated by the orchestration layer. For the sake of brevity, this was not demonstrated.

Booking an appointment can be slightly more complex!

We have walked through an illustrative example of appointment booking. A more robust example would include calling an appointment service API which would check availability, book the appointment, and return a success or failure response. Your platform may enable this – look for "webhooks", "orchestration", "connectors", "integration", or anywhere you can write code!

Slots are a useful way to implement a process flow that needs to collect multiple pieces of information before proceeding. Slots provide a syntax shortcut for gathering multiple input variables in a coordinated way. You can choose to implement process flows with or without slots. Anything you could implement in Slots could also be implemented in regular dialog nodes.

[7] https://cloud.ibm.com/docs/assistant?topic=assistant-dialog-methods

Further discussion on conventions and style

As in the entities example, there are multiple ways of implementing process flows. You may wish to gather "slot" variables one-by-one and do an external validation. For instance, when authenticating with two pieces of information like user ID and date of birth, you may want to validate the user ID exists in the system before prompting for date of birth, and you'll want to execute a resolution flow if the user ID is not correct.

2.4 Other useful responses

FICTITIOUS INC's assistant now has the following capabilities:

- Greeting the user
- Providing a static, non-contextual answer for employment inquiries
- Providing static, contextual answers for store location and store hour requests
- Executing a process flow to set up appointments
- Responding with "I didn't understand" to any other utterances

Before FICTITIOUS INC accepts this demonstration they need two more capabilities:

- Determining when the assistant is unsure about the user's intent
- Handing the user off to another channel when the assistant repeatedly misunderstands the user

Three strikes and you're out!

Virtual assistants often use a metaphor from baseball. When a baseball batter swings and misses it's called a "strike". If the batter accrues three strikes before hitting the ball, they're out!

Similarly, if a virtual assistant misunderstands the user, we can call that a strike. If the assistant misunderstands the user three times, the assistant should hand the user off to get their resolution elsewhere.

Many virtual assistant implementers use a "two strikes" rule with similar logic.

Let's update the assistant with a two-strikes rule. We'll implement two pieces of logic in the dialog nodes:

1. Detecting low confidence conditions from the classifier and routing them to the "Anything Else" node.
2. Counting how many times the "Anything Else" node is visited.

2.4.1 Detecting low confidence

Most virtual assistant classifiers not only predict an intent but also tell you the confidence in that prediction. By default, they only return an intent if the best prediction has confidence over some threshold (for example, this threshold is 0.2 in Watson Assistant). Intent predictions with low confidence values are frequently incorrect and you may need to tune this threshold for your particular assistant.

In Chapter 7 we will see how the confidence value is derived. For now, just consider that the lower the confidence value is, the more likely the predicted intent is wrong. Figure 16 shows how you can inspect the confidence values in a virtual assistant testing interface.

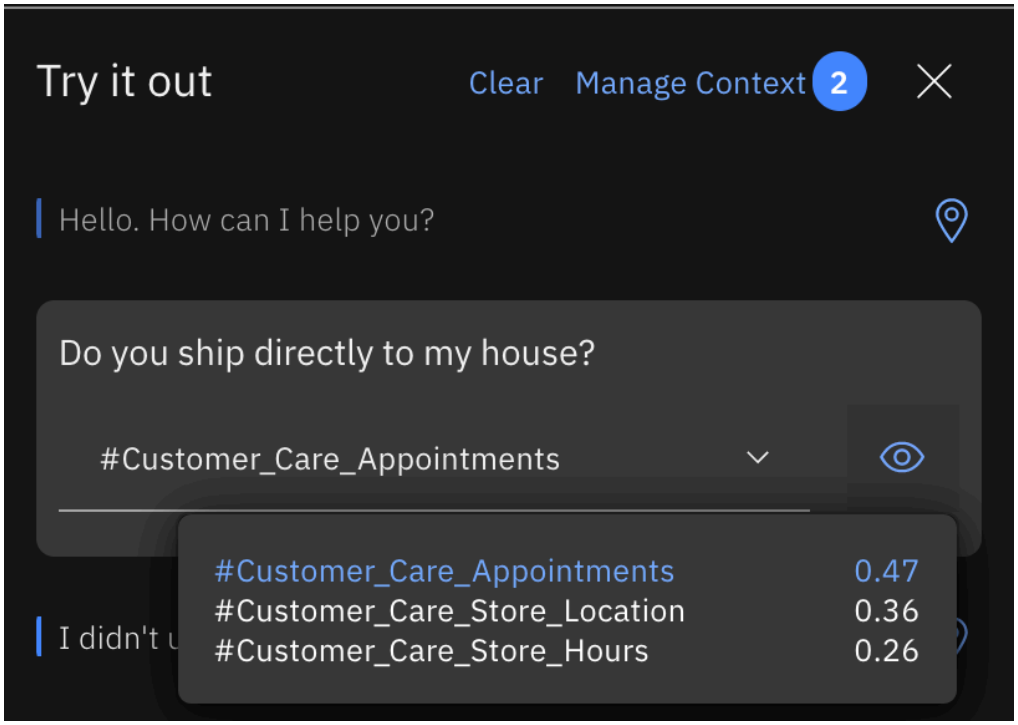


Figure 16 The most likely intents and a confidence value associated with each for an input utterance. Note that low confidence results are often wrong. The utterance "do you ship directly to my house" does not match ANY of the intents coded into this assistant.

The confidence values can be used in conditional logic in dialog nodes.

Listing 3: Conditional logic for low-confidence intent selection in Watson Assistant

```
intents.size() > 0 && intents.get(0).confidence < 0.5
```

Terminology alert

Each platform I surveyed had a "confidence" term. Hooray again for common terminology!

The platforms differed in how they referred to the threshold and how you configure it. The threshold is also called "confidence cutoff", "intent threshold", or "classification threshold", though is still described as a threshold.

Most virtual assistant platforms have a single setting for this classification threshold to apply it globally to your assistant. Some platforms let you use a different (higher) threshold in specific dialog logic.

2.4.2 Counting misunderstandings

When the dialog reaches the catch-all "Anything else" node, this implies the user was not understood, and their satisfaction with the assistant will decrease. Therefore, it's a good idea to count how many times this dialog node has been visited. The second time the catch-all node is visited we should respond by offering the user an alternate channel before they get really frustrated.

Most virtual assistant platforms come with a way to store arbitrary information in variables during a conversation. These are often called "context variables". We'll use a context variable to count how many times the catch-all node is visited.

Some platforms automatically count the number of visits to the catch-all node

Be sure to check your platform's documentation to see if it has this feature – it will save you some time!

The two-strikes rule is thus implemented by storing the number of visits to the catch-all "Anything else" node in a context variable. Two strikes (two visits to this node) and the assistant should send the user to another channel such as a customer service phone line, website, or another human-backed process.

FICTITIOUS INC wants to be conservative with their first assistant. They only want the assistant to respond to a user's intent if that intent is detected with high confidence. Thus, their assistant should treat low-confidence predictions as misunderstandings. Low confidence predictions should be sent to the "Anything else" node. Let's see how to implement these two features in the assistant.

2.4.3 Implementing confidence detection and the two-strikes rule

This functionality is implemented via the following steps:

1. Initialize a context variable `misunderstood_count` to 0 in the `Welcome` node.
2. Create a new "Misunderstanding" node at the bottom of the dialog with the fallback condition.
3. Increment `misunderstood_count` inside the "Misunderstanding" node
4. Add a child node to "Misunderstanding" checking the misunderstood count, and if `$misunderstood_count >= 2` deflect the user with the message "I'm sorry I'm not able to understand your question. Please try our phone support at 1-800-555-5555."

5. Moving the existing "Anything Else" node as the last child of the "Misunderstanding" node.
6. Create a "Low Confidence Prediction" node before all the other intent nodes with the "low confidence" conditional logic (`intents.size() > 0 && intents.get(0).confidence < 0.5`). When this node fires, it should "go to" and evaluate the "Misunderstanding" node conditions.

The code is visualized in Figure 17. The code listings are also shown below the figure.

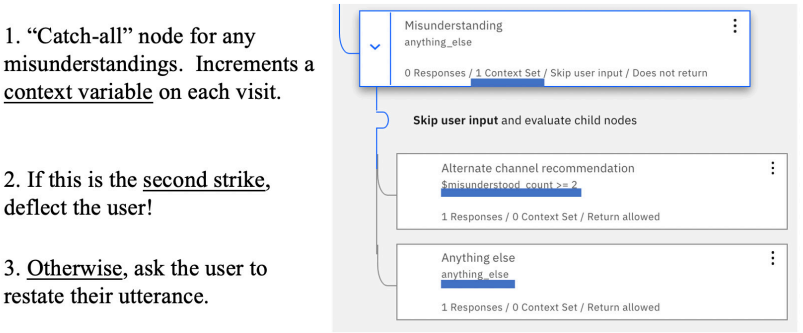


Figure 17 Visualization of two-strikes logic. This will look different in your virtual assistant platform of choice but should still be implementable.

Listing 4: Watson Assistant code for the Welcome node: initialize the misunderstanding counter

```
{
  "output": {
    "generic": [
      {
        "values": [
          {
            "text": "Hello. How can I help you?"
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  },
  "context": {
    "misunderstood_count": 0
  }
}
```

Listing 5: Watson Assistant code for the catch-all node: increment the misunderstanding counter

```
{
  "output": {
    "generic": []
  },
  "context": {
    "misunderstood_count": "<? $misunderstood_count + 1 ?>"
  }
}
```

A demonstration of the "two strikes" rule in action is found in Figure 18.

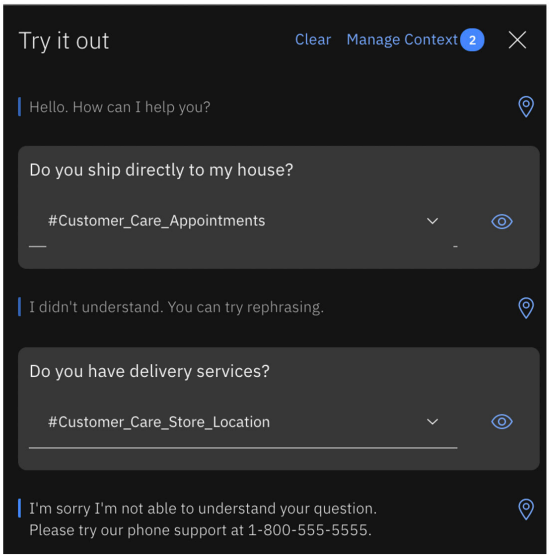


Figure 18 The assistant is asked two questions. For each one, it detects an intent with low confidence and visits the catch-all node. After two misunderstandings, the user is deflected to call customer service.

In Figure 18, the assistant selected an intent for each user utterance. The first utterance, "Do you ship directly to my house" was classified as `#Customer_Care_Appointments` but with low confidence. The second utterance "Do you have delivery services" was classified as `#Customer_Care_Store_Hours` but again with low confidence. Because of the Low Confidence Response dialog node, the low confidence condition fired before any of the intent-based nodes, and thus the system does not respond based on a (possibly wrong) intent selection.

How can you know the intents were recognized with low confidence?

Watson Assistant will show you the confidence behind an intent when you hover over the "eye" icon next to the identified intent. The algorithms used by virtual assistant providers are constantly changing, but if you try out this example you should see low confidence values, probably between 0.4 and 0.5.

Your virtual assistant platform will provide a way to see the intent confidence either through a testing interface or API.

2.5 Try it out!

The principles demonstrated in this chapter will work across most virtual assistant platforms. I have provided screenshots, terminology guides, and the reasoning behind each concept and example so that you can implement them in your virtual assistant platform of choice. Some of the low-level details are specific to Watson Assistant but I am confident you can implement FICTITIOUS INC's assistant in any virtual assistant platform.

The Watson Assistant source code is available in this book's GitHub site at https://github.com/andrewrfreed/CreatingVirtualAssistants. The file is `skill-Customer-Service-Skill-Ch2.json` and you can import it into your Watson Assistant instance. The JSON format used is not human-friendly and is best read from the Watson Assistant interface.

Try expanding the capabilities of the assistant on your own! For a first exercise try responding to `#Customer_Care_Contact_Us` questions by giving out a phone number and an email address. Next work on a multi-step response for `#Customer_Care_Open_Account` inquiries. What other capabilities would you like to add to the assistant?

2.6 Summary

- User input to a virtual assistant is called an utterance. An utterance contains an intent (centered around a verb) and/or entities (centered around nouns).
- A virtual assistant responds to a user by first extracting meaning from the user's utterance and then mapping that meaning to a response.
- The simplest virtual assistant response is to provide an answer to the user's utterance and wait for the user to send another utterance.
- A virtual assistant can execute a process flow based on a user's utterance. That process flow will dictate additional questions or steps in the conversation that needs to take place before an answer can be provided to the user.

3

Designing effective processes

This chapter covers:

- Organizing a set of user needs according to complexity and frequency and prioritizing them into a milestone plan.
- Identifying all steps required to fully automate a process flow, then identifying steps that can be removed or replaced in a minimum viable product.
- Assembling your dream team to implement a virtual assistant.
- Recognizing parts of a process flow that will be challenging in a web channel, or a voice channel. Devising a flow that solves for the challenges and optimizes for that channel.
- Assembling a rich dialog response for a web channel and converting this to a suitable voice response.

A virtual assistant is more than just a classifier and some orchestration logic, in the same way that a graphical user interface is more than just lines and styling. A virtual assistant is a complete user experience channel. It requires a careful and thoughtful design to satisfy end users. The quality of your design sets an upper limit on how successful your virtual assistant can be.

This chapter focuses entirely on virtual assistant design aspects, setting aside (almost) all machine learning and coding considerations. After reading this chapter you will be able to design process flows for your virtual assistant optimized for your preferred deployment channel and you will be able to decide what processes you should implement first. You'll be ready to learn how to write the dialog supporting your new processes. Figure 1 shows the parts of your assistant that the user sees – but the "invisible" parts need to work well for a good user experience.

A virtual assistant should be optimized for the channel it is delivered on. Assistants are commonly deployed in voice channels (i.e. telephone) and web channels (i.e. websites).

Voice and web have different strengths and weaknesses and your design should account for these. A good response for voice may not be a good response for web chat and vice versa.

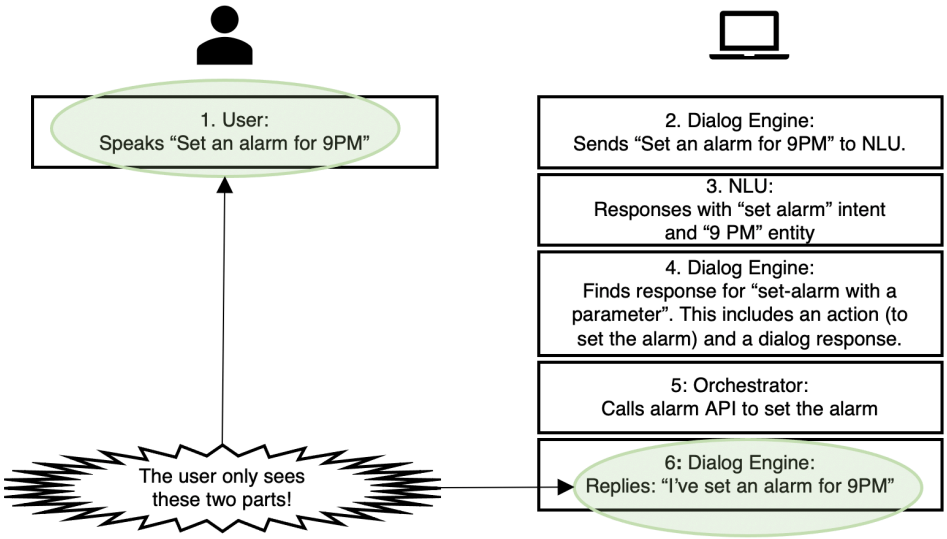


Figure 1 Motivating example. The user only experiences what they say and how you respond. The user cannot see the internal parts. The parts invisible to the user need to be well-designed so that the user has a good experience.

Regardless of channel, it is important to design a virtual assistant that can be useful without requiring a "boil the ocean"[1] approach. You should brainstorm on all the things that your assistant could do and then build an iterative plan to implement them. High-frequency and low-complexity requests are the first things an assistant should cover. ("What time do you close?" is a high-frequency, low-complexity question in retail.) As the frequency of a request decreases or its complexity increases you need to decide an appropriate strategy - is there a simple strategy for the request, or should it be escalated out of the assistant?

In this chapter, I will demonstrate how to prioritize a list of user requests into a phased implementation plan. Then, I will help you assemble your "dream team" to build your own virtual assistant solution. Next, I will demonstrate key differences between the voice and web channels, and show how to optimize your assistant to respond with questions and answers that will work well for a given channel.

3.1 What processes will the assistant handle?

In order to build a virtual assistant that delights your users, you must have a solid understanding of what your users want from the assistant and how you can provide it. You need to have business processes that your assistant should follow – and if these don't exist

[1] "Boiling the ocean" refers to an impossibly large scope. Consider how much energy it would take to boil the ocean versus to boil a gallon of water.

you will be creating them! For example, if you don't have a repeatable business process for resetting a user's password, how can you automate it in a virtual assistant?

As you design business processes and adapt them for a virtual assistant you may find challenges that require you to use a phased approach in delivering functionality. Processes often have a "shortcut" version (for example, tell the user where they can go to reset their password) and a fully automated version (ask a series of questions verifying the user and then resetting their password for them).

When you work out a plan for which processes to implement in your assistant first, make sure you keep the user in mind.

3.1.1 Designing for the most common user needs

When building a virtual assistant you should not try to boil the ocean but rather focus on high-volume and high-value activities. This is particularly true for your first venture into virtual assistants. You should do an assessment of what information and processes you already expose to your users through other channels.

You can review call logs, search queries, interview subject matter experts, or more. If you have a call center - shadow them for a day and see what kinds of calls they receive. Your business may already keep track of the top customer demands, and if so you can use that information repository to guide this exercise. Many companies do not keep detailed records of what types of customer interactions they have and how frequently, and for them a major benefit of virtual assistants is to start collecting that information. Note that the closer you are to real user data, and the farther away you are from opinions, the better you will do in this exercise. In Chapter 6 we introduce an additional methodology for gathering user input.

In this section, we'll develop an implementation plan for a customer service virtual assistant for FICTITIOUS INC. We'll reference the Watson Assistant Customer Care Content Catalog[2] which contains a list of intents covering frequent customer service request types. These are shown in Table 1. These intents are fairly generic for a retail customer service use case.

Table 1 Customer Care intents, covering frequent customer service request types

Schedule/Manage Appointments	Reset Password
Add/Remove Authorized Users	Change Security Questions
Cancel Account	Get List of Programs
Get Contact Information	Redeem Points
Apply for a job	Report Fraud
Check loyalty status	Store Hours
Adjust Notification Preferences	Store Location
Open Account	Transfer Points
Get List of Products	Update User Profile

[2]https://cloud.ibm.com/docs/assistant?topic=assistant-catalog

We need to develop a plan for FICTITIOUS INC. We have the list of their most common request types above. Now we must approximate the complexity behind satisfying each type of request. One can use a simple low-medium-high scoring, or a t-shirt sizing (small-medium-large). The goal is to quickly sort the types of requests into a 2x2 grid where the dimensions are formed by complexity and frequency. An empty grid is supplied in Figure 2. Try placing each of the request types from Table 1 into the grid for FICTITIOUS INC.

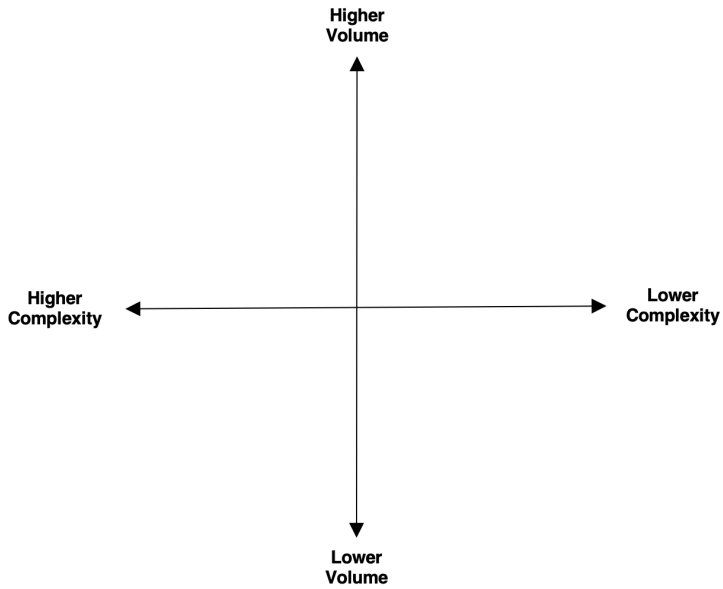


Figure 2 Empty prioritization grid. Try placing the intents from Table 1 into this grid!

Gauge complexity by what it would take completely and automatically satisfy the request according to your business processes. Many request types can be satisfied with either a simple question and answer response or a complete process flow. Consider different possible versions of the "open account" flow.

Simple (Redirection) flow for "Open Account"

```
Question: "I'd like to open an account"
Answer: "Please call our service line at 1-800-555-5555 and press 2 to open an account, or
       go to our account creation site at [link]"
```

Fully automated flow for "Open Account"

```
Question: "I'd like to open an account"
Answer: (A process flow with these steps)
1. Gather identifying information
2. Determine account type
3. Use account query API to see if an account already exists
4. Use account creation API
```

You can always start by addressing a request via redirection or giving a purely informational response, then later automate more of the response in future versions of your assistant. A virtual assistant that always redirects may not provide the best user experience, but it can still be helpful to users. For the sake of this planning exercise, let's focus on the fully automated response first.

"Open account" requests are one of the most common request types for most retailers. This request type definitely belongs on the top half of the grid. As shown above, creating an account requires several follow-up questions and several API calls. That's enough complexity to earn a spot in the upper-left quadrant (high volume, high complexity).

Requests for your contact information, such as for your phone number or mailing address or email address, are also high-volume requests. These requests are simple to address: just return a static answer and the request is satisfied. We can place "Get Contact Information" in the upper-right quadrant (high volume, low complexity). Try placing all of the FICTITIOUS INC requests from Table 1 into a grid. When you're done, review Figure 3 which contains my analysis of the FICTITIOUS INC intents.

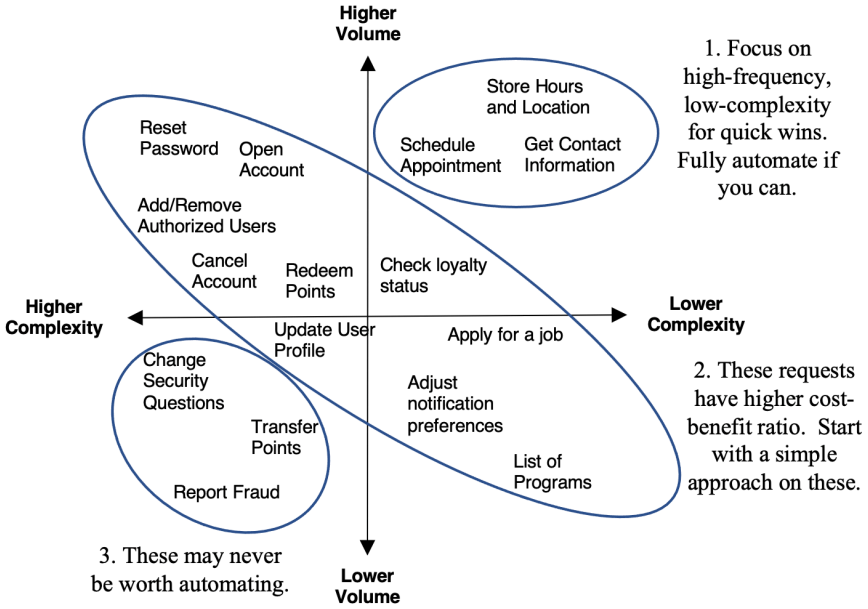


Figure 3 Volume and complexity analysis of customer care request types from Table 1. An implementation plan must be based on cost-benefit analysis. Focus first on high-volume, low complexity tasks.

Based on the plot of frequency and complexity you can perform a cost-benefit analysis. That analysis shows you what to implement first in your virtual assistant. The highest return on investment comes from the high-frequency, low-complexity requests, and you should handle as many of these as you can.

Next are the high-frequency, high-complexity request types. For these, you need to determine if there is a useful quick win to satisfy the requests (can you provide a link or general answer?) or do you need to escalate these types of requests to an alternate channel like a human-staffed call center. You will need to train your assistant to recognize these requests even if the assistant does not service them.

As time permits you can address the easiest requests in the low-frequency, low-complexity segment. Take care not to spend too much time and introduce too much virtual assistant development work for these.

The low-frequency, high-complexity requests may never be appropriate for an automated solution and you can use a generic escalation strategy on these. A generalized plan is shown in Figure 4.

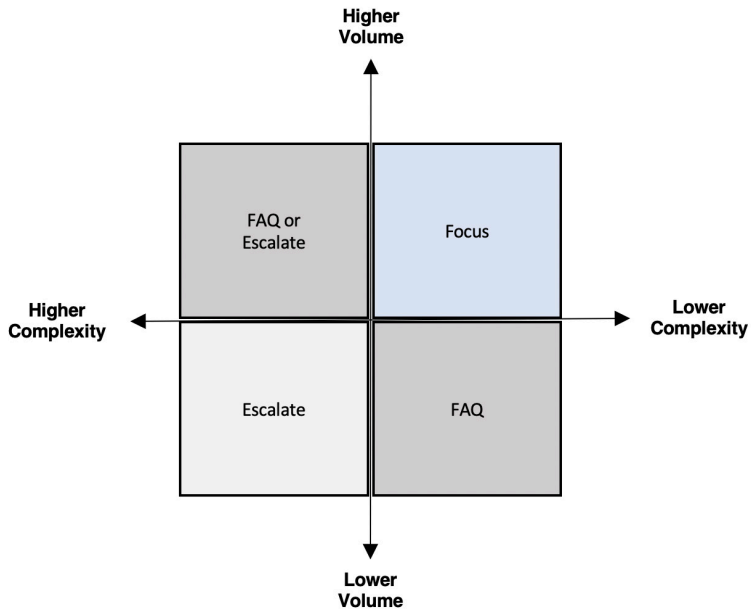


Figure 4 A generalized plan for requests based on volume and complexity

3.1.2 Assemble a plan and a dream team

When you review the request types on the frequency/complexity graph you can sub-divide each request type into the processes and APIs that support it. The first common process may be authenticating a user. Other commonalities may include high volume "store hours" and "store location" requests as both are likely to reference a "store locator" API (since each store likely has different operating hours). You can also plot any APIs on your graph to influence what you implement first, as shown in Figure 5.

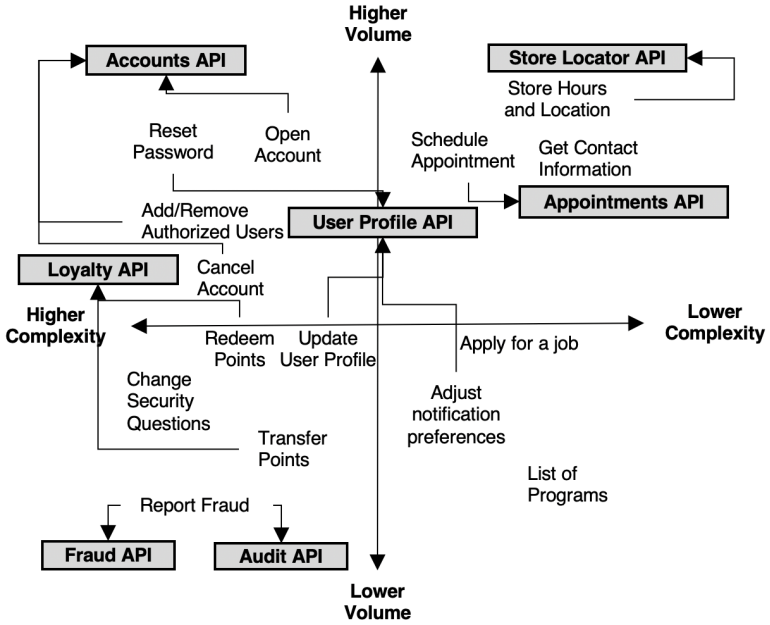


Figure 5 Overlaying the APIs required to fully automate an intent. The more intents that use an API the more valuable it can be to integrate with that API. The User Profile API is a good candidate to integrate with since it supports so many common request types.

For your first virtual assistant, a good plan is to implement a maximum of two or three process flows and a series of question and answer responses as in Frequently Asked Questions (FAQ). You should train the assistant to recognize the high-frequency request types even if it will not answer them directly. You should use an incremental development process to build and expand the virtual assistant. Each significant milestone can focus on a small number of process flows and expanding the number of request types recognized.

Table 2 Example delivery milestone plan for FICTITIOUS INC

Milestone	Features delivered in the assistant
1	• Give store hours & location answers for user's local store • Provide FAQ-style answers for ~10 more intents including account management, the loyalty program, appointments, and job listings • All other inquiries are redirected to another channel
2	• Fully automated appointment scheduling • Fully automate password resets • Detect intents about fraud and redirect to a special Fraud Hotline

For production-grade virtual assistants, you'll need a multidisciplinary team to implement this plan. The team will design your virtual assistant and the process flows it will use. The team will also work together to develop and deliver the assistant according to time and budget. Figure 6 shows what your dream team might look like.

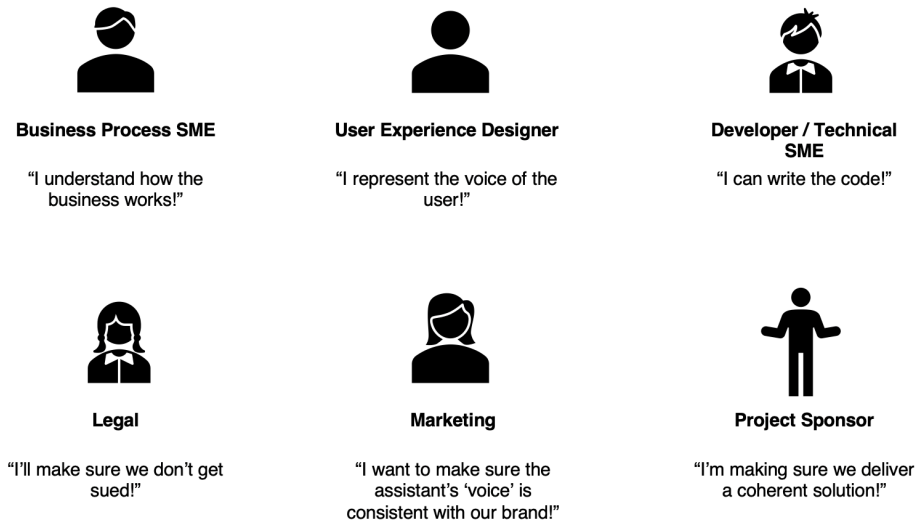


Figure 6 This is the dream team for delivering a successful virtual assistant!

That's a huge dream team – can't I just build an assistant myself?

You can certainly build a prototype or proof of concept virtual assistant by yourself, sometimes in a matter of hours. There's more to virtual assistants than just throwing together some technology. Most commercial-grade virtual assistants require insight from a multi-disciplinary team for both the sake of the project and the company. But, if you are willing to perform multiple roles there's no reason that you can't build a virtual assistant completely by yourself.

Depending on your use case and business processes your team should include the following skills and roles:

- **Business process subject matter experts**: You need people who understand the business and the current as-is scenario. These experts should have a deep understanding of current processes, and ideally will have been a part of creating and shaping those processes. They will provide critical guidance on how conversations can and should flow within a business process.
- **User experience designer**: A UX designer is the chief advocate for the user and makes sure the experience works well for users. The designer provides valuable advice on how users will interact in a given channel. They will make sure the conversation design delivers what the user wants, not just what you want!

- **Developer / Technical subject matter experts**: The developer will code the dialog rules into the virtual assistant. Many virtual assistants will include some sort of technical integration into existing resources, potentially including a website, a telephony system, an authentication mechanism, or any number of APIs. Technical experts are needed to make sure the conversation is technically possible, such as making sure any parameters required for API calls are gathered during a conversation.
- **Legal**: This team makes sure legal and regulatory concerns are addressed in the assistant. For instance, you may need to read out a disclaimer before providing sensitive or rapidly changing information through the assistant.
- **Marketing**: They will help manage the tone and some content in the virtual assistant. Since the assistant can be another public-facing channel it should stay in line with the existing tone, branding, and textual copy from other channels.
- **Project executive sponsor**: The sponsor should provide clear direction to the desired future-state and keep the team focused and working together. The sponsor can also adjudicate any conflict resolution between the team.
- **Analytics experts** (optional): If you are running analytics on your other business processes you will want some insight so you can design the virtual assistant to support similar analytics. This will allow you to do an apples-to-apples comparison of your new virtual assistant to the existing process. This is marked as optional since you may not need this from Day One, but it is easier to design in the analytics from the beginning rather than bolting them on at the end. Your business process subject matter experts may be able to cover this role.

3.1.3 Managing the design process

The multidisciplinary team should work together to document and build the process flows the virtual assistant will follow. If your organization has a call center, you may already have a "decision tree" or "call script" that your agents follow. Your first goal is to document all of the steps in the process. You'll need multiple team members to contribute: one team member may know every last detail about the process, another may know which systems and APIs support a given step, another may see how the new assistant channel can improve the user experience.

Let's revisit the "reset password" process flow example from Chapter 1. The team may note that password resets have the following requirements:

- Determine who needs the password reset by user ID.
- Authenticate that user by date of birth and a security question.
- Reset password only after successful authentication.
- No other authentication options are provided. If authentication fails, do not reset the password.

These requirements can be converted into a process flow as depicted in Figure 7.

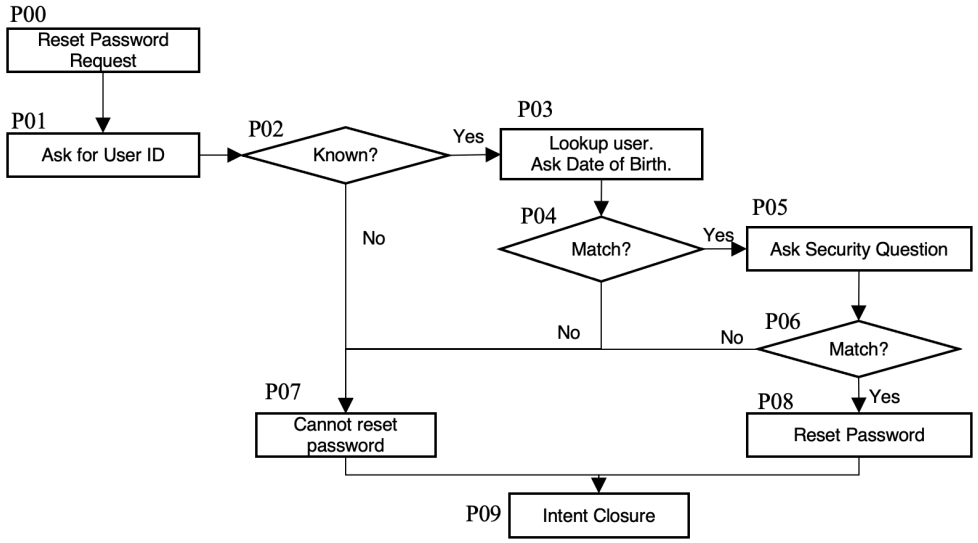


Figure 7 Example business process for resetting a password. You can quickly refer to "P05" rather than "the spot where we ask the security question"

In the process flow, it is useful to label each point in the process flow with an identifier as shown in Figure 7. An identifier makes it easier for each team member to be explicit about what part of the process flow they are working on. The team implementing the virtual assistant can label the nodes with the identifier, the copy editor can supply text for each identifier, and it's overall much easier to say `P04` than saying "the dialog node after we ask for the date of birth"

You can diagram your process flow in any software you wish. Some teams prefer to use an old-fashioned text editor or a spreadsheet tool. I prefer to use a drawing program where you can "see" the flow right in front of your eye. With a drawing program you can even print out the process flow and hang it up in your team room or whiteboard.

Whatever software you use to create the process flow you should treat the design artifact as a first-class output. Make sure the latest approved copy is readily available in a shared location that the entire team has read access to.

The first version of the script should be developed and approved by the entire team. No matter how good that first version is, you will need to make updates and it's important to have a good change management process in place. A sample process is shown in Figure 8.

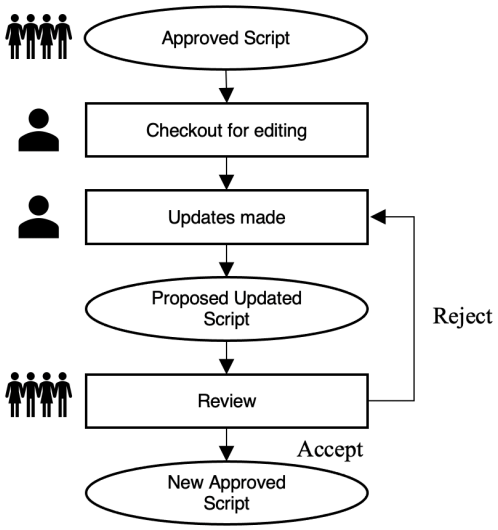


Figure 8 Example dialog script update process for a team

You should use a "lock" or "checkout" process so that everyone knows who owns the "edit pen" - who is making changes. This is especially true if your script is written in software that does not allow collaborative editing, or has poor merge capability. With locking/check-out practices, everyone on the team can make suggestions for the script but there should only be one editor at a time.

The editor should always:

- Work from the latest approved script
- Notify team members that the script is being edited
- Make one or more changes
- Summarize the changes made
- Distribute the updated script for review
- Store the approved script in a location accessible by the team

By following this process, the editor makes sure the team is on the same page. Ideally, the updates are made during a collaborative design session with the team. This ensures that the updates will not cause a business process problem, or a technical implementation problem, or a user experience problem, or a legal problem.

Additionally, this process helps keep version control sane. Most process flows are created in drawing programs and stored in binary file formats. For all but the simplest file formats, the merging and reconciliation is arduous or impossible. You may never get back to a single copy that has everyone's changes! Figure 9 shows how process flows can get out of sync.

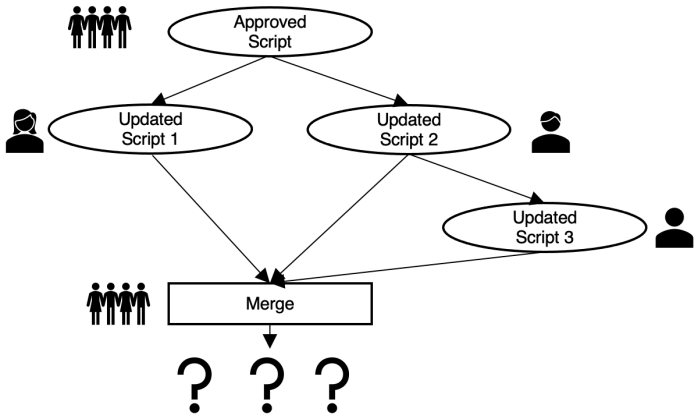


Figure 9 Chaos ensues when team members make independent copies of the script! Merging flow charts can be very difficult!

One significant source of process flow changes comes when you start using or testing the process flows. While you are diagramming a process flow be sure to ask "what if?" questions that test the robustness of your processes. The earlier you think about "what if" questions, the stronger your process flow will be.

Exercise: Test your process flow!

It is much cheaper to change your process flow while it is "just a diagram". After you put a process flow together, try it out. Does it work?

The reset password flow chart in Figure 7 may work well for the majority of cases, however there are several questions raised by this chart.

- Can we help the user find their user ID?
- What if we can't find the user in our backend system by their user ID? (How many times should we ask them to try again?)
- What if the user has not set up security questions?
- Do we need to log failed (or successful) attempts?

Have your dream team review all process flows before you implement them!

3.1.4 Cross-cutting design aspects

There are additional aspects to consider besides volume and complexity when exposing business processes through a virtual assistant channel. Cross-cutting aspects include concerns that cut across all request types, such as:

- Authentication

- Customer Relationship Management
- Legal or Regulatory

Cross-cutting design aspects may need to be built into your assistant from Day 1. If so, that may reduce the number of process flows you can implement in the first version of your assistant.

AUTHENTICATION

The most common cross-cutting concern is authentication. Many business processes can only be executed once you know who the user is and have validated that they are who they say they are. Many APIs that you want to call from your assistant will also require that the user has been authenticated.

Integrating a virtual assistant into your organization's security and authentication routines may be a complex task. There may be a long lead time to getting the right IT resources and approvals together.

Many organizations do not include authentication in the first version of their assistant. Their Milestone 1 includes only unauthenticated process flows or static question-and-answer responses. In Milestone 2 they add authenticated process flows.

CUSTOMER RELATIONSHIP MANAGEMENT

Related to authentication may be a corporate dictate to store all customer interactions in a Customer Relationship Management (CRM) software system. The minimum CRM integration is that when the assistant finishes a conversation with a user, the conversation transcript is stored in the CRM system and associated with that user.

A CRM system generally tracks additional structured fields. FICTITIOUS INC may have a CRM system that tracks every call with the user calling, their reason for the call, and a summary of the call.

The assistant could integrate with the CRM software by storing the following fields as structured data:

Table 3 Example mapping between CRM fields and virtual assistant data

CRM field	Virtual Assistant Data
User	User ID
Reason for Call	Name of first intent
Call Summary	Entire chat transcript

CRM software can help you design your virtual assistant!

The "Reason for Call" or "Request Type" field in your CRM software is a great clue for what types of requests your virtual assistant should handle. You can use your CRM data to empirically determine which request types are the most common.

Further, if your CRM software has "required" fields for data, your virtual assistant is probably required to collect that data as well!

LEGAL OR REGULATORY

If some request types have legal or regulatory ramifications, you may decide that your assistant should never handle them directly. The assistant can instead redirect the user to another answer source like a website or telephone hotline.

FICTITIOUS INC's legal team weighed in on the "report fraud" intent and noted over three dozen regulatory nuances that must be met during an interaction about fraud. The team decided this is too complex and risky to implement in their virtual assistant. In their assistant, they will immediately redirect all fraud-related requests to a human-staffed call center.

If you work in a heavily regulated domain, be sure your legal team weighs in on your virtual assistant design! You want to find out as early in your design process as possible what rules and regulations your assistant needs to abide by.

3.2 Choose the channel you will implement first

The choice of a channel has far-reaching considerations into how your virtual assistant works. Each channel has pros and cons and you will likely need to customize your virtual assistant to exploit the benefits of a channel while avoiding the pitfalls. The specific channel you already use - voice or web - can also influence what you can go after first. Additionally, some business processes may be friendlier to adapt to either voice or web. For example, you can give driving directions with a map in a web channel – how would you give directions over voice?

Today's consumers are used to getting the information they want, when they want it, in the mode that they most prefer, as shown in Figure 10.

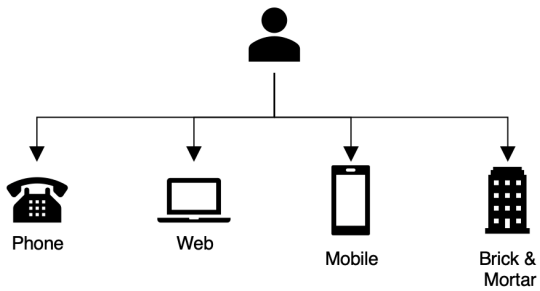


Figure 10 Consumers are used to picking the channel that works best for them

Which channel has the closest affinity to most of your users? If your users are on the phone all day then they will be more likely to prefer a voice channel. If your users primarily interact with you through your website, app, or email they will be more likely to prefer a web chat option. Phones are ubiquitous and nearly everyone knows how to make phone calls - whether or not they like using the phone is a different story! Similarly, instructing users to "go to our website" may be a welcome relief or a daunting challenge depending on your user

base. Of course - there is nothing to stop you from supporting both channels - just time and money!

With all else being equal it is easier to start with web chat rather than voice. Voice introduces two additional AI service (speech to text and text to speech conversion) which requires additional development and training work - though training a speech engine is continuing to get easier[3]. I will dive deeper into voice training and voice solutions in Chapter 9.

Table 4 Comparison between web and voice channels

Web Benefits	Voice Benefits
• Technical implementation has less moving parts • Do not have to train speech recognition • Easily deployed on websites and mobile apps	• Almost everyone has a phone • Friendlier to non-tech-savvy users • Easy to transfer users in and out of virtual assistant using public telephony switching

Let's explore more deeply the differences between voice and web channels. The key differences are aligned by how users receive information from the assistant, and how the assistant receives information from the user.

3.2.1 How users receive information in voice and web

The first difference between voice and web is the way the user receives information from the solution. In a web solution, the user will have the full chat transcript on their screen - a complete record of everything they said and what you said. While many users don't like scrolling, they have the option to scroll back to view the entire conversation at any time and re-read any part they would like. They can print their screen or copy/paste parts or the entire transcript at any time. A lengthy response can be skimmed or read in full at the user's discretion. The user may multi-task while chatting with little impact on the broader solution. A web assistant can return rich responses including not just text but images, buttons, and more. Figure 11 shows some of the differences.

[3] I have published a quick guide to bootstrapping speech to text training from Watson Assistant training data with Marco Noel: https://medium.com/ibm-watson/quickly-improve-your-voice-agent-with-a-speech-model-15f20749cfb

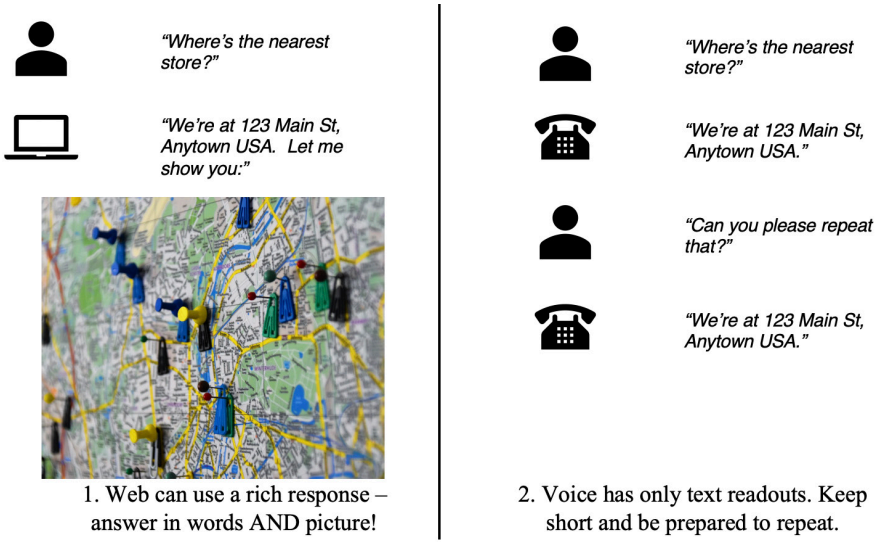


Figure 11 The web channel allows rich responses like images. Voice solutions should be prepared to repeat important information. Map photo by Waldemar Brandt on Unsplash.

Conversely, in a voice channel, the user's interaction is only what they hear. If the user misses a key piece of information, they do not have a way of getting it again, unless you code a "repeat" question or functionality into your assistant. Figure 11 shows how web and voice channels can best handle a question like "where's the nearest store?"

Further, a long verbal readout can be very frustrating for a user: they may need a pencil and paper to take notes, they may have to wait a long time to get what they want, and they probably have to be very quiet for fear of confusing the speech engine. (I practically hold my breath when talking with some automated voice systems!). Also, directly sending rich media responses like images is impossible over voice though you may be able to leverage side channels like SMS or email to send information for later review.

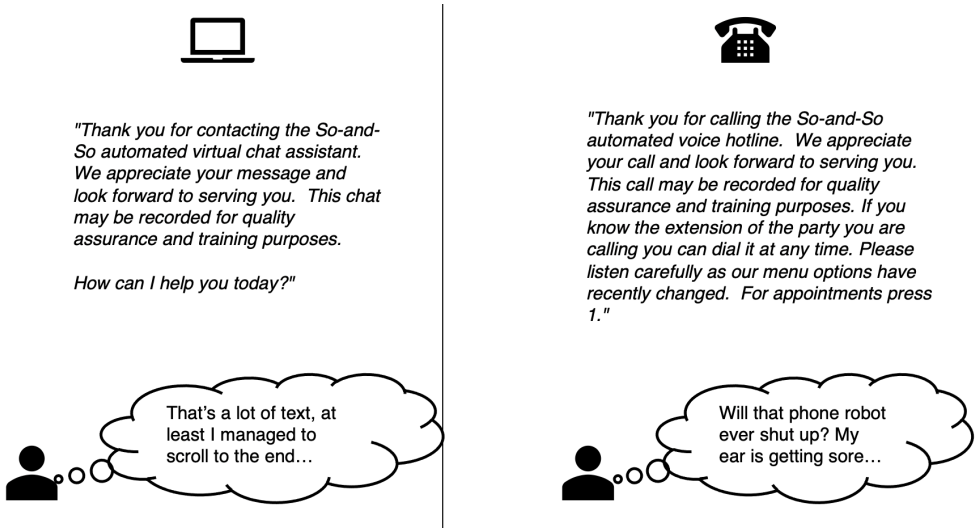


Figure 12 Different channels have different user experiences

You must be aware of the cost to the user when you have a long message. As shown in Figure 12, a web user can skim long messages, but a voice user cannot. Beware the temptation to cram every last piece of information into a message, especially in voice. The average adult reading speed is around 200 words per minute and speaking speed is around 160 words per minute, though automated speech systems can be tuned to speak more quickly.

Consider a hypothetical greeting:

> ***"Thank you for calling the So-and-So automated voice hotline. We appreciate your call and look forward to serving you. This call may be recorded for quality assurance and training purposes. If you know the extension of the party you are calling you can dial it at any time. Please listen carefully as our menu options have recently changed. For appointments press 1."***

I timed myself reading this 62-word message. It takes 20 seconds of audio to get to the first useful piece of information! (Hopefully you wanted appointments!) Perhaps the lawyers insisted - but look at how much "junk" is in that message from the user's point of view. Figure 13 breaks down the greeting.

Text	Feeling	Internal monologue	Time
"Thank you for calling the So-and-So automated voice hotline"		“I appreciate the thanks. I know this is an automated hotline just from listening to the voice.”	0-3 seconds
“We appreciate your call and look forward to serving you.”		“Please don't waste my time with this corporate-speak.”	3-6 seconds
“This call may be recorded for quality assurance and training purposes”		“I'm sure you are recording this and doing everything you can with this recording. Please let's get to what I need.”	6-10 seconds
“If you know the extension of the party you are calling you can dial it at any time.”		“If I knew their extension I wouldn't be here! I never know the extension when I call these things!”	10-14 seconds
“Please listen carefully as our menu options have recently changed.”		“I've been listening carefully for an eternity and you have yet to tell me anything useful!”	14-18 seconds
“For appointments press 1.”		“Finally some useful information! But I don’t want appointments!”	18-20 seconds

Figure 13 User's thought progression through a long greeting

Contrast that with the following greeting:

> ***"Thanks for calling So-and-So. Calls are recorded. How can I help you?"***

This new message has 4 seconds to value while still covering the basics of greeting, notification, and intent gathering. You only get one chance to make a great first impression - don't waste your users' time on your greeting!

Take to heart the following quote:

> **"I have only made this letter longer because I have not had the time to make it shorter."**
>
> **Blaise Pascal, The Provincial Letters (Letter 16, 1657)**

It takes work to be concise, but your users will appreciate you for it!

3.2.2 How the assistant receives information in Voice and Web

Another key difference between voice and web is how you receive input from the user. In a web channel, you can be sure of receiving exactly what was on the user's screen. You may provide a pop-up form to collect one or more pieces of information at once (first and last name, full address). The user may have clicked a button and you will know exactly what they clicked. The user may have misspelled one or more words but virtual assistants are increasingly resilient to misspellings and typos as demonstrated in Figure 14.

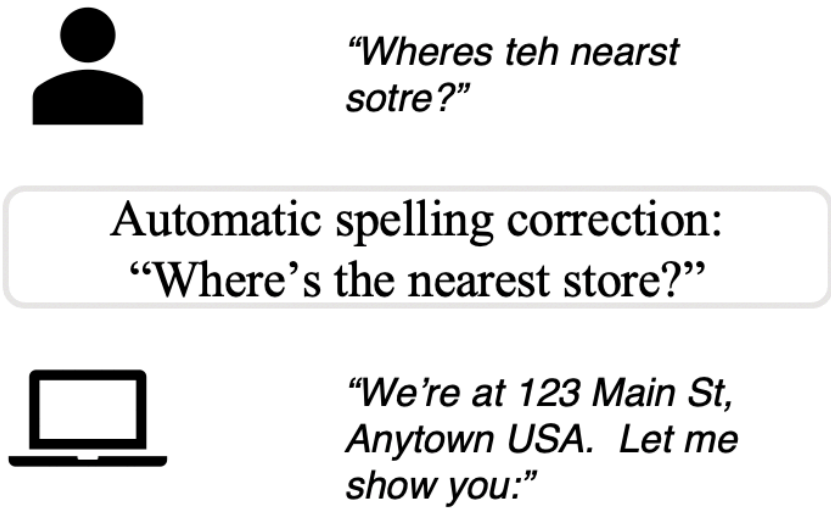


Figure 14 Most major virtual assistant platforms are resilient against misspellings

In a voice channel you will receive a textual transcription of *what the speech engine interpreted*. Anyone who has used voice dictation has seen words get missed. The assistant can be adaptive to some mis-transcriptions (just like it can be adaptive to misspellings in chat) when the words are not contextually important. Figure 15 shows a voice assistant adapting to a pair of mis-transcriptions: "wear" for "where" and "a" for "the".

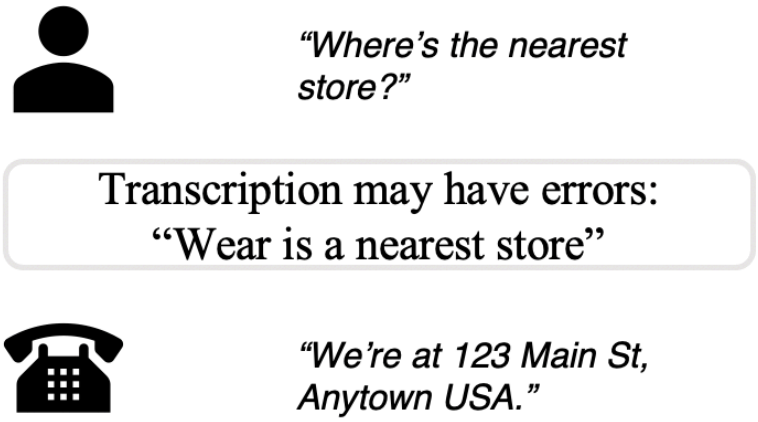


Figure 15 Voice assistants can adapt to mis-transcriptions in general utterances as long as the key contextual phrases are preserved, like "nearest store"

Aside from simple mis-transcriptions, another class of inputs gives speech engines trouble - any input that is hard for humans will be hard for speech engines as well.

Proper names and addresses are both notoriously difficult for speech engines in both recognition (speech to text) and synthesis (text to speech). When I'm talking to a person on the phone and they ask for my last name I say "Freed. F-R-E-E-D" since many people hear "Free" or try to use the old German spelling "Fried". Common names are not that common - you should be easily able to rattle off a couple of "uncommon" names within your personal network rather quickly. Speech engines work best with a constrained vocabulary, even if that vocabulary is "the English language", and most names are considered out-of-vocabulary.

What's a vocabulary?

Speech to text providers refer to a "vocabulary" – this is simply a list of words.

A speech model is trained to recognize a set of words. A generalized English model may have a dictionary that includes all of the most common words in the English language.

Your virtual assistant will probably need to deal with uncommon words and jargon. The name of your company, or the products your company offers, may not be included in that vocabulary of words. If so, you will need to train a speech model to recognize them.

Addresses are harder than names. I found a random street name on a map "Westmoreland Drive" - if you heard that would you transcribe "Westmoreland Drive" or "W. Moreland Drive" or "West Moorland Drive"? Figure 16 shows a challenge in mapping similar phonetics to words that sound similar.

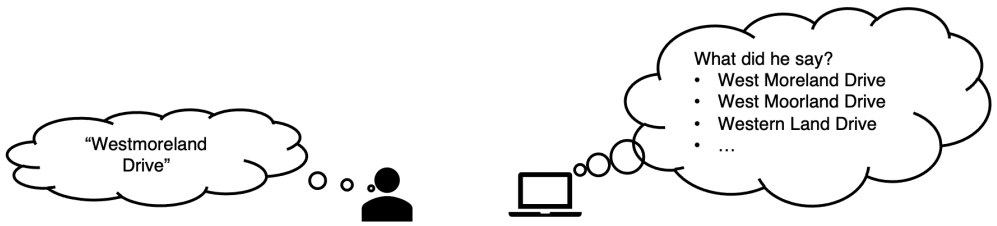


Figure 16 Transcription challenges on unusual terms

On spelling out words and the difficulty of names and addresses

Spelling out a difficult name can sometimes be helpful for humans, but it does not help machines much. Letters are easily confused with each other - B/C/D/E/G/P/T all sound similar without context. Humans may require several repetitions to correctly interpret a proper name, even spelled out.

There is rich literature on the difficulty of names and addresses. One such article is "The Difficulties with Names: Overcoming Barriers to Personal Voice Services" by Dr. Murray Spiegel (2003)
http://web.media.mit.edu/~geek/TheDifficultiesWithNames.htm

The difficulty in receiving certain inputs from users affects the way you build a dialog structure, perhaps most significantly in authenticating users. In a web chat you can collect any information you need verbatim from a user. You may in fact authenticate in your regular website and pass an authenticated session to the web chat. In a voice channel, you need to be more restrictive in what you receive. During authentication, a single transcription error in the utterance will fail validation, just like if the user mistypes their password, as shown in Figure 17.

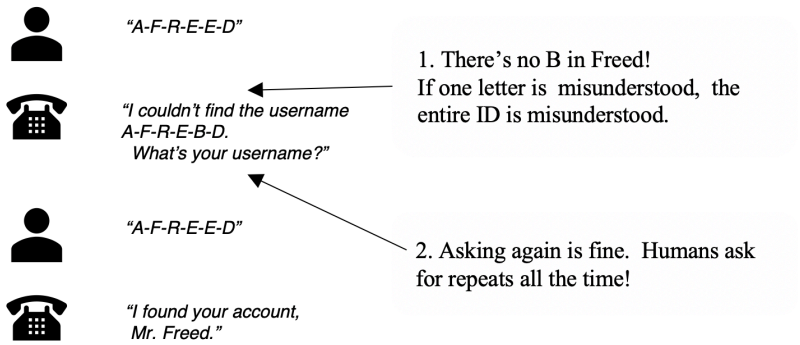


Figure 17 For most data inputs, a single transcription mistake prevents the conversation from proceeding. Voice systems need to take this into account by using re-prompts or alternate ways to receive difficult inputs.

In the best cases, speech technology has a 5% error rate and in challenging cases like names and addresses the error rate can be much, much higher. (Some voice projects report a 60-70% error rate on addresses.) With alphanumeric identifiers, the entire sequence needs to be transcribed correctly, as shown in Figure 17. A 5% error rate may apply to each character, so the error rate for the entire sequence will be higher. For this reason, a six-digit ID is much more likely to transcribe accurately than a twelve-digit ID.

For accurate transcriptions, constrained inputs like numeric sequences and dates work best. If you encounter a transcription error, you can always prompt the user to provide the information again. Keep in mind that you will want to limit the number of re-prompts (more coming up in Chapter 4, section 4 - What if the bot doesn't understand). You may implement a "three strikes rule" – if three consecutive transcriptions fail then you direct the user to alternate forms of help that will serve them better.

Table 5 Summary of data types by how well speech engines can transcribe them

Data types that transcribe well	Data types that do not transcribe well
• Numeric identifiers (ex: Social Security Number) • Dates • Short alphanumeric sequences (i.e. "ABC123")	• Proper names • Addresses

Voice authentication can make use of an alternate channel such as SMS. You can send a text message to a number on file for a user with a one-time code and use that in authentication, instead of collecting information over voice. If you absolutely must authenticate via an option that is difficult for speech, over the speech channel, be prepared to work hard in both speech training and orchestration layer post-processing logic. You will need a good call hand-off strategy in this scenario.

3.3 Summary

- Users will have a variety of request types with a range of frequency and complexity. You need to carefully prioritize a subset of these request types to implement in the first version of your assistant.
- It takes a village to build a successful virtual assistant. There are many different perspectives you need to consider such as business, technical, and legal. A diverse team should contribute to the solution.
- Review each request type and determine all the steps required to fully automate it in a process flow. Identify steps that can be removed or replaced so you can deploy a minimum viable product.
- There are different requirements for effective dialog over web and voice channels. Web allows rich responses and users can scroll whereas voice responses must usually be shorter and may need to be repeated. In the web channel, you are always sure of what the user typed but in the voice channel you receive what the speech engine transcribed (not necessarily what the user said!) Some parts of a process flow will need to be altered based on the specific channel.

4

Designing effective dialog

In this chapter, you will learn:

- How to write dialog supporting your process flows
- How to write dialog with tone, empathy, and variety
- How to keep dialog on track without being rude, to increase your chances of a successful process flow
- How to ask questions that increase your chances of getting a useful response
- How to respond to conversational mishaps, which may result in a communications transfer to a human agent

In the previous chapter, you learned how to design effective processes that your virtual assistant will implement. After you design all the flow charts and user-to-system interactions, you need to breathe life into them by writing the actual dialog text your assistant will read or speak.

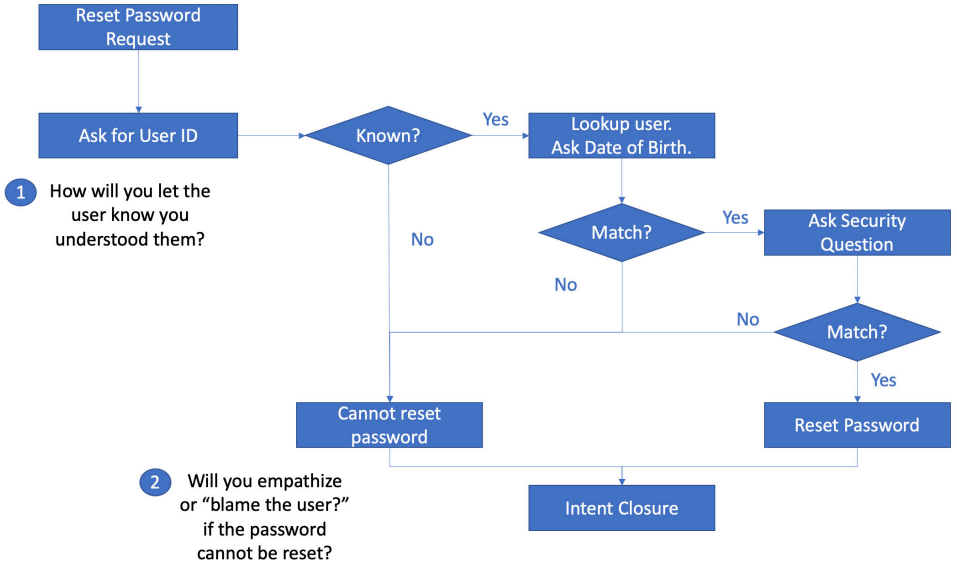


Figure 4.1 Motivating example for the chapter. This is a good process flow, but what will the assistant actually say at each of these prompts? How can we prime users to answer questions in a way that moves us through this process? What tone will the assistant use at each prompt?

In Figure 4.1 the assistant can respond several different ways to a user who needs to reset a password. A response of "I need your user ID so I can reset your password" is direct but has nearly accusatory tone. A response of "I'm sorry you can't access our system. I'll try to help you. To start, what's your user ID?" conveys a softer tone and lets the user know what's happening next. From a programming standpoint both of these responses are identical, but users will not perceive them the same way!

There are multiple types of nodes in a given process flow. Sometimes the assistant will be passive, answering questions directed by the user. Sometimes the assistant will take a more dominant approach, directing a user through a process flow. Each of these response types requires dialog that conveys empathy and a consistent tone while getting the job done.

The assistant can ask for any required information from the user in several different ways. For instance, one could get the user's zip code by asking for an "address", a "zip code", or a "five digit zip code". You can surely imagine which one will generate the most consistently usable responses!

In human-to-human conversations we know it's not just what you say, but how you say it[1]. Words matter but so does tone. A difficult message can be softened with empathy. Just like you shouldn't blurt out the first thing that comes to mind, you shouldn't be thoughtless in what goes into your virtual assistant's dialog.

[1]Some people even formulize it. Albert Mehrabian's 7-38-55 Rule of Personal Communication is that spoken words are 7%, voice and tone are 38%, and body language 55% of verbal communication. Read more at https://www.rightattitudes.com/2008/10/04/7-38-55-rule-personal-communication/

In this chapter you will learn how to write dialog with tone, empathy, and variety such that a user is not constantly reminded they are talking to an unfeeling bot. You will learn how to keep dialog on track (or bring it back on track) without being rude, and increasing your chances of a successful process flow. You will learn how to respond when the assistant doesn't understand the user or when you need to transfer the conversation away from the assistant and into a human-to-human interaction.

By the end of this chapter you'll be able to write dialog supporting your process flows and you'll be ready to train your assistant to recognize the intents that will initiate these process flows.

4.1 How to write dialog

Writing the dialog copy is a first-class skill in building a virtual assistant. If you use a professional copy writer for your website and marketing materials, you should apply the same level of skill and polish to the textual dialog within your virtual assistant. Like I introduced earlier, your virtual assistant is out on the front lines interacting directly with your customers and represents your brand. The text your assistant returns on the screen or speaks over the phone is as important as the technical functionality you enable in your assistant. Users will remember a cold, impersonal assistant negatively and will dislike it even more if they feel the assistant is wasting their time. Conversely a pleasant bot delivering the right amount of information in a timely manner will be remembered fondly.

It is important to get the structure of the dialog tree right. The previous section outlines a number of considerations to build a robust process flow for dialog to sit in. After you and your team have done all this precision work it is important to breathe life into the process with well-designed dialog copy. The process tree should be designed by your full team such that it includes at least a description of each step (this node is where we ask for the date of birth) and the specific wording can be refined by a smaller group of writers.

A "war room" approach

You may find it advantageous to "lock everyone in a room" and get immediate consensus on the dialog. This certainly reduces the number of back-and-forth sessions and review cycles for edits. Some teams have implemented self-help virtual assistants after a single all-day writing session to design their process flow.

Regardless of the approach you take to writing the dialog copy you need to take it seriously. There are a couple of specific writing considerations you need to take. These are described in the remainder of this section.

4.1.1 Conversational tone

Earlier in this book we talked about how there are an infinite number of ways for users to express the same intent with different word variations. Recall all of the different ways people might express the need to reset a password? The infinite variations of language apply to the text you send back to your users as well. At any phase of a conversation there

are a near-infinite ways for you to express what you want to say but there are definitely some good practices to follow that will help you provide a good experience.

The first thing to consider is what is the tone your virtual assistant will generally use? Whether you intentionally inject a tone or not, users will find or imply a tone from your virtual assistant. Many people hate talking with bots because they are impersonal, and the bot responses feel disembodied. Figure 6.2 demonstrates two extreme tonal variations

Your assistant does not need to pass the Turing Test[2], and it does not need to go all the way to the Uncanny Valley[3], but you should be mindful of the way the bot's language will be perceived by users. It's good to be proactive and define a target tone.

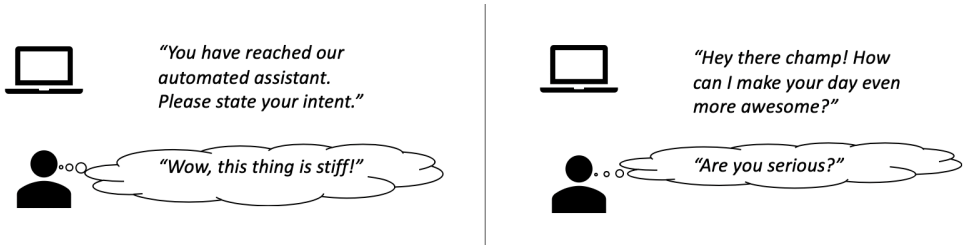


Figure 4.2 The tone used by your assistant influences how users feel about it

Does your organization already have a brand? If so, your users will expect your assistant's tone to have some affinity to that brand. Does your brand have a serious nature or are you perceived as informal? If your assistant is 180 degrees different than your existing branding your users will find this jarring. If you have a marketing team they need to be "in the room" to make sure the tone of your assistant is in line with the rest of your branding. As briefly discussed in Chapter 1 the tone should line up with the persona your brand uses, if it uses a persona.

With a target tone in mind it is still important to use an appropriate tone for the situation. If a user is bringing a problem like "I can't login!!!!" you do not want to react in a callous manner. Your assistant should respond with empathy. Acknowledging a user problem with a simple "I'm sorry" is a useful and empathetic step. If empathy does not come naturally to you, you should have the assistant dialog written or revised by someone for whom it does!

[2]Turing Test: A test devised by Alan Turing in 1950. A computer application that behaves in a manner indistinguishable from a human is said to pass the Turing Test. https://en.wikipedia.org/wiki/Turing_test

[3]The Uncanny Valley is a proposed relationship between an object and human responses to it. There is an inflection point the more "human-like" an object is made the more disconcerting it is. This is perhaps most well-known from animated films that try to make photo-realistic recreations of humans. Read more at https://en.wikipedia.org/wiki/Uncanny_valley

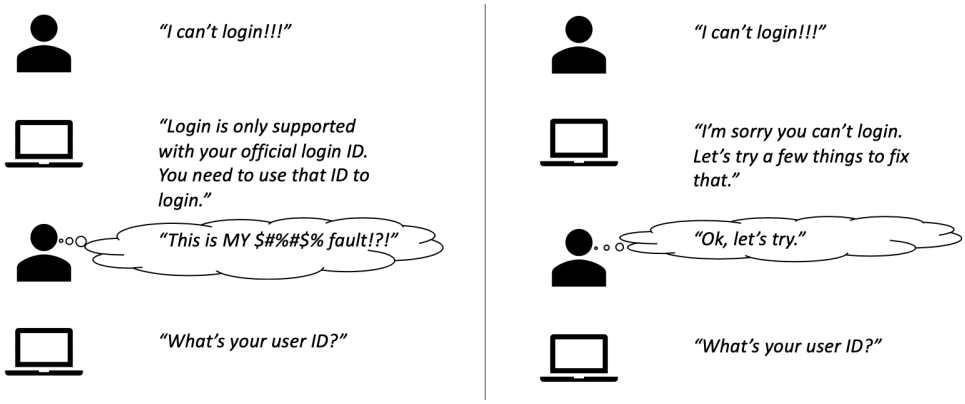


Figure 4.3 A little empathy goes a long way. Both dialogs are using the same process flow but the cold, technical approach on the left implies "blame the user" language.

4.1.2 Don't repeat yourself (too much)

Users will sense robotic-ness if every interaction with your assistant has a templated, formulaic feel. Variety is the spice of life! Variety is particularly useful in the areas of the dialog that users will encounter the most:

- **The greeting message**: The very first message your users encounter when they interact with the assistant. Some components of the message may need to stay the same (for instance, a disclaimer) but the others can be varied.
- **List of capabilities**: Assistants often tell users what capabilities are available, either directly (in a greeting) or only when asked. Consider varying the order the capabilities are listed, at least for separate interaction sessions.
- **I don't understand**: It's frustrating for users to hear "I don't understand" over and over. Humans naturally vary the way they say they don't understand, and it removes a little robotic-ness when your assistant varies responses this way too.

Your virtual assistant platform will let you vary dialog responses for any given part of your dialog tree. Figure 6.4 shows one such example from a Watson Assistant implementation.

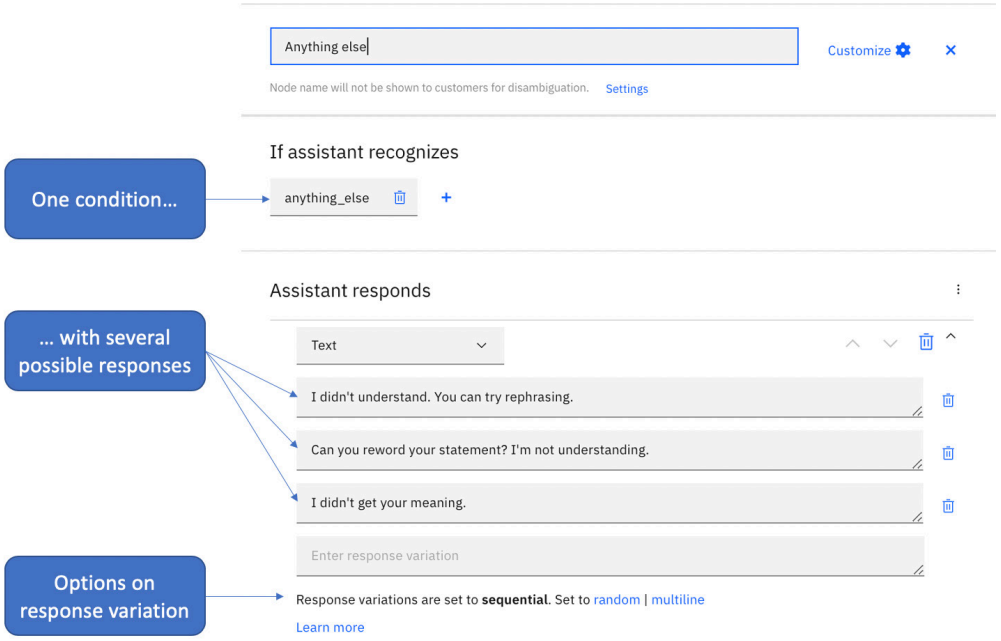


Figure 4.4 Most virtual assistant platforms let you set multiple possible responses for a given dialog node. The platform will randomly select one of these responses each time the dialog node is visited.

Check the documentation on your virtual assistant platform to see how you can vary the responses given by a dialog node. This is a common feature, and some platforms even offer multiple message selection policies in addition to random selection. Message variation is a simple way to improve the way your users perceive the assistant – go use this feature!

4.1.3 Acknowledge the user

The way to be a better communicator is to listen carefully to the other side, empathize with the other side and reflect back what you are hearing, and only then move on in the conversation. This generalized advice holds true for interacting with virtual assistants as well as humans. Consider these two hypothetical dialog flows in Figure 4.5:

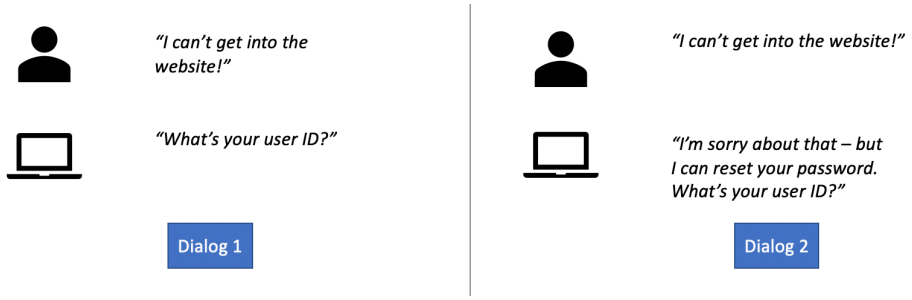


Figure 4.5 Acknowledge that you understood the user. (And use empathy while you're at it!)

Figure 4.5 shows two different dialogs implementing the same process flow. Dialog 1 is concise, but the user has no indication that they were understood. Dialog 2 we have acknowledges the user with a simple reflection of the understood intent (reset password) and shows empathy to the user as well. When Dialog 2 requests the user ID the user has good situational context for why the assistant asks this question. Dialog 2 is more likely to lead to a good outcome than Dialog 1.

Additionally, it is a good practice to give something to a user before asking them for additional information. At the start of the conversation we put the onus on the user with something like "How can I help you?" They have provided some information in their utterance in the form of an intent. We can give something - a message that we understand their initial request - before requesting additional information. Users will generally understand that more information from them will be required as the conversation goes on, but it is still good to give something to the user when you can.

Reflecting back your understanding is particularly important in the cases of open-ended input, where you are attempting to extract the user's intent. *Intent detection* is by its nature a probabilistic exercise and there is a chance you will misunderstand the user's intent. You can acknowledge the intent and move on (as in this previous example) or you can ask the user to confirm your understanding - for example "You'd like to reset your password, correct?")

A *confirmation prompt* will improve your overall solution accuracy and if the confirmation is negative you can easily handle it in your dialog flow, however it will necessarily lengthen the conversation. A confirmation prompt is a good idea when your classifier is not performing well, and you have not been able to improve it. Allowing the user to know what you think their intent is gives them useful information they can act on. In the password example they could interrupt your flow with "that's not what I want!" or worst-case restart the conversation - at least they have the choice.

Acknowledging the user is also good even when the expected input is constrained. For instance, when you ask for a user ID you generally have a pattern of what a user ID looks like – let's assume an alphanumeric sequence with minimum/maximum length. When the user provides the input that you expect, you should give a clue that you've received it, before moving on with the next step of the conversation. Consider these dialog examples for our password reset flow:

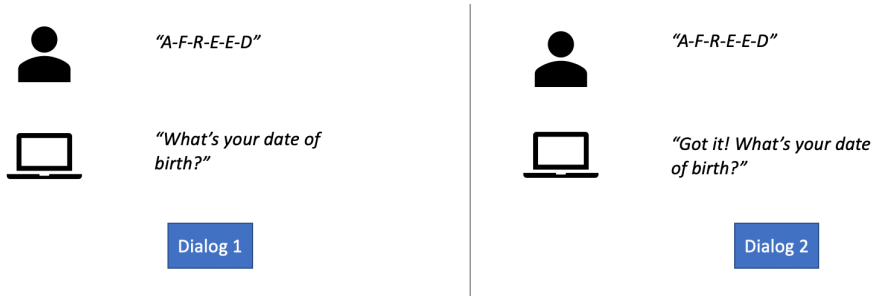


Figure 4.6 A simple acknowledgement lets the user know the conversation is proceeding well

Figure 6 contrasts two approaches. In Dialog 1 the user might assume the assistant collected the user ID, but they can't be sure. Is the system going to validate all of the input at the end or has the user ID been validated already? It's hard to say and the user can't be sure where they stand. In Dialog 2 it is clear that the user ID has been received and probably validated as well. With clear acceptance of the user ID it makes sense to advance the dialog to the next input question on date of birth. Dialog 2 can be further shortened to "And what's your date of birth?" with the "And" providing a short acceptance message without causing a pause between the statement and the question.

There's a lot more to conversational design!

Conversational design is a rich field - entire books have been written on the process including "Conversational UX Design" for general conversational design and "Practical Speech User Interface Design" for voice solutions.

4.1.4 Handling chit-chat

Humans interact differently with each other than they do with machines. Human-computer conversation generally follows "turn-taking" with each party saying one thing and waiting for the other to respond. Human-human conversation is a bit more free-form, often including digressions and meandering and side conversations. Humans are remarkably adaptive in this way!

It's common for a human-human conversation to start with "How are you doing today?", "How's the weather?" or a friendly greeting "I hope you're healthy and happy." Some people even like to start conversations that way with a virtual assistant. They may be trying to find out if they are talking to a bot or might just want to mess with it. Any conversational messages not directly related to serving a goal are generally referred to as "chit-chat".

Your virtual assistant should be prepared to deal with a little bit of chit-chat. Ironically, the more chit-chat your assistant engages with the more chit-chat it will get. There can be a kind of arms race - if you start responding to "Tell me a joke" you'll soon get requests for "Tell me another one." This is generally not a good use of your time in both training the assistant as well as constructing the dialog it will use. Rather than meeting it head on it is a

good idea to turn the chit-chat back to the specific goals and training embedded in your assistant.

For instance, if your virtual assistant greets with "How can I help you?" and a user responds, "Hi there", you can respond with a greeting of your own and a small list of capabilities to start a goal-oriented conversation. In our sample assistant a good response to any of these could be "Hello and thanks for your interest. I'm able to help with several things including password resets, store locations and hours, and making appointments. How can I help you?"

By default, most virtual assistants will not recognize chit-chat unless they are trained to. In these cases, the assistant will respond with the "default" or "catch-all" condition. This is not the best user experience - if the user says "Hello" and the assistant says "I don't understand" that is very machine-like! - but it can be used in initial assistant construction. This may be all you need if you have a very task-oriented user group. You can also address this situation by putting additional information, like a listing of capabilities, in the "I don't understand" node to help put the conversation "back on track".

4.2 How to ask questions

The concept of nearly infinite variations applies to question-asking as well. Your virtual assistant is driving or steering the conversational flow and the questions and statements it makes can help keep the conversation "on track." While you cannot exactly predict what direction a user will go you can increase the probability of an "expected" response through gentle nudges in the dialog. There are two primary types of questions:

- **Open-ended questions**: In a question like "How can I help you?" the user could respond with nearly anything. In an open-ended question you are generally trying to determine the user's intent.
- **Constrained questions**: Questions like "what's your date of birth?" have an expected outcome - the response will generally include a date.

As you design your conversational flow you should be aware of what type of question you are asking and what kinds of responses you are scripting for. You will get better responses during conversations with users if you know the question type for each section in your dialog flow. A poorly worded constrained question can generate open-ended responses and could confuse your assistant.

Users say the darndest things!

No matter how well you script out your processes, users may still surprise you. For example, when you ask, "what's your date of birth?" a user may "opt-out" by responding "I want to speak to a human".

Virtual assistant providers will give you a way to provide a default response in these cases. Be prepared to use it.

Open-ended questions can be constrained if you provide a list of suggestions. The first question in most assistants is generally some variation on "How can I help you?", a perfectly

open-ended question. You can help users by including a brief list of the assistant's capabilities, like "I can help you with several things including finding a store near you and setting up an appointment. How can I help you?" Alternately, you can choose to list the assistant's capabilities when the assistant doesn't understand the user's response to this first question. By providing the user a hint, you have effectively narrowed the range of possible responses.

One question that sits on the border of open-ended and constrained is a common question situated at the end of a dialog flow - after the assistant has completed servicing one intent and is ready to start another. The question often looks like "Can I help with anything else today?" As currently worded, this is ambiguous - it could be a constrained question (user says "yes" or "no"), or user directly starts a new intent (user says "I also want to make an appointment"). You can lightly nudge this question to a constrained question by asking "Yes or no, can I help with anything else today?" or provide a stronger nudge by displaying "Yes" and "No" button options. Alternately you can code the dialog node to expect a "yes", "no", or intent-based response. A nudge cannot guarantee a user will do what you expect but it primes their brain towards the direction you desire.

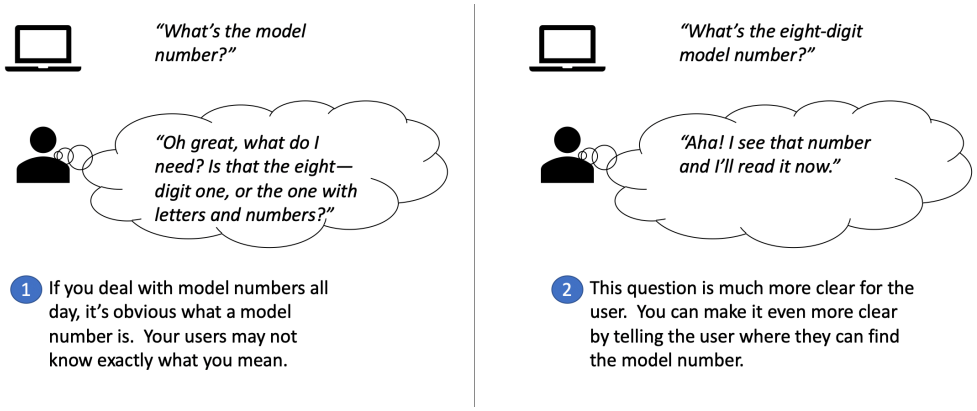


Figure 4.6 If you ask an unclear or ambiguous question, the users will struggle. Give the user a hint whenever possible.

Constrained questions have an implied response type, for instance a "date of birth" question should have a date response. However, there are many times it is appropriate to be more specific about what you want. You may integrate with an API that expects information in a certain format or you may be asking for a piece of data users are unfamiliar with. Asking a question precisely with consideration of your constraints will yield more responses that move the conversation forward.

Table 1 Improving unclear questions by adding precision

Imprecise question	Precise question
What's your date of birth?	What's your month, day, and year of birth?
What's your zip code?	What's your five-digit zip code?
What's your employee ID?	What's your nine-digit employee ID number?

The last example in Table 1 is particularly useful for information users don't reference often. When I call a support line and they ask for a serial number on some product, I am often squinting at the bottom of the product and trying to figure out which of the alphanumeric sequences I'm being asked for. Knowing the format and length of the input helps me figure out what information is needed and saves the technician the problem of saying "no not that ID". Depending on your use case you may wish to supply additional context ("what's the nine-digit number on the top-left of your ID card?"). In order to save time on dialog readout you may wish to save the lengthier instructions for a failure scenario. You need to balance the time and cost of providing the additional context versus the time and cost of getting "invalid" input.

Constrained questions should also have a mechanism for recognizing statements like "I don't know", or "I don't have it", in addition to recognizing expected responses.

4.3 What if the assistant doesn't understand?

Despite all of your planning, design, development, and test efforts there will still be times that the assistant does not understand the user. These times are wonderful opportunities to learn about unexpected user behavior and can help inform future improvements. You also need to have a plan for these situations – it's frustrating for a user when they are dealing with an assistant that does not understand them (as shown in Figure 8).

Figure 4.7 A virtual assistant may have no idea what the user means

Do not fall into the temptation that an artificial intelligence can or even must always understand the user. Misunderstandings happen in human-human conversations as well and you can take a cue from how we as humans resolve misunderstandings.

There are several ways to respond in cases where the assistant does not understand the user:

- **Reprompting**: Admitting that you did not understand and asking the user to clarify.
- **Disambiguation**: Giving the user a list of possible interpretations of what they said and asking them to pick one.
- **Escalation**: The assistant "admits defeat" and sends the user to an alternate channel.

4.3.1 Reprompting

The simplest response when the assistant does not understand is to be completely transparent, responding with some flavor of "I don't understand." Most virtual assistant platforms will provide the capability for a default response, one when no other dialog response satisfies the conditional logic within the dialog flow. If you tried the demo in Chapter 2 you probably saw and tested this response node. Reprompting simply means to ask the same question again, usually with different wording.

There is an unfortunate consequence of new virtual assistants that the first direct response the user receives from the assistant may be an "I don't understand" message. This is a necessary side effect of giving users a free range on what they can say - both a blessing and a curse compared to strict menu options or numeric call trees. Thus it is good to handle this response with care.

A good "I don't understand" message should convey empathy or at least not blame the user. Take a stance that it's the assistant's fault for not understanding! "I don't understand" places responsibility on the assistant - "You provided an invalid input" places responsibility on the user. This advice also applies to constrained questions. When asking the user for a particular input (say, a numeric identifier) and the response is the wrong format (say, too many digits), a response like "You provided too many digits" sounds accusatory compared to "I couldn't find that ID, could you provide the nine-digit ID again?" This is particularly important in voice solutions where it's likely that the voice system missed or misheard a digit rather than the user provided the wrong number of digits.

In addition to accepting responsibility for the misunderstanding, it is good to nudge the conversation back on track by providing additional guidance. The phrase, "I didn't understand. You can try rephrasing." accomplishes both the acceptance of responsibility ("I didn't understand") and gives a suggestion to move the conversation forward ("You can try rephrasing.") Without a nudge the user may decide to repeat themselves and in a web chat solution that will not get them any farther (it may help in a voice solution, if the misunderstanding was due to a speech transcription failure). Positive suggestions for next steps include a request to reword/rephrase or to give additional clarifying instructions. For an open-ended question you may suggest a couple of common questions you support. For constrained questions you might provide additional guidance on how to provide the information you are seeking ("the employee ID is on the back of your corporate badge").

An "I don't understand" node should be triggered when the user response is not understood with high confidence. For instance, Watson Assistant only selects an intent if that intent is predicted with confidence greater than 0.2. (Other platforms use different threshold levels.) Depending on your use case and training you may wish to use a different confidence level. In Chapter 2's demo I used a confidence threshold of 0.5. A low confidence intent has a probability of being wrong and it is better for your users to respond with "I don't understand" rather than to respond based on the wrong intent! (In Chapter 7 we will explore the meaning of confidence further)

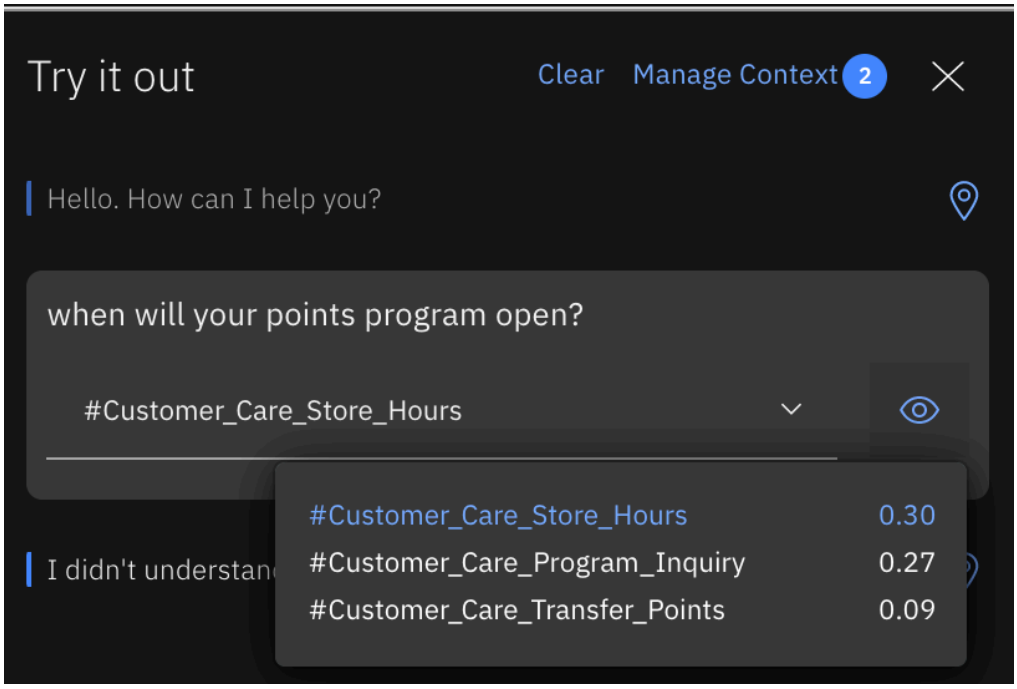


Figure 4.8 A virtual assistant may weakly understand an utterance, detecting intents with low confidence. The highest-confidence intent #Customer_Care_Store_Hours is still a low confidence intent. When the intent confidence is low you are often better off responding that you didn't understand the user.

In Figure 4.9 the utterance "when will your points program open?" triggered a low confidence intent. The `#Customer_Care_Store_Hours` (with 0.30 confidence) just barely beat the more correct `#Customer_Care_Program_Inquiry` intent, (with 0.27). If the dialog was not coded to consider intent confidence the response would have been to provide the store hours - a completely unsatisfying response for the user.

User utterances that trigger a reprompt should be periodically reviewed to see if the assistant's dialog rules or intent training need to be updated to capture more user utterances. We will discuss analyzing user utterances further in Chapter 6.

Table 2 Best practices for re-prompting

Do	Do Not
• Convey empathy ("I'm sorry, I did not understand") • Give specific advice how to move the conversation forward ("Please give me your nine-digit member ID")	• Blame the user ("You provided an invalid input") • State a problem with no solution ("You provided the wrong number of digits") • Assume it is bad to re-prompt when you

• Re-prompt if your assistant has low confidence in the detected intent	are uncertain

4.3.2 Disambiguation

Sometimes the virtual assistant will not be able to choose a single intent based on the user's utterance. The utterance may trigger multiple intents with medium to high confidence and in this case it can be appropriate to ask the user what they meant. Well-defined intents are trained with general boundaries but an utterance can still sit near multiple boundaries.

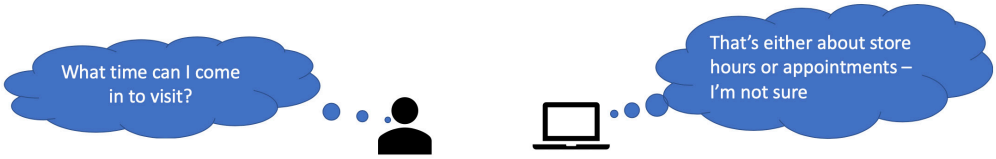


Figure 4.9 When multiple intents could possibly be correct, disambiguation is appropriate

As shown in Figure 4.10, consider the case of "What time can I come in to visit?" This is an ambiguous utterance for the FICTITIOUS INC assistant – this utterance sounds a little like a request for an appointment and a little like a request for store hours. It's ambiguous because it has a time component ("what time") but the ending language ("visit") does not neatly fit into appointments or store hours. *Disambiguation* is best invoked when two or more intents have similar confidence values.

In this case it's appropriate to offer the user a list of suggestions reflecting your partial understanding. The assistant's confusion is not total, just partial, and it's reasonable to reflect this. This gives the user a clear path forward selecting from a short list of choices - one of the choices generally being "none of these", which would invoke the standard misunderstanding flow.

If you followed the Chapter 2 demo, try asking your assistant "What time can I come in to visit you?". On my assistant the intents with the highest confidence were `#Customer_Care_Store_Hours and #Customer_Care_Appointments`, with 0.58 and 0.57 confidence respectively. This is the perfect time to use disambiguation. Both intent confidences are above our low confidence threshold, and they are very similar to each other.The assistant can respond with a clarifying question: "Did you mean 'Store Hours', 'Appointments', or none of the above?"

The major virtual assistant platforms give you enough information to both determine when disambiguation is needed and to help you implement the disambiguation. Disambiguation can be implemented in application logic, described in Table 3. Some virtual assistant platforms offer a disambiguation feature that automates this logic for you.

Table 3 General rules for using disambiguation, in case of intents with similar confidence

Trigger disambiguation when…	Multiple intents have a similar confidence values, and that value is above your "low confidence" threshold.
Disambiguation choices should include…	An option for each intent that had similar confidence, and a "none of the above" option.

Disambiguation is also a good idea when the user makes a one-word utterance. In the case of the FICTITIOUS INC assistant, consider if the user utterance is "account". This could reasonably mean `#Customer_Care_Open_Account` or `#Customer_Care_Cancel_Account` or `#Customer_Care_Loyalty_Status` or even possibly `#Customer_Care_Products_Offered`. You may have a temptation to update training such that this utterance is forced to go to one of the intents, but this is a reflection of what you want and not what the user wants. One-word utterances are very frequently ambiguous, and in this context "account" is certainly ambiguous. A disambiguation flow is useful for most one-word utterances that have affinity to multiple intents.

User utterances that trigger a disambiguation flow should also be periodically reviewed. These utterances can show unclear boundaries or unexpected relationships between different intents. These utterances may form useful new training data. You should periodically review disambiguation results in your assistant's logs to see if you need to improve the training behind any ambiguous intents.

Table 4 Best practices for disambiguation

Do	Do Not
• Use disambiguation when multiple intents are returned by the classifier with high confidence • Use disambiguation when a user utterance is a single word that relates to several different intents • Review your assistant's logs to see when disambiguation was needed	• Ignore intents with high, but not the highest, confidence for an utterance • Pick a single intent for a single-word utterance when multiple intents are reasonable.

4.3.3 Escalation

There are some times when a virtual assistant should stop trying to handle the conversation and hand it off to another channel. Just like a human-to-human conversation, sometimes the conversational participant should be escalated to a manager.

There are three primary reasons for escalation:

1. opt-out
2. misunderstandings
3. business process

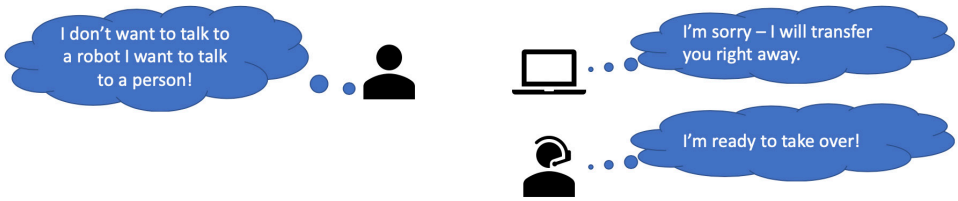


Figure 4.10 An opt-out flow

The first escalation reason is opt-out. This occurs when the user decides they do not want to converse with a virtual assistant anymore. Statements like "I want to speak to a real person" or "Get me out of here" are things an automated solution cannot solve and should not try to. A user should have this level of agency in their conversation. Some virtual assistants implicitly use a tone analysis of the user's utterances and interpret a negative sentiment or angry tone as an opt-out request. Telephonic assistants treat a touch-tone "0" as an opt-out as well.

The second escalation reason is misunderstandings. It is appropriate for the assistant to use reprompting and disambiguation on occasion - remember that humans do this too - but if the assistant is repeatedly not understanding the user will get frustrated with the experience. It is common to implement a "three strikes" rule (or "two strikes") rule, where if the assistant misunderstands the user a given number of times in a row, or total across a conversation, the user will be escalated out of the conversation.

The third escalation reason is business process. In this case the assistant understands enough about the user's request to know that the assistant cannot service the request. This escalation path is frequently coded to divert complex or sensitive business process flows away from an automated solution and towards a dedicated channel.

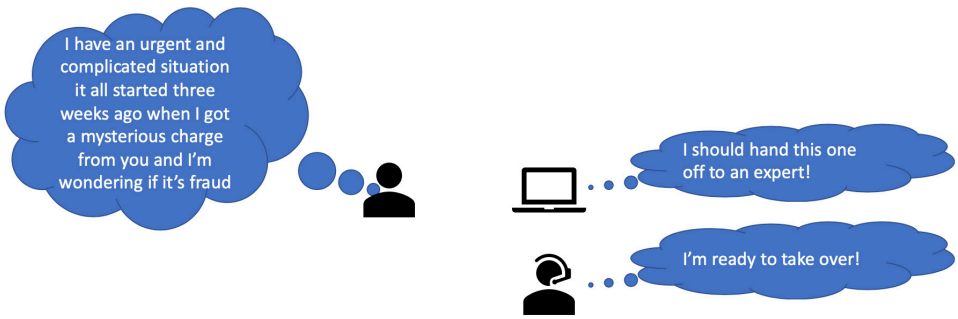


Figure 4.11 Business processes can dictate that certain conversations should be handled by a human

Regardless of the escalation reason there are multiple ways to implement escalation. The most satisfying experience for users is to stay within the same channel. If the conversation is happening over web chat, then an escalated conversation invokes a web chat backed by a

human. If the conversation is happening over the telephone then the call is automatically transferred to a human or call center. Less satisfying is ending the conversation with instructions to the user to go somewhere else - a web chat ending with "call our support line" or "email us" puts extra burden on the user. Whatever channel the user chose to interact with you in is probably the channel they prefer so you should make an effort to stay in that channel.

Note that many virtual assistant providers have built-in integration layers to automate escalations. For instance, these layers can be used to seamlessly hand off a chat to a human agent. You should have a plan for where your assistant sends users when the assistant cannot satisfy the user.

Table 5 Best practices for escalation

Do	Do Not
• Let the user request to leave the assistant (opt-out) • Help the user leave the assistant when the assistant consistently misunderstands them • Assume that some business processes will not be handled by the assistant	• Fail to recognize an opt-out request • Fail to keep track of how many misunderstandings occurred in a conversation • Keep the user in the assistant forever

4.4 Summary

- Dialog should be written with an intentional tone. The tone affects how users perceive your assistant. You should be proactive about the tone your virtual assistant uses.
- Show empathy for the user when they have a problem and let the user know that you understand them. Give cues that reflect you have understood their intent and that you are intentionally moving the dialog forward.
- There are many ways to ask a question. It is important to include enough specificity in your questions to get the responses you need from users.
- The assistant will not always fully understand the user and you need a plan for when that happens. You can handle misunderstandings with re-prompts, disambiguation, or escalation.

5

Building a successful assistant

In this chapter you will learn:

- How to avoid the most common virtual assistant failures
- How to select appropriate success metrics for your assistant
- Which metrics are commonly used for which types of assistants
- How to instrument your assistant to measure the right metrics

So far in this book, we have covered WHY you would build a virtual assistant (Chapter 1), and HOW to build a virtual assistant (Chapters 2-4). In this chapter, we will take a different approach. We will examine the practices that make virtual assistants succeed. Specifically, we will focus on how to tell if a virtual assistant is making a positive impact on your bottom line.

When I started writing this chapter I typed "why do AI projects fail" into a search engine. This query returned over 81 million results! Avoiding failure is clearly important when creating a virtual assistant. But we should aim higher than not failing and consider how to make our virtual assistant projects succeed.

In the previous chapters, we used FICTITIOUS INC as a motivating example. FICTITIOUS INC is trying to improve its customer service department by using a virtual assistant. They are getting overloaded with routine customer service questions. They've heard that many AI ventures fail, and they don't want to join that statistic.

In this chapter, we will examine how FICTITIOUS INC can succeed with their assistant. Further, we'll look at a variety of virtual assistant types in a variety of industries. Table 5.1 shows some virtual assistant use cases, and in this chapter, we'll look at how each type of assistant can be successful. Specifically, we will look at the key business objectives for each type of assistant and the most common metrics used to assess if the assistant is successful.

Table 5.1 Virtual assistant use cases sorted by virtual assistant type

Virtual Assistant Type	Industry/Use Case
Conversational Assistant (self-service)	Retail customer support – (ala FICTITIOUS INC in Ch 2)
Conversational Assistant (agent-assist)	Insurance sellers
Command Interpreter	Smartphone
Command Interpreter	Consumer electronics (smart TV)
Event Classifier	Customer support
Event Classifier	Email classification

As described in Chapter 1, there is overlap in the technology used in each assistant type (they all use classifiers. There is some commonality in how these different assistant types are developed. However, each assistant type has different usage patterns. These differences lead to using different success metrics, or key performance indicators (KPIs), for each assistant type.

When you deploy your virtual assistant to production, you'll want to have your success metrics easily accessible to all of your stakeholders. The most common approach is to create a dashboard that displays your metrics in real time. Many virtual assistant platforms come with built-in analytics and dashboards. You can start with these dashboards, but you may need to do some custom work to track the success metrics important to you.

Success metrics differ by virtual assistant type. The assistant types we will review are:

- **Conversational Assistant** – This assistant type uses a fully conversational interface, using text or voice input. The users may be end-users (who are helping themselves, in a "self-help" scenario), or agents (who are helping other users, in an "agent assist" scenario). Users are likely to ask several questions in one session. You may also call this a "chat bot" or "voice bot".
- **Command Interpreter** – This assistant is generally used to invoke a single command ("set an alarm for 9:30" or "turn on the lights"). The assistant gathers a set of information required to execute a command, executes that command, and then leaves the user alone.
- **Event Classifier** – This assistant type is often embedded inside another tool. Users may not interact directly with the classifier. The assistant analyzes user input and makes inferences about that input. Those inferences are used to assess or route user input.

Regardless of which assistant type you are using, it's a good idea to have SMART[1] goals around the success metric(s) you use. There are slightly varying different definitions for the acronym, and I will use the following from Wikipedia:

- Specific
- Measurable

1https://en.wikipedia.org/wiki/SMART_criteria

- Achievable
- Relevant
- Time-bound

Each of the metrics in this chapter is fully compatible with a SMART goal mindset. For each assistant type, I will give Specific metrics and how to Measure them. I will discuss Achievable targets (I promise that each is Relevant). The Time-bound will vary depending on your project, but I advise using a span of less than a year.

Let's start by exploring how conversational agents succeed.

5.1 Conversational assistant success metrics

Conversational assistants are perhaps the most common type of virtual assistant created today. Conversational assistants are deployed because the problems they solve cost companies a lot of money.

FICTITIOUS INC deployed a virtual assistant so that they could deflect calls from their human-staffed customer service department. Every customer service inquiry costs money to handle (not to mention any impact from complaints on revenue). Self-service conversational assistants are used in many different industries, especially retail and insurance, to reduce total customer service expense. It is not unusual for a human-handled call to cost several dollars, and for a virtual assistant-handled call to cost one dollar. The savings can quickly add up. Further, a virtual assistant can provide answers faster than a human agent, especially since it can handle requests from multiple users at the same time.

Agent-assist virtual assistants are used in industries with high turnover, such as customer service, sales, and technical support. It can cost thousands or tens of thousands of dollars, and weeks or months, to train a new employee to full self-sufficiency, and then the employee might leave. Agent-assist assistants can dramatically reduce the time to self-sufficiency. This both reduces training costs and improves customer satisfaction.

Conversational agents most commonly use the following success metrics:

- **Containment**: How many requests does the assistant complete (start to finish), without transferring the user to another system?
- **Time to Resolution**: From start to finish, how quickly does the user get the information they need?
- **Net Promoter Score (NPS)**: Does the user have a negative or positive experience? Will they brag about it to their friends?
- **Coverage**: How many requests does the assistant attempt to answer, without transferring the user to another system?

Let's explore each of these metrics in detail.

5.1.1 Containment

A **contained** session is one where the user never leaves the assistant. This is the most popular metric for self-service conversational agents. Figure 5.1 demonstrates the concept of containment.

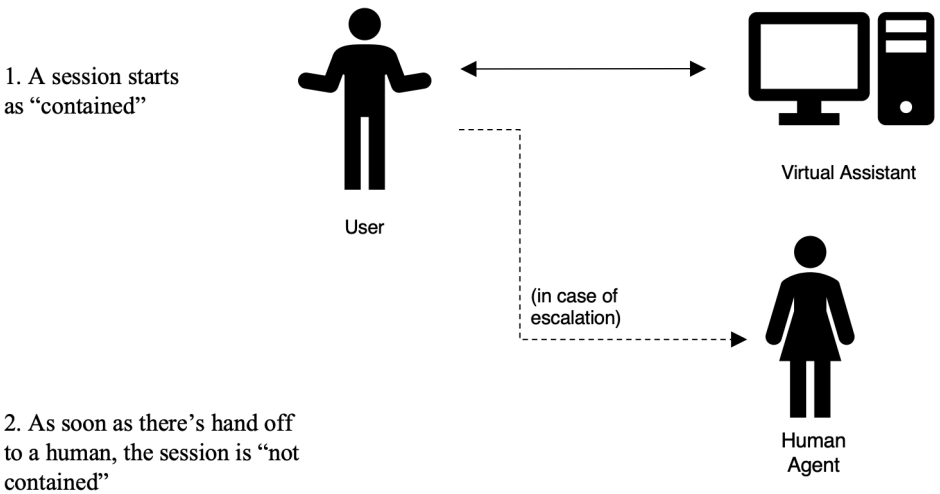


Figure 5.1 A session with a virtual assistant starts as "contained". If there is an escalation, the session is "not contained".

The simplest definition of containment is "how often were requests NOT directed to another channel".

Containment is a metric commonly used in call centers and interactive voice response (IVR) systems. Containment is important when the cost disparity between resolution channels is high. Before virtual assistants were popular, call center employees would be measured on containment (how often they resolved a call without escalating to a manager). Measuring a virtual assistant on containment is a natural extension of this. Table 5.2 demonstrates how several different conversations rate on containment.

Table 5.2 Containment analysis of several conversations

Contained	System: "How can I help you?" User: "When are you open today?" System: "We are open from 9AM to 9PM" User: (Hangs up)
Not Contained	System: "How can I help you?" User: "When are you open today?" System: "I'm sorry, I'm not trained to answer that question. Let me transfer you to customer service."

The reasoning behind containment is simple. Every phone call, or chat session, completely handled by your virtual assistant is by definition not using your human agents.

Each of those sessions can be claimed as cost savings against the human cost. If it costs $5 for a human to handle a call, and $1 for a virtual assistant to handle a call, the assistant breaks even at 20% containment. Table 3 demonstrates the impact of containment on a company's bottom line.

Table 5.3 Total costs vary by containment. Assume 100 calls where a virtual assistant costs $1 to handle a call and a human costs $5.

Contained	Not contained	Virtual assistant costs	Human costs	Total costs
10	90	$100	$450	$550
20	80	$100	$400	$500
30	70	$100	$350	$450

Table 5.3 should be compared to the baseline scenario – where no virtual assistant exists. In this scenario 100 calls cost $500. When a virtual assistant is the first point of contact it receives all the calls (or chats) first. Thus, the break-even point in Table 3 is at 20% containment. If the virtual assistant can exceed 20% containment, it will start saving you money!

From a SMART goal perspective, it's a good goal to target a high enough containment rate such that you save money.

However, you should not abuse or cheat this metric! Table 5.4 shows additional containment scenarios. These scenarios show why containment is not the end-all success metric. Containment should be combined with other metrics to measure the success of your assistant.

Table 5.4 Containment analysis of additional conversations

Contained	System: "How can I help you?" User: (Hangs up)
Not Contained	System: "How can I help you?" User: "I need to speak to a human." System: "Let me transfer you to customer service."
Contained	System: "How can I help you?" User: "When are you open today?" System: "Sure, I can help you reset your password" User: (Hangs up)
Not Contained	System: "How can I help you?" User: (Repeatedly presses the "0" button on their phone) System: "Let me transfer you to customer service."

It may surprise you that if either party disconnects the session, the session is contained. You've probably had a bad customer service experience, where they were trying to get you to hang up. In Figure 5.2, Dialog 1 shows that user disconnections are always counted as contained, and Dialog 2 shows artificial containment by ignoring an opt-out request. Neither dialog is a containment you should feel good about.

Figure 5.2 Do not cheat containment! It's an imperfect metric!

The simplicity of containment, and its direct impact on the bottom line, make it an important metric for evaluating the success of a conversational assistant. Table 5 lays out the pros and cons of containment as a success metric.

Table 5.5 Pros and cons of containment as a success metric for conversational assistants

Pros	Cons
• Easy to measure • Directly aligns with the bottom line • Industry standard	• Susceptible to gaming • May not align with user needs

Containment cannot be the only metric, especially since it does not truly measure user satisfaction. Let's move on to metrics that consider the user's perspective.

What if the session has some success before the session is transferred?

Sometimes a session will have one or more successful interactions before there is an interaction that causes a transfer. How is a call like that measured?

In the strictest definition of containment, *any* transfer means the session is not contained. Full stop. However, several practitioners use a "partial containment" metric, or a similar metric, that gives the system credit for any requests fulfilled before a transfer.

5.1.2 Time to Resolution

Time to resolution is a popular metric across both self-service and agent-assist scenarios. Simply put, time to resolution is the time between a user initiating a session and the user getting resolution. Corporations frequently build virtual assistants so they can decrease time to resolution for their users. Figure 5.3 demonstrates the time to resolution concept.

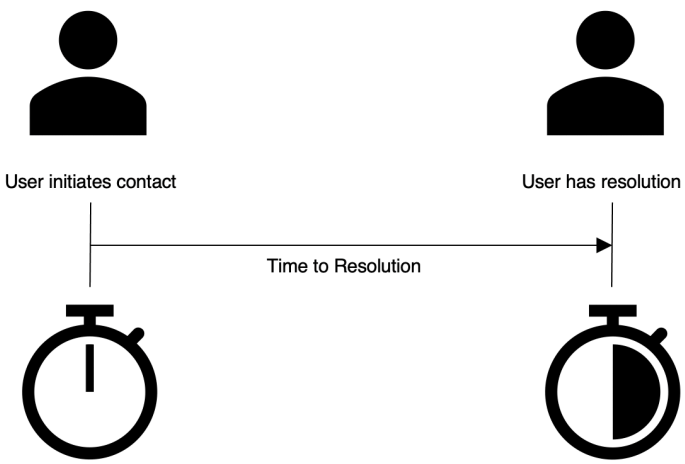


Figure 5.3 Time to Resolution, explained

A good virtual assistant can greatly reduce the average time to resolution. This may seem counterintuitive – human experts can provide answers pretty quickly when you ask them. How much better can a virtual assistant be?

In the case of a self-service assistant, the virtual assistant gets a head start. When you open a session with a virtual assistant, it's ready to respond right away. When you open a session with a human agent, you often have to wait in a queue or be placed "on hold", before you ever get to a human. If your virtual assistant can resolve a session without any human involvement, the time to resolution can be quite low. Figure 5.4 demonstrates the head start that self-service virtual assistants have over human agents

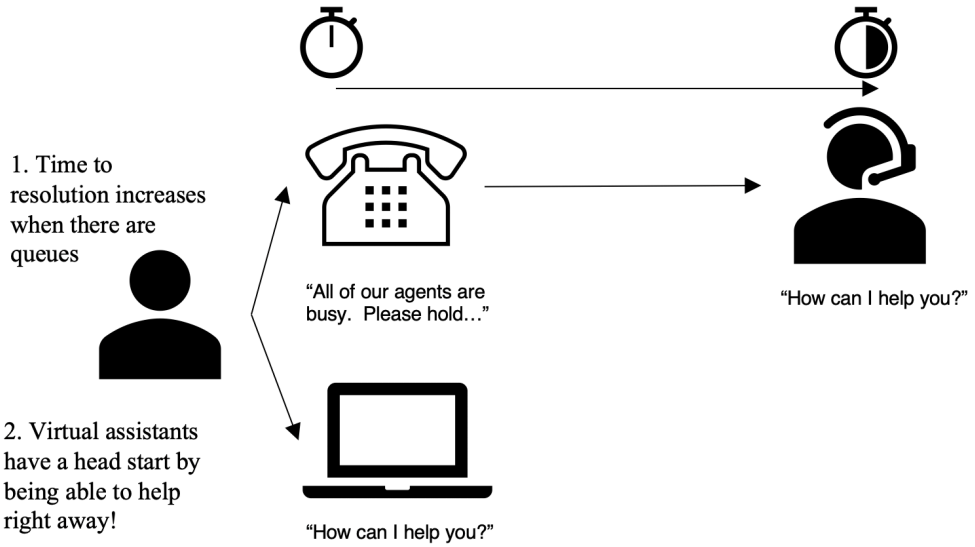


Figure 5.4 Time to resolution is heavily impacted by wait times. Virtual assistants can serve many users at once, greatly reducing wait time and time to resolution.

For agent assist virtual assistants, time to resolution is decreased because the human agent works with the virtual assistant to resolve issues. How many times have you been on the phone with customer support and you were placed on hold? Agents place customers on hold so they can go look up the answer online, or ask a colleague, or mutter to themselves while they thumb through a manual. With an agent-assist assistant in the picture, the agent will have more answers at their fingertips and will be able to resolve issues more quickly. Figure 5.5 shows how agent-assist virtual assistants reduce time to resolution.

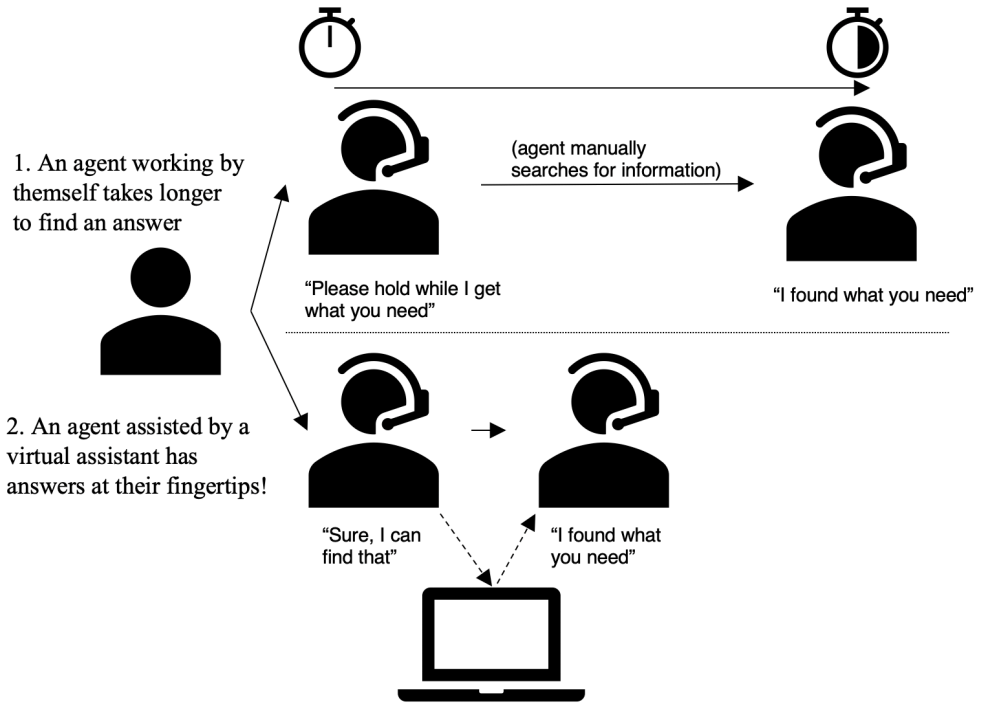


Figure 5.5 Agent-assist virtual assistants speed up time to resolution by giving the human agent the information they need, when they need it.

It's important to note that you only get decreased time to resolution if your assistant is smart enough to resolve user requests, or at least gather the information required to resolve them. A virtual assistant that constantly responds with "I'm sorry, I didn't understand" is not helping the user.

Time to resolution is a very useful metric. Time to resolution is more resilient to attempts to "game the metric" as long as you pick a good resolution marker. A simple marker, but one easily gamed, is how long until the user disconnects. (Don't game time to resolution by trying to make the user hang up!) A more appropriate marker for time to resolution is when the assistant provides useful information to the user, or when the user states that their request is satisfied.

One shortcoming of time to resolution is that it can be difficult to decide when a request is truly resolved. You can use implicit resolution markers (user disconnects, or assistant sends some information), or you can ask the user directly ("does this resolve your request?). Explicit markers are more precise, but users often disconnect before confirmation questions are asked.

For a SMART goal, you should at least aim to decrease the current average time to resolution. The assistant will resolve some calls without a transfer; other users will use your

assistant then transfer (and potentially sit in a queue). The callers that get transferred will have a longer time to resolution. Be sure that enough calls are resolved without transfers that the average time to resolution decreases.

Table 5.6 lays out the pros and cons of time to resolution.

Table 5.6 Pros and cons of Time to Resolution as a success metric for conversational assistants

Pros	Cons
• Easy to measure • Fast resolution aligns with user needs	• May require using an implicit marker in conversation to determine when resolution is achieved • Users may disconnect before you know if an issue is resolved

If users are having their requests resolved quickly, they should be happy. Next, we will look at a metric that determines how pleased (or displeased) they are.

5.1.3 Net Promoter Score (NPS)

Net Promoter Score[2] (NPS) is based on responses to the following question: "How likely is it that you would recommend our company/product/service to a friend or colleague?". Users respond with a score between 0 and 10. High scores (9 and 10) are the gold standard – your service is so great that users will brag to their friends about it. The lowest scores indicate your service is so bad that users will tell their friends to avoid it!

A variation on Net Promoter Score is to use an arbitrary satisfaction survey at the end of the virtual assistant interaction. The precise scale is less important than gathering direct feedback from your users.

Corporations frequently do Net Promoter Score surveys of their customers independent of their virtual assistants. Corporations often launch virtual assistant projects to help with Net Promoter Score. If a company has long wait times for customer service, or field agents who take too long to resolve issues, or human agents who simply don't know the right answers, that company may already have an NPS problem. Virtual assistants can improve NPS by quickly resolving issues and thus pairs nicely with time to resolution as a success metric.

When the user provides a Net Promoter Score, there is no need to guess if the user is satisfied – they've just told you! The direct connection to the user, and the difficulty in gaming NPS, make it a valuable success metric.

There are two primary criticisms of Net Promoter Score. The first is that users frequently don't answer surveys – perhaps only 5-10% of your users will answer a survey question. This sampling of users may not be representative to draw bad conclusions from. (If you average a 0 NPS, you can certainly draw conclusions from that!)

The second criticism of NPS is that it you may not be able to tell why the user is unhappy. If the user asks, "Am I covered to see a specialist?" and the assistant accurately answers

2https://en.wikipedia.org/wiki/Net_Promoter

"No, your plan does not cover specialists", the user may provide a low NPS score even though the assistant performed well.

Finally, it is only appropriate to ask the NPS question after the assistant has resolved at least one issue. As shown in the following sample dialog, the survey should be the third question you ask the user at the earliest

1. Intent Capture: "How can I help you?"
2. Intent Resolution: "Can I help with anything else?"
3. Survey: "On a scale of 0 to 10, how satisfied are you with your service today?"

The implicit contract between user and assistant should be that the assistant can only *ask* for something from the user after it has *given* something to the user. It's best to ask the survey question when the user is done.

Further, it does not make sense to ask a satisfaction question if the assistant will be transferring the user to a human agent. Partly because it's highly unlikely that the user is satisfied, and partly because the agent can ask a survey question after they have resolved the user's issue. That survey can cover both the agent and the assistant. If you are measuring containment in addition to NPS, the assistant will be sufficiently "punished" by the containment score

For a SMART goal, aim to improve your current NPS in the first few months after deploying your assistant.

Table 5.7 summarizes the pros and cons of Net Promoter Score.

Table 5.7 Pros and cons of Net Promoter Score as a success metric for conversational assistants

Pros	Cons
• Easy to measure • Direct reflection of how users actually feel • Difficult to game	• Users frequently disconnect without answering surveys, leading to small sample sizes • Users may give you a bad score for a "right" answer that they don't like

Both self-service and agent-assist virtual assistants can improve a company's NPS rating. Let's look at one last common metric used to evaluate conversational assistants.

5.1.4 Coverage

The last metric commonly used to evaluate is *coverage*. Coverage can be used to evaluate both self-service and agent-assist types of conversational assistants. Simply put, coverage is a measurement of how many requests the assistant tries to answer. In Figure 5.6, Dialog 1 is covered and Dialog 2 is not covered, giving our assistant a 50% coverage score.

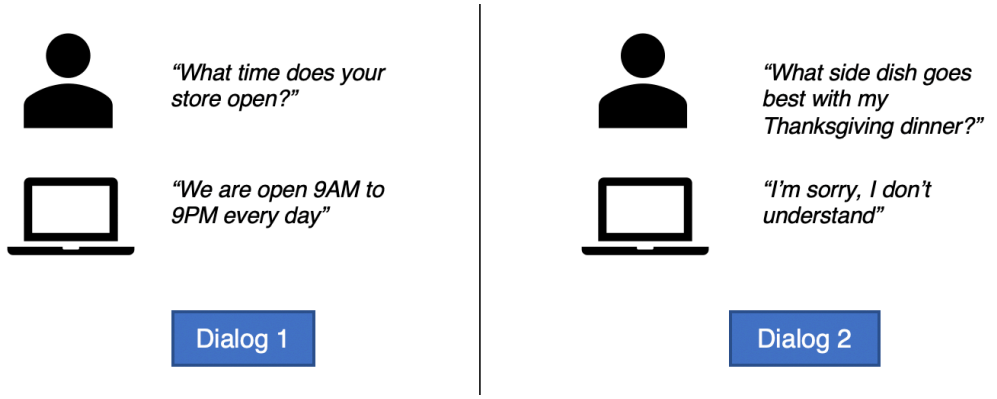


Figure 5.6 Demonstration of coverage. Dialog 1 is covered because the assistant tried to answer. Dialog 2 is not covered because the assistant did not try to answer.

An assistant with very low coverage is not providing much value to users. When coverage is low, the user is likely to keep getting responses like "I'm sorry, I didn't understand" until the assistant decides to transfer the user to an alternate channel. In this case, the user would probably prefer to go straight to that other channel, without having to go through your assistant first!

High coverage can be guaranteed by giving a targeted response to every single request. However, the more requests you answer, the more likely you are to give an incorrect answer. Figure 6 showed an assistant with 50% coverage. If we made that assistant less conservative about when it responds to a user, we could end up with Figure 7. The assistant might classify "What side dish goes best with my Thanksgiving dinner?" as a `#Store_Location` intent with low confidence. If the assistant responds based on that low confidence intent, the coverage will go up, but the user will have a worse experience.

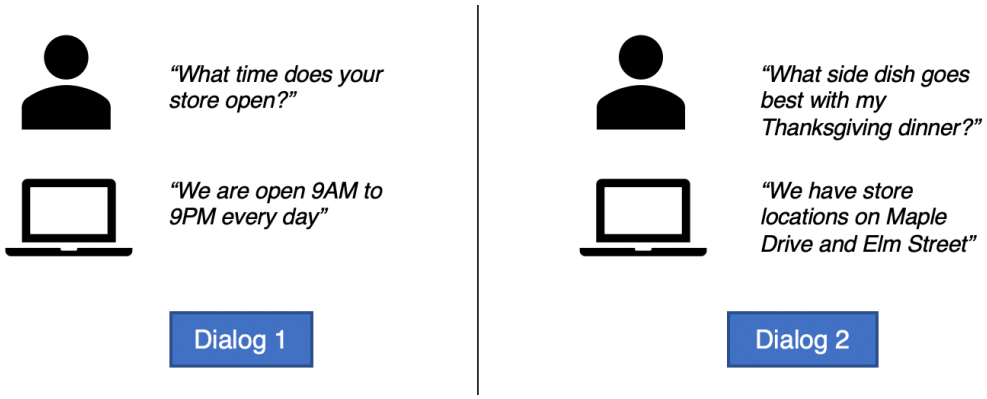


Figure 5.7 Cheating the coverage metric by responding even if the assistant is not sure about the intent.

Dialog 1 is covered and effective. Dialog 2 increases coverage but is not effective.

Coverage is achieved by answering a lot of questions, but effectiveness is measured by how many responses are useful. In Figure 5.7, the assistant provides an answer to Dialog 1 that's useful – this conversation counts towards both coverage and effectiveness. Dialog 2 gives a completely wrong answer – the response increases coverage but decreases effectiveness.

Solving the tension between coverage and effectiveness

Coverage and effectiveness have a natural tension. The more questions you answer, the more likely you are to give wrong answers. Each question you answer increases coverage, each wrong answer decreases effectiveness.

Chapters 8 discusses how to train your assistant (to optimize coverage), and Chapter 9 discusses how to test your assistant (to optimize effectiveness).

In Chapter 2 we implemented a low-confidence response filter for FICTITIOUS INC so that their assistant only responded when intents were recognized with high confidence. This decreases coverage but increases effectiveness and improves the user experience.

For agent-assist conversational assistants, there is less downside to having high coverage. The agent can always choose to ignore any advice coming from the virtual assistant. But, if the assistant gives too much irrelevant advice to the agent, the agent may tune out the virtual assistant altogether

Coverage is most useful when it is used to determine how to improve the assistant in the future. When you analyze the performance of your virtual assistant, look at all of the user requests that were not covered. This set of data shows you what kinds of requests your users really want. For instance, if you find a lot of meal planning questions like "What side dish goes best with my Thanksgiving dinner" you could plan a new `#recipes` or `#guided_shopping` intent.

In Chapter 3 we discussed how to plan the implementation of your virtual assistant. During your planning phase, you'll probably identify more intents you want to service than you can easily implement. My advice was to choose high-value intents and start with those. Once your assistant goes into production, the uncovered conversations can be more helpful in prioritizing new functionality into your assistant than your original plan was.

For a SMART goal, follow the Pareto Principle[3], also known as the 80/20 rule. You may never cover 100% of user requests, but if you can identify and respond intelligently to the most common requests, you will provide value to yourself and your users. You may need to adjust this target, but 80% is a great place to start for both coverage and effectiveness.

Table 5.8 summarizes the pros and cons of coverage as a success metric.

3https://en.wikipedia.org/wiki/Pareto_principle

Table 5.8 Pros and cons of Coverage as a success metric for conversational assistants

Pros	Cons
• Easy to measure • Shows where bot capability may need to be expanded	• Cannot be used as the only success metric • Does not measure if responses are useful (effective); only that they are given • May not be aligned with user goals • Susceptible to gaming

5.1.5 Instrumenting your conversational assistant

Your virtual assistant platform may automatically gather some of these metrics for you. Some metrics can be determined generically, without any knowledge of your assistant. Other metrics require a detailed understanding of your specific assistant. For these metrics, you'll have to help out the assistant by writing some code

For instance, containment can be generically measured by many virtual assistant platforms, especially if they provide built-in integrations to escalation channels. Transfer or escalation scenarios are often handled by a single dialog node. Containment can be measured by counting the number of visits to that dialog node, divided by the total number of conversations.

Figure 8 demonstrates a manual instrumentation of containment. (Your virtual assistant platform will offer similar capability.) In this diagram, any transfer will add the `context_connect_to_agent=true` flag to a conversation. If you also update the "conversation start" dialog node to include `context_connect_to_agent=false`, you can use the `context_connect_to_agent` variable value to count how many conversations are, and are not, contained.

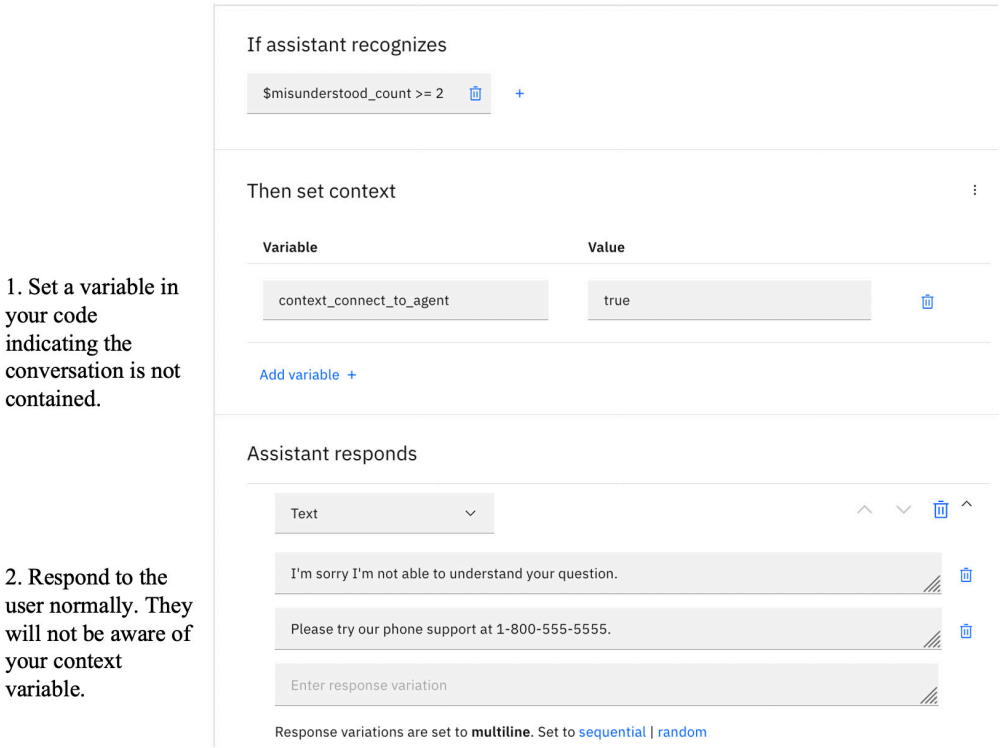


Figure 5.8 Manually instrumenting a virtual assistant to measure containment.

Instrumenting your conversational assistant to help measure the most common success metrics generally only requires one of these actions, though you can mix and match them depending on your platform:

1. Recording which dialog nodes, or dialog responses, align with success metrics
2. Adding variables to your dialog, associated with events
3. Adding code to emit events to a reporting tool

Table 5.9 shows some common techniques for instrumenting each of the common conversational agent success metrics.

Table 5.9 Instrumentation methodologies for common conversational assistant success metrics

Metric	Common instrumentation techniques
Containment	• Define a "transfer" or "escalation" node • Set a variable for containment status • Emit "conversation started" and "conversation transferred" events
Time to resolution	• Define which dialog nodes equate to resolution • Compute time between conversation start and visiting a resolution node • Emit "conversation started" and "conversation resolved" events
NPS	• Set a variable for numerical survey result • Emit event with NPS response
Coverage	• Define a "fallback" or "default" dialog response • Emit "coverage failure" and "coverage success" events

5.2 Command interpreter success metrics

Conversational agents are frequently deployed to reduce costs for corporations – either by improving self-service options to consumers or by augmenting the intelligence of users in agent-assist deployments. Conversational agent success metrics directly focus on cost savings and time to value. Conversational agents are often backstopped by a human expert, and success is partly measured by how rarely that backstop is used.

Command interpreters on the other hand don't lead to direct cost savings. Instead, a good command interpreter makes you excited to use a product or a series of products. A command interpreter rarely exposes new capability – it just makes that capability faster to use. Table 5.10 shows how some capabilities were achieved, before and after command interpreters.

Table 5.10 How to access capability, without and with command interpreters

Scenario	Without command interpreter	With command interpreter
Set an alarm for 9PM on your phone	1) Find the Clock app 2) Find the Alarm setting 3) Use a slider to set an alarm time	1) Press the "wake" button 2) Speak "Set an alarm for 9PM"
Turn up the volume on your smart TV, from volume 15 to volume 20	1) Figure out which of the many, many buttons on your remote is "Volume Up" 2) Press the button several times	1) Press the "wake" button 2) Speak "Set volume to 20"
Order new paper towels online	1) Open a browser or app 2) Search for paper towels 3) Select the product you want 4) Click Buy	1) Press the "wake" button. 2) Speak "order more paper towels"

The capabilities in Table 10 are nothing earth-shattering. We've long been able to set alarms, increase television volume, and order more paper towels online. What makes the command interpreter exciting is how quickly the capabilities can be enabled, once the appropriate device is in hand. Table 5.11 shows a dramatic reduction in time to value for these same scenarios.

Table 5.11 Time to value without and with using a command interpreter

Scenario	Time without command interpreter (expert user)	Time with command interpreter
Set an alarm for 9PM on your phone	18 seconds	5 seconds
Turn up the volume on your smart TV, from volume 15 to volume 20	10 seconds	5 seconds
Order new paper towels online	30 seconds	5 seconds

Voice-enabled command interpreters are just fast! As fast as you can say what you want, it can happen. In Table 5.11 I did an apples-to-apples test by timing each action while having a device in my hand. Many popular command interpreters can be set to an "always listening" mode, helping them further reduce time-to-value.

Should a device be "always listening?"

A device that is always listening can respond to user requests more quickly than a device that needs to be "woken up" first. However, there are considerable ethical and privacy concerns with a device that's always listening, even if a user opts-in to that service. Tread carefully if you create a command interpreter that's "always listening".

5.2.1 Usage

Time to value is an important design consideration for conversational assistants. However, it is difficult to draw a direct connection between time to value and your bottom line. If I use a voice command to set a lot of alarms on my phone, that does not make my phone manufacturer any money, and it does not save them any time either. My smart TV manufacturer is not directly affected whether I change the volume by voice command or by button press.

The closest connection to bottom-line impact comes from the "order more paper towels" scenario. If I can quickly order paper towels via a home assistant, I'm going to be ordering paper towels from the same retailer that created the assistant. For this specific type of command interpreter assistant, the connection to real money earned is clear. For all other command interpreters, the value is still indirect.

Usage is perhaps the best success metric available for any command interpreter. As a producer of command interpreters, you can infer that your users like the command interpreter if they use it and use it a lot. My phone lets me set alarms through a dedicated app or a voice-enabled command interpreter. When I set multiple alarms per day through the voice option, the manufacturer can assume I'm happy with the command interpreter! For measuring usage, you can use either the total number of times the command interpreter is used or the relative percentage of how often the command interpreter is used versus other methods

Further, the way the command interpreter is used (and not used) can give a hint as to how well it is performing. I set a lot of alarms and reminders on my phone – clearly, the command interpreter works well on those. I rarely use the command interpreter on my phone to play music, open apps, or to send text messages. This is useful data for my phone manufacturer, and they can use it to further improve their command interpreter.

It's challenging to describe a one-size-fits-all SMART goal for usage. You will want to see an increase over time in the number of users who use the assistant, as well as the number of times they use it. Usage has a natural upper bound. Recall setting an alarm on a phone – there are only so many alarms one needs to set every day. You may convert users to setting all their alarms via your command interpreter (instead of via app), but you probably won't increase the number of alarms they set.

Table 5.12 shows the pros and cons of Usage as a success metric.

Table 5.12 Pros and Cons of Usage as a success metric for command interpreters

Pros	Cons
• Easy to measure • Shows where bot capability may need to be expanded • Directly aligned with user value	• Does not usually tie directly to the bottom line • Does not tell you why a feature is not used

5.2.2 Stickiness

Stickiness is the only other success metric I associate with command interpreters, and it's hard to measure directly. However, stickiness is directly correlated with income and is thus worthy of discussion.

If your command interpreter has a lot of usage that suggests its users like to use your products and services. Most cell phone users in the United States get their phones through multi-year contracts. It behooves the manufacturers to have a good command interpreter on their phones. The more I use the voice assistant on my current phone, the less likely I am to want to switch phone manufacturers at the end of my contract. This phenomenon is often called "stickiness" – either because users are more likely to "stick around", or because it's a little like a "sticky" adhesive.

Stickiness is not just true of phones. Consider any smart device in your home – TV remote, thermostat, or security system. If you get used to and really like the command interpreter interface from one device, you're likely to want to buy more devices from that manufacturer, especially if they're compatible with the interface you already know.

You may consider "stickiness" a fanciful way of saying "vendor lock-in". There may be some truth to this. It is still true that if you provide users plenty of value now, they are likely to have a positive impression of you in the future.

Stickiness is hard to measure because it can occur over a long-time scale, and the data required to measure it may not be directly available. The opposite of stickiness is almost impossible to measure (if someone stops using your product, and buys a competitor's product, how would you know?) Ideally, you would segregate sales or subscription data by how often consumers used your command interpreter and determine if the command interpreter users are buying more (or less) of your product.

Improvements in your command interpreter will take a long time to influence stickiness. If my phone manufacturer released a fantastic new feature today, and I loved it, it's still a few years until I'm ready to buy my next phone. This lag time can be even worse in other industries – I may like my smart TV, but it'll be several more years until I buy another TV. Stickiness is perhaps best thought of as the method to connect usage to revenue.

If stickiness is your success metric, your SMART goal needs to include an increase in sales or revenue in the time following your assistant's deployment.

Table 5.13 lists the pros and cons of using stickiness as a success metric.

Table 5.13 Pros and cons of stickiness as a success metric for command interpreters

Pros	Cons
• Connects usage to the bottom line	• Difficult to measure • Takes a long time for correlations to show up

5.2.3 Instrumenting your command interpreter

The command interpreter is an interface to invoking commands. The usage metric is mostly simply measured by counting how many times the interpreter was used. The only question is where to do the counting. Figure 5.9 lays out the alternatives – should you measure usage on the command interpreter itself, or on the commands?

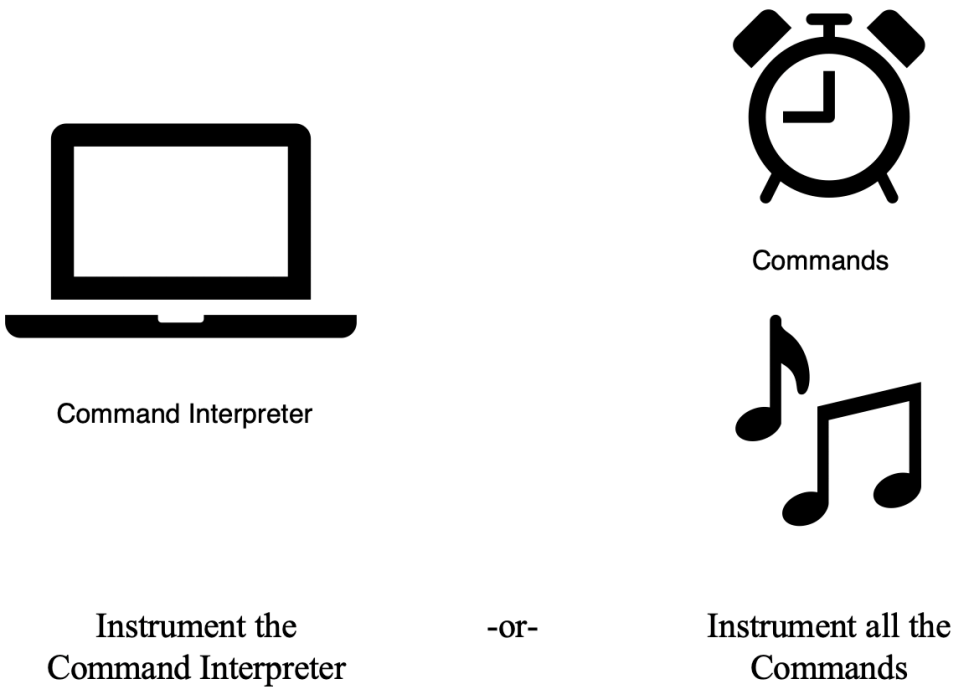


Figure 5.9 Command interpreter instrumentation options - instrument the interpreter or each of the commands

Instrumenting the command interpreter is the simplest option. Every time you execute a command, you can emit an event to a reporting solution about which command was

executed. The positive of this approach is that you can write a few lines of code and you're done! The negative of this approach is that you will only be able to measure absolute usage, not relative usage. If my phone manufacturer does this, they can only tell how many alarms I create by voice, but not what percentage of all alarms I create by voice.

My phone manufacturer could infer some relative metrics by comparing me to other users. If the average user sets six alarms a week, and I use voice to create six alarms per week, I'm probably setting all of my alarms by voice. Or the relative usage may not matter at all. My manufacturer may learn enough about me by looking at what features I bother to activate via voice at any rate, and what features I don't.

If measuring relative usage is important for an analysis of your command interpreter, the only way you can be certain is to instrument each of the commands you enable. The instrumentation is simple: each command should report every time it was invoked, and if it was invoked by the command interpreter. The downside of this approach is that you need to make this instrumentation change to every one of the commands you support. Worse, there is a heavy maintenance cost to this approach if you ever need to change the instrumentation.

Regardless of which method you choose, instrumenting for your command interpreter generally just involves emitting a single event at the right time.

5.3 Event classifier success metrics

An event classifier examines input data and classifies it into some target. Throughout this book I have provided several examples of classification, usually classifying a user utterance against a set of intents. Classifiers are frequently deployed outside of conversational systems. For instance, classifiers are used to classify email (is it spam or not spam?) and to route customer support requests to the appropriate expert (is it a hardware or software issue?)

Figure 5.10 shows an exemplary classifier use case. A classifier can be assigned to a generic email address like info@fictitiousinc.com and can direct any mail it receives to the appropriate department to be answered. FICTITIOUS INC might use the classifier to send mail to sales, legal, HR, and customer support.

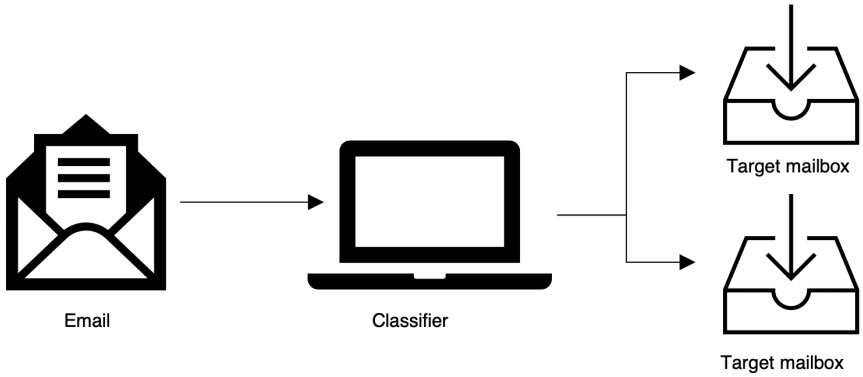


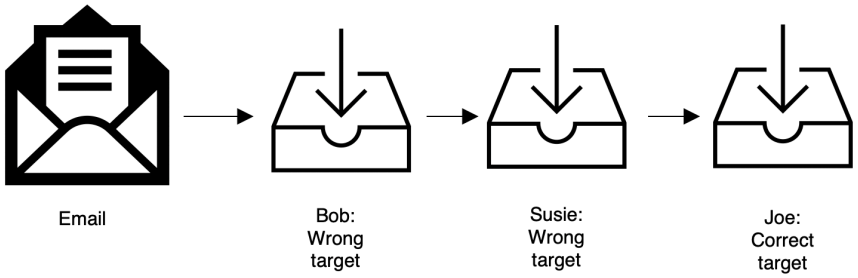
Figure 5.10 Example email classification flow. A classifier can listen to a functional email address like info@fictitiousinc.com and route the email to the person best suited to handle it.

While developing a classifier, the most useful metric is usually accuracy. How often did the classifier pick the correct classification target? However, accuracy does not directly impact the bottom line. Rather, accuracy generally influences some other metric that is connected to monetary value, generally through cost savings. Let's explore a couple of these metrics.

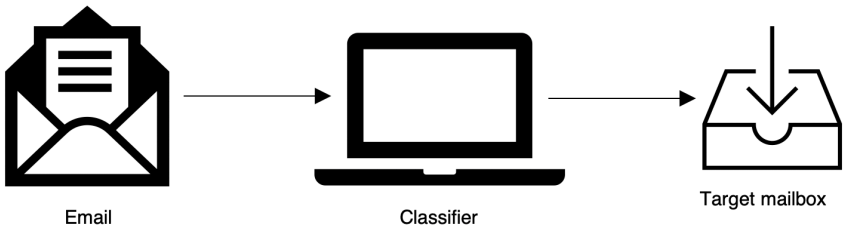
5.3.1 Time to resolution

Time to resolution is defined in a similar way for classifiers as it is for conversational assistants. A clock starts ticking as soon as the user makes a request and stops when they get a useful answer.

Figure 5.11 shows how a classifier can decrease time to resolution. In the first example, Bob may pick up an email out of the info@fictitiousinc.com functional mailbox. However, he may not know how to answer the email and may forward it on to Susie. The email could sit in Susie's inbox a few hours (or days!) before she sees it, and realizes that she too can't answer it, forwarding the message to Joe. Thankfully, Joe can answer the email. In the second example, a classifier is used, and the correct recipient is identified the first time.



Without a classifier, an email may be forwarded several times until it's sent to the right destination. Each handoff causes wait times and delays.



A classifier is likely to route the email to the right destination the first time, reducing delays.

Figure 5.11 Comparing email routing with and without classifiers

In either scenario, the time spent waiting for the first person to see the email is the same. The non-classifier scenario has a higher time to resolution when the email sits in a "wrong" mailbox, waiting again to be routed. During this wait time, the end-user is not getting any resolution, and will slowly but surely get frustrated

Alternately, a classifier can be used to make sure enough information is present before a human touches the input and "starts the clock". Figure 5.12 reminds us of the example from Chapter 1. The classifier can review input from a user and immediately give them feedback about how their input will be classified and if any information is missing from their request. Without that help from the classifier, the human receiving the input might just immediately return it to the end-user saying "I need more details", and this is a frustrating response, especially if the user was waiting for days already!

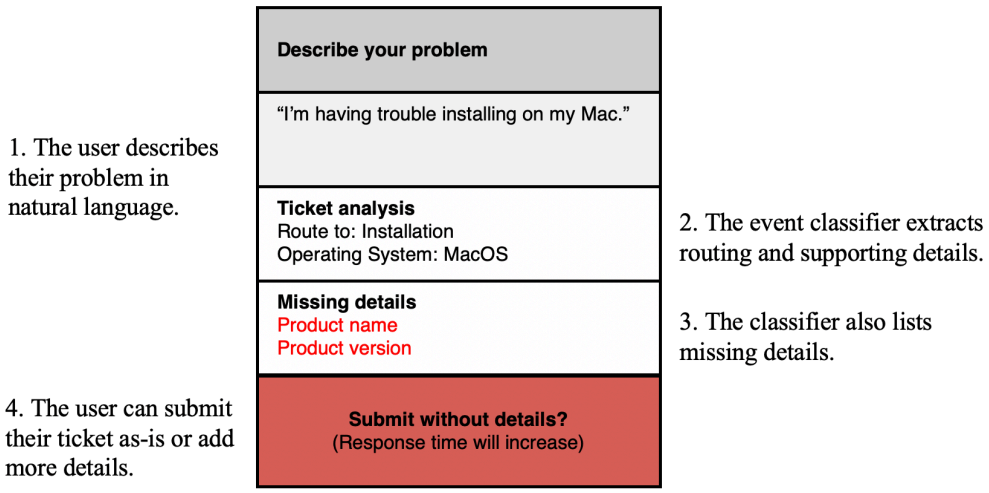


Figure 5.12 A classifier can immediately notify users of details missing from their request. The users won't have to wait for a human recipient to respond to the request with "I need more details".

Finally, an automated classifier can save time because it's "always-on" and never gets tired. Many companies use humans as classifiers (for example, "Level 1 Support"). When I call customer support, I have to wait on hold before I even get to Level 1! A classifier by contrast can handle many requests at one time and starts my resolution process more quickly.

Your SMART goal for time to resolution will depend on how long your average resolution time was before. Classifiers are often able to shave hours or days off of the average resolution time, and you may see an order-of-magnitude reduction in time. Try to at least reduce the resolution time by half.

Table 5.14 outlines the pros and cons of time to resolution as a success metric.

Table 5.14 Pros and cons of time to resolution as a success metric for event classifiers

Pros	Cons
• Easy to measure • Aligns with user value	• Only appropriate for certain classifier use cases

5.3.2 Number of hand-offs

As alluded to in the time to resolution section, the average *number of hand-offs* is a useful metric for determining how well your resolution process is working. As shown in Figure 5.11, each hand-off is an opportunity for time to be lost waiting for a response. Extra hand-offs

due to mis-routes are thus expensive both in time and in reputation. Your service looks worse to a user the longer they have to wait for a useful response.

There are multiple ways that a process could have multiple hand-offs. Here are a few:

- Email is sent to someone who cannot address it, and they must send it to someone else
- A trouble ticket is placed in the wrong work queue, and the receiving team must place it on another queue
- A support request must be escalated to a superviso

You can assume an average cost in time to resolution for every hand-off. Further, each hand-off is a waste of your company's money, when your personnel are looking at issues they cannot resolve. Finally, hand-offs can lower employee satisfaction and morale – who wants to be spending their time looking at "the wrong things"?

Number of hand-offs may require some work on your part to measure. For instance, it may involve finding out how many times an email is forwarded (how will you automatically measure this?). Trouble ticket systems often maintain an audit trail of who "owned" or worked on a ticket, but this audit trail may be difficult to parse. Depending on what process your classifier is helping to address, number of hand-offs may still be worth measuring.

Your SMART goal for number of hand-offs will depend on how many hand-offs you averaged before the classifier. The 80/20 rule is a good guideline here. Aim to have zero additional hand-offs for 80% of your cases that use the classifier.

Number of hand-offs and time to resolution are closely related, but it is worth tracking them both. Table 5.15 shows the pros and cons of "number of hand-offs" as a success metric for classifiers.

Table 5.15 Pros and cons of "number of hand-offs" as a success metric for classifiers

Pros	Cons
• Aligned with user value	• Indirectly linked to the bottom line • Tracking number of hand-offs may require special reporting logic.

5.3.3 Other customer satisfaction metrics

Several of the metrics described in the conversational agent and command interpreter sections of this chapter can also apply to classifiers. If you are injecting a classifier in a process you might still want to measure NPS, or customer retention, or similar metrics.

The challenge with using these metrics to evaluate your classifier is that your users probably don't interact with the classifier directly. NPS makes sense for a conversational agent because a user has probably spent several minutes talking with it, having a series of back-and-forth interactions. With that length of interaction, the user can form an instructive opinion about the assistant.

With a classifier, the user's interaction is brief or in fact invisible. As a user, I don't know if you're using a classifier, I just know if you take a long time to respond to me! If you mis-

route my email, I'm unhappy, you don't even need to ask! Every time you send my trouble ticket to someone who can't help, I get more frustrated.

Even so, user feedback is valuable. You can still collect NPS, or measure customer retention, but you should apply that metric to the entire process – not just to the classifier. A customer satisfaction metric belongs as one of your success metrics.

Table 5.16 lays out the pros and cons of using customer satisfaction metrics to evaluate your classifier.

Table 5.16 Pros and cons of customer satisfaction metrics for evaluating your classifier

Pros	Cons
• Aligned with user value and the bottom line • Difficult to game	• The metric evaluates your entire process, not just the classifier • Should be paired with another success metric

5.3.4 Instrumenting your classifier

Similar to the command interpreter section, a classifier is usually embedded into another tool or process. Therefore, the instrumentation often happens outside of the classifier itself.

The process needs to be instrumented with the following:

- Timestamp of when user input was received
- Timestamp of every time the user input was routed
- Timestamp of when the user's request was resolved

If you are using a classifier in an email-based process, this information already exists in most email headers. If your classifier is embedded in a trouble ticket system, this information is probably encoded in a history or an audit trail. Similarly, most customer relationship management (CRM) systems will also keep track of this information.

Depending on your system, you may not need to instrument your system to add this tracking data, but rather you might have to write reporting code to extract this data!

5.4 Summary

- To avoid virtual assistant failure, you must plan for success, and decide how you will measure success. You should use SMART goals to assure you are heading in the right direction with your metrics.
- Conversational assistant success metrics include containment, time to resolution, NPS, and coverage
- Command interpreter success metrics include usage and stickiness
- Event classifier success metrics include time to resolution, number of hand-offs, and assorted customer satisfaction metrics
- You may need to add code to your virtual assistant, or to its associated infrastructure, to measure success metrics

6

How to train your assistant

This chapter covers:

- Training an assistant to recognize intents from user input
- Collecting a viable training data set for your virtual assistant
- Analyzing how the volume, variety, and veracity of training data affect an assistant's performance

Training is an important part of building a virtual assistant. Good training increases the intelligence of the assistant and enables it to be more useful to your users. Training influences almost every single success metric and is important no matter which type of virtual assistant you are building. This chapter will teach you what you need to start training your virtual assistant!

FICTITIOUS INC prototyped their first virtual assistant using sample data from their virtual assistant provider. This data trained the assistant to be proficient at recognizing generic customer service intents. But what if a suitable sample data set did not exist? How could FICTITIOUS INC build a training data set and use it to teach their virtual assistant? Let's explore a scenario where FICTITIOUS INC was not given a pre-created training set and had to build their own.

Virtual assistant platforms come with samples or "starter kits" for rapid prototyping. If you are building a custom virtual assistant, you may be able to get started with this sample data. However, you will eventually need training data that is customized to your specific use case. Whether you are starting from scratch, or extending a sample data set, the steps in this chapter will help you train your assistant.

About training different kinds of virtual assistants

We will use FICTITIOUS INC's conversational assistant as an example, but the steps are mostly the same for command interpreters and event classifiers, with minor terminology differences.

6.1 How to train a virtual assistant

When we created a new virtual assistant for FICTITIOUS INC, that assistant did not know anything about customer service. A new virtual assistant comes with very little knowledge built in. It had some basic knowledge the English language, including grammatical constructs and some spellchecking. That's it! At the end, the assistant knew a few things about customer service, but only what we trained it on.

FICTITIOUS INC needs to build a training data set so that their assistant can recognize and serve their most common requests. FICTITIOUS INC has identified the most significant intents their customer service-based virtual assistant needs to handle, including store hours, store location, and password reset. Now they will need to train the virtual assistant to recognize these intents.

The generalized process for training a virtual assistant is shown in Figure 1.

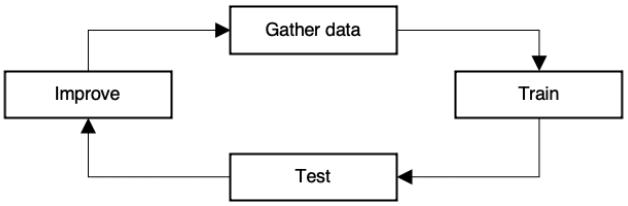


Figure 1 Training a virtual assistant is part of a continuous cycle.

For FICTITIOUS INC's conversational virtual assistant, there are two things needed to start training.

- **Gather data**: What do FICTITIOUS INC's users say? What are the textual *utterances* they will send to the virtual assistant? FICTITIOUS INC needs to find both what intents their users will have and how they will textually phrase them.
- **Train**: What do the utterances mean? The assistant will be trained on pairs of utterances with their associated intents. From this training data the assistant will reverse-engineer how to find the intent in user utterances.

Training data is always a pair of input and expected output. For FICTITIOUS INC the input is an utterance, and the output is an intent. Table 1 shows example training data for FICTITIOUS INC's assistant.

Table 1 Example training data for a conversational assistant.

Utterance	Intent
"Where are you located?"	#Store_Location
"What time do you open?"	#Store_Hours
"How can I reset my password?"	#Reset_password

You train a conversational assistant by defining intents and providing several examples of each intent. The virtual assistant's classifier uses machine learning[1] to analyze the training data, extracting patterns that it can use to identify the categories in future input data.

You train a conversational assistant by defining intents and providing several examples of each intent.

The process for training a virtual assistant is very similar to how students are taught. Students are given homework and provided an answer key to check their work. The student learns concepts from the homework and can confidently solve problems they have seen from that homework. Students do homework until they can consistently produce correct answers. From the teacher's point of view, it is tough to tell how much the student has learned – are they mastering the concepts or simply working back from the answers? Teachers assess student performance by giving exams with problems that were *not in the homework*. Grading performance on these "unseen" problems tells the teacher how well the student has learned. This analogy is demonstrated in Table 2.

Table 2 Analogy comparing virtual assistant training and student education.

Task	Virtual Assistant	Student Education
Preparation for learning	Collect a set of utterances and their associated intents as candidate training data.	Create problem sets with an answer key. Teacher assigns some of the problems as homework.
Practice and Learning	Give some training data to the assistant.	Student does homework and checks the answer key.
Assessment	Create a test set from the candidate data not included in training. Test the assistant on data it was not trained on.	Teacher creates an exam from problems not included in homework. Student takes an exam.
Declaration of mastery	High accuracy when you test the assistant with data it has not been trained on.	Student performs well on the exam.
Remediation if no mastery	Additional training of the	Additional homework and new

[1]Virtual assistants generally use supervised learning classification algorithms. For more details on how these algorithms work, read Chapter 11 and the Appendix on machine learning.

	assistant.	exams.

Why don't we train and test with the same data?

Just like a teacher does not put homework questions on the exam, we should not use the same training data to test the virtual assistant. The true test of the training process is how well the assistant classifies input it has never seen before. Most assistants correctly classify the exact data they were trained on, so testing with that data is not informative.

It's ok to have some overlap in the train and test data sets, especially when the inputs are short, as they are in many voice-based conversational assistants. Just don't overdo it.

FICTITIOUS INC's virtual assistant-building team is ready to teach their assistant. Let's help them start by finding data they can use to train their assistant.

6.2 How do you find training data?

The training process starts with finding a good source from which to build training data. This helps determine what users want and how they ask for it. For instance, FICTITIOUS INC's customer service team does a lot of "password resets" for users. FICTITIOUS INC might assume that users will typically ask "reset my password", but as they review usage data, they may find users prefer to ask, "unlock my account". An assistant will perform much better when trained based on how users actually say things, rather than using assumptions of users might say things.

A virtual assistant learns everything it knows from training data. The data used in the training process should be as close as possible to what the assistant will encounter in production. The quality of the training data determines how accurate this assistant will be at recognizing what real users ask of it. How can FICTITIOUS INC find the best possible training data?

They ultimately need to find out three things:

1. What are the intents, or categories of questions, that users will ask?
2. What is the relative frequency, or distribution, of those intents?
3. How are those intents worded?

FICTITIOUS INC has already explored the first two of these questions. As a reminder, Figure 2 shows how FICTITIOUS INC plotted the intents based on their relative frequencies and the complexity of automating them.

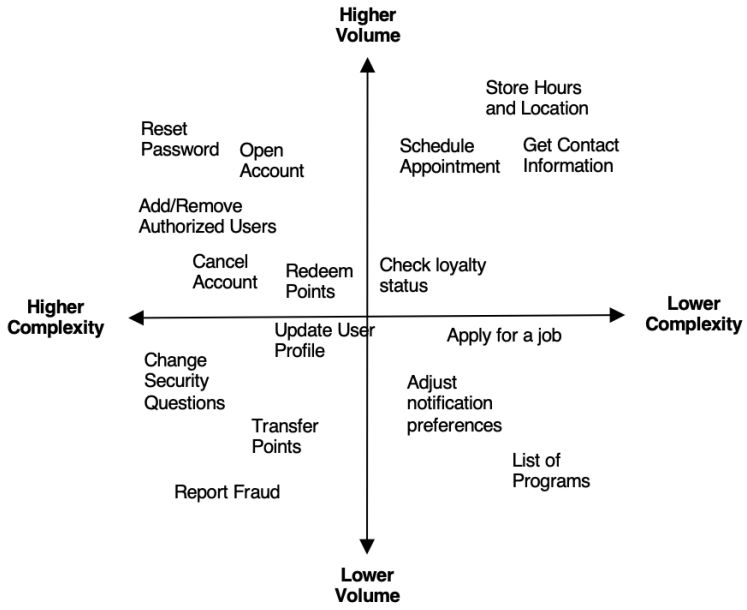


Figure 2 Example distribution of intents for FICTITIOUS INC. The assistant still needs to be trained on how these intents are phrased - what wordings are given by end-users?

FICTITIOUS INC had several potential data sources available to help them create Figure 2. Each of the sources has pros and cons. Like many virtual assistant builders, FICTITIOUS INC can use a mixture from some or all of these training sources. The primary sources include production logs, mock user interfaces, and subject matter experts. Let's first dive into production logs.

6.2.1 Production logs

The best place to find real user utterances is from a real virtual assistant in the same use case. FICTITIOUS INC would love to get training data from another customer service text-based conversational assistant. But for most people, this presents a chicken-and-egg problem: needing a production virtual assistant so that you can build a virtual assistant! Still, there are several cases where this is workable, or that a reasonable alternative exists.

When you don't have an existing virtual assistant for the same use case in production, ask if you have something close? Table 3 shows some candidate training data sources.

Table 3. Production data sources suitable for training virtual assistants

Data Source	Good For	Reason	Caveat
Human to human chat logs	Conversational assistants	Will closely mirror utterances sent to an	Will need adaptation for voice assistants –

		automated text-chat conversational assistant.	the intents will match but not the wording.
Human to human call transcripts	Conversational assistants	Will closely mirror the distribution of intents received by a conversational assistant.	People speak differently to other humans compared to how they talk to machines.
Search logs	Conversational assistants	Will closely mirror the distribution of intents received by a conversational assistant.	Search queries tend to be much shorter than conversational utterances
Emails	Event classifiers Conversational assistants	An email classifier should be trained on real email!	An email is longer than a conversational utterance. But the subject line, or first sentence, can be close to a conversational utterance.
Customer Relationship Management (CRM) event records	Any	Will closely mirror the distribution of intents	Will not provide any insight into **how** these intents are phrased by customers

The best part about using production data is that it is hard, verifiable data. Rather than theorize about what end-users might do, you can consult production data to find out exactly what they do! This is much better than guessing or theorizing.

> **"Most of the world will make decisions by either guessing or using their gut. They will be either lucky or wrong."**
>
> **Suhail Doshi, chief executive officer, Mixpanel**

In the ideal scenario, your production data would line up exactly to your new virtual assistant use case. In these cases, no caveats are required on the data source. For instance, if you are converting a human-to-human chat channel to an automated chat powered by a virtual assistant, you can reasonably assume users will ask the exact same questions to the assistant as they do to the humans.

Most times you will not be so lucky as to have an exact alignment of data sources. You can still use production data to guide your training, but you'll have to be careful. Generally,

the distribution of intents will be similar across channels, but the wording from users will differ.

Let's assume FICTITIOUS INC has access to transcripts from their human-to-human call center. This is not a perfect alignment to FICTITIOUS INC's text-chat use case. Humans talk differently to other humans than they talk or type to machines – human conversations are often more wordy, indirect, and may stray "off-topic". Figure 3 compares a typical human-to-human conversation against a human-to-machine conversation.

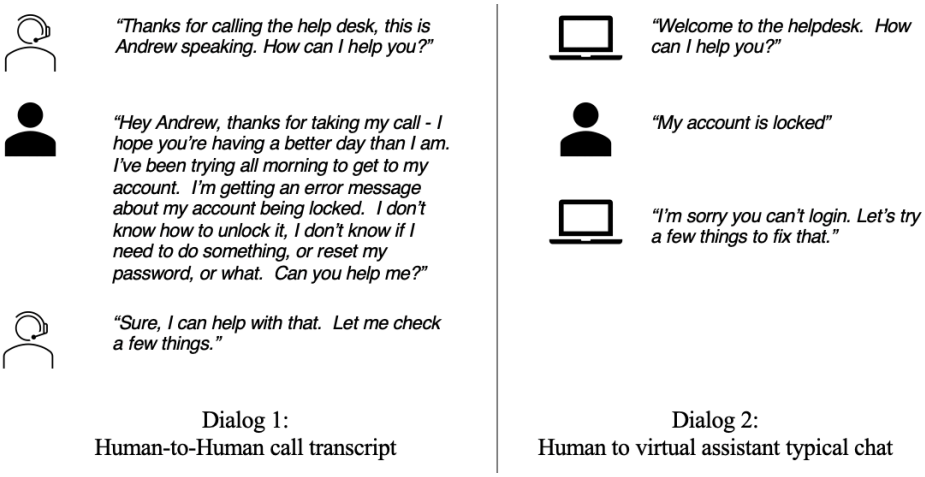


Figure 3 Comparing an actual FICTITIOUS INC human-to-human call transcript to a hypothetical human-to-virtual-assistant log, where the user needs a password reset.

Dialog 1 comes from production data. Dialog 2 is an imagination of what the same user from Dialog 1 might say if they spoke to an automated solution on the telephone. The same user who spoke 63 words to another human might only type 4 words to a bot. We can't be sure how they will phrase their utterance to the bot. We expect that some of the language will be similar and that the overall intent will be the same. FICTITIOUS INC should NOT train the virtual assistant on the full 63-word utterance.

I would use the following statements extracted from Dialog 1 to train my assistant:

- I'm getting an error message about my account being locked
- I don't know how to unlock my account
- reset my password

Each of those statements describe the "reset password" intent with no ambiguity. Each of those statements was extracted directly from the user transcript. (With one exception: in the second statement, I replaced the anaphora[2] "it" with "my account".) Figure 4 breaks

[2] A grammatical substitute, generally a pronoun, that refers to a previous word or group of words (https://www.merriam-webster.com/dictionary/anaphora)

down the transcript into fragments and identifies the role each fragment plays in the conversation.

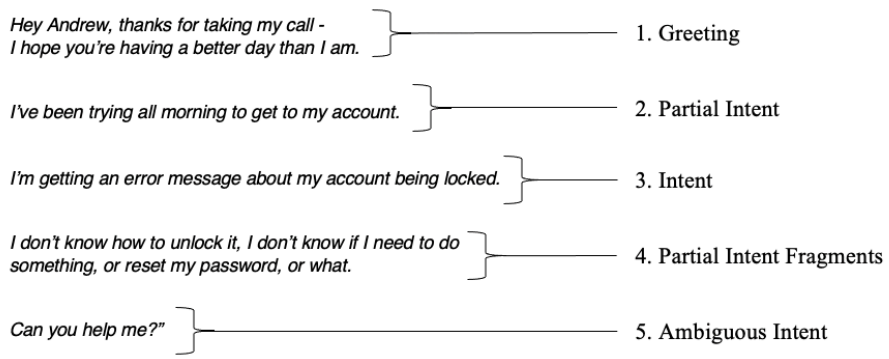


Figure 4 Breaking down the 63-word opening of a human-to-human conversation.

The remaining statements in Figure 4 are not useful for training. "I've been trying all morning to get to my account" is close but is a bit ambiguous by itself. "I don't know if I need to do something" and "Can you help me?" are so ambiguous they could apply to any intent. Similarly, no part of the greeting unambiguously hints at an intent.

FICTITIOUS INC will be able to use human-to-human transcripts and emails to train their conversational assistant by slicing and dicing the data in this way. These are great data sources for FICTITIOUS INC to use because they contain textual phrases from real users. These are data sources where the end-users are verbose, and it's easier to trim down verbose data than to augment terse data.

The best production logs include the user's actual phrasing. FICTITIOUS INC can also look at their Customer Relationship Management (CRM) records, but these usually only include a summary of the user interaction and not the user's direct phrasing. In Figure 5 we see a CRM record where the user asked a question related to a "locked account" and the representative recorded the call as a "password reset" issue. In this case, the intent is preserved but the user's wording is lost.

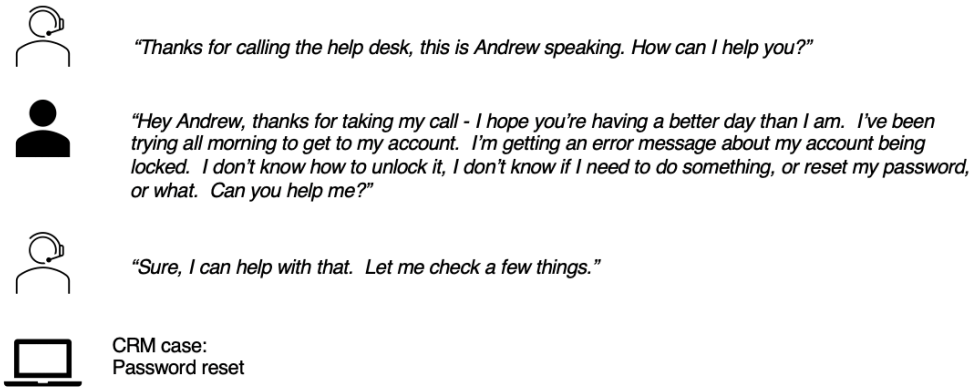


Figure 5 Beware of logs where the user utterance is lost. They only help you find intents, not the phrasing of those intents. Here the user asked a verbose question full of nuance and the representative only recorded "password reset".

Still, it is useful for FICTITIOUS INC to verify which intents are the most common in their CRM records. This will help them prioritize which intents are most useful for their virtual assistant to handle.

The best way to find out what users might say is to review what users have said before. Production logs are a great source of real phrasing from users, even if they require some adaptation. If you don't have production logs, there are still ways to get representative utterances from end-users. If you can't find out what users *have* said before, you can still find out what they *would like* to say. A "mock" user interface is one possible way – let's explore that next.

6.2.2 A mock user interface

If FICTITIOUS INC did not have access to production data, they could gather some simulated data from a "mock" user interface. A mock user interface is a prototype or "dummy" interface with no intelligence except the ability to log questions asked of the assistant. The interface looks like (or for voice sounds like) "the real thing", but it does not understand the questions. A mock interface is functionally equivalent to a survey or user interview, but it puts the user in a context closer to the intended usage and thus will gather more representative (and less fabricated) questions.

Many virtual assistant platforms can help FICTITIOUS INC build a mock user interface. An example mock interface is shown in Figure 6. A good mock user interface should:

- Greet the user and set expectations that answers will not be given.
- Allow users to ask questions.
- Log the questions and invite users to ask additional questions.

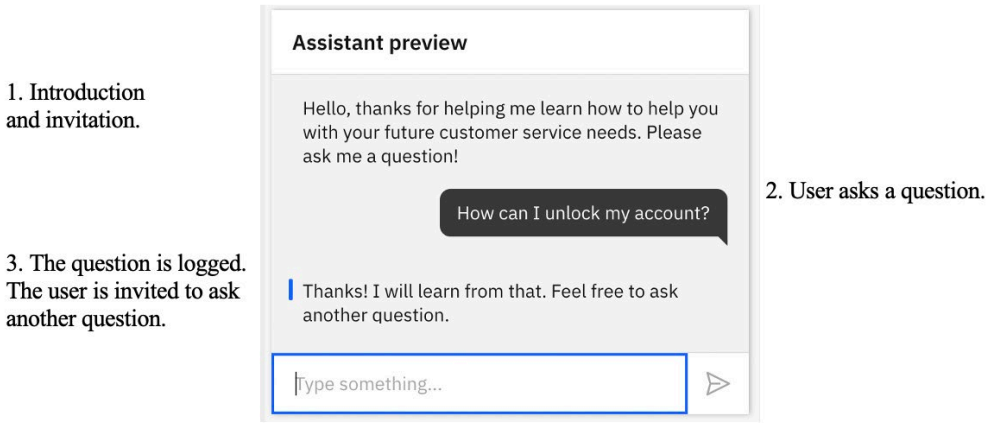


Figure 6 A sample "mock" user interface. It has the look-and-feel of a real virtual assistant. However, it does not understand anything the user says, it only logs utterances for future training.

Be polite with your mock user interface. Only expose it to volunteers. Do not violate a social contract by letting users discover this on their own, where they may have false expectations of actually getting help from the assistant. Users should explicitly opt-in to your mock user interface. You can send a link to your assistant through email or include it at the end of any survey that your users can opt-in to. Figure 7 shows an exemplar introduction to a mock user interface.

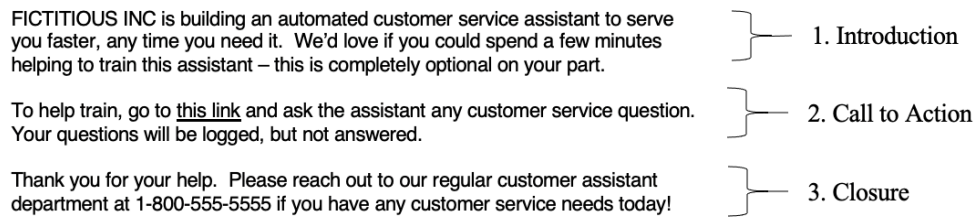


Figure 7 Example introduction to a mock user interface. You are asking users to do work for you, so be polite!

When asking for input through a mock user interface, introduce users to the interface, give a call to action, and conclude with thanks. The introduction should properly manage expectations. The call to action must strike a delicate balance. If you are too vague, the questions you collect may not be useful. If you are too prescriptive, you may bias your users and skew the way they ask questions. Table 4 shows some variations FICTITIOUS INC could use as their call to action.

Table 4 Variations of a call to action that invites participants to a mock user interface.

Call to action	Usefulness	Reason
"Ask me anything!"	Too vague	This is an open invitation for off-topic questions like "what's the weather like" and "tell me a joke". While it's fun for the participants, it doesn't help train your assistant!
"Ask me any customer service question."	Just right	Users are given an appropriate scope. The bot will not be open-ended, it will focus on customer service.
"Ask me any customer service question, like how to reset your password."	Too biased	The users will be primed to include your words in their questions. You will get mostly meaningless wording variations like "please reset my password" and "help reset my password", rather than getting a varied and representative distribution of questions.

The strength of a mock user interface is that it interacts with real users. The reason to interact with the real users is to find out the phrasing that they *actually* use, rather than the hypothetical or synthetic phrasing *you think* they will use. When you are using production logs, there is no question that the phrasing you see is actual user phrasing. A mock user interface is trying to be the next best thing.

FICTITIOUS INC must be aware of biasing participants in the mock interface. If FICTITIOUS INC asked their users "how would you ask to have your password reset", the users' brains will be primed (biased) with the words "reset" and "password". Without that bias, they may be more likely to ask about "locked accounts" or "login problems".

Build your own mock user interface!

A mock user interface can be built in most virtual assistant platforms. You only need to configure two pieces of dialog: the "welcome" node, and the "fallback" node. (Terminology may vary slightly across platforms.) The welcome node includes the greeting and initial call to action. The fallback node should thank the user and invite them to ask another question. Sample code for a mock interface built-in Watson Assistant is provided in the book's GitHub repository at https://github.com/andrewrfreed/CreatingVirtualAssistants/blob/main/code/skill-Question-Collection-Ch8.json.

It can be difficult to get enough training data from a mock user interface alone. FICTITIOUS INC can supplement the data they gathered from production logs with data from their mock interface. FICTITIOUS INC still has one more potential source of training data: their subject matter experts.

6.2.3 Subject matter experts

If you can't access user utterances directly, the next best data source is to talk to someone who knows about users. Subject matter experts (SMEs) are skilled in their domain and often interact with real users. FICTITIOUS INC can ask their senior-level customer service agents about how they interact with users in the call center. Production logs and mock interfaces give FICTITIOUS INC first-hand insight into users; interviewing their SMEs gives them second-hand insight. Second-hand is still better than none!

FICTITIOUS INC can ask their SMEs what questions users ask and how those questions are phrased. They should be cautious – this is the most heavily biased approach of them all! This approach relies on human memory, and human memory is not reliable! FICTITIOUS INC's SMEs will generally know the most common intents, though not with mathematical rigor.

The challenge in getting training data from the SMEs is when they are asked how users phrase their questions. SMEs may be overconfident in their ability to recall what users actually say. (It's very hard to know your own bias!) FICTITIOUS INC's SME might describe the password reset intent phrasing as follows: "Mostly users ask about 'password', 'reset', 'locked accounts'…. That's about it". When pressed for additional variations, they may just start reordering the keywords given previously: for instance, "reset my password" and "my password needs to be reset". They are unlikely to remember the full variety of user phrasings when asked.

FICTITIOUS INC can ask their SMEs how they train new customer service representatives. Documentation for new employees can be used to gather virtual assistant training data! Figure 8 shows an annotated excerpt FICTITIOUS INC's call-center playbook.

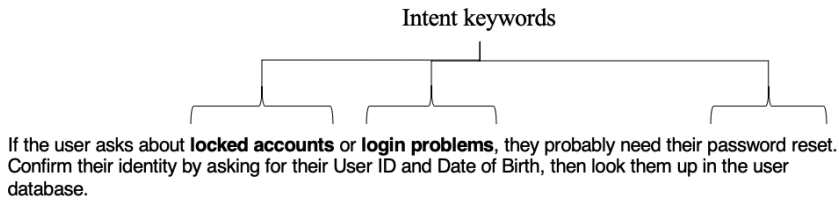


Figure 8 Training materials for customer service agents may include keywords for the intents your assistant will receive. From this material, we can infer that "locked account" and "login problem" are synonymous with "password reset".

The training material in Figure 8 gives us keyword suggestions for the "password reset" intent. The keyword phrases "locked account", "login problems", and "password reset" are treated synonymously. These keywords will not line up exactly with how users ask their questions (they may instead say "unlock my account"), but this level of variation is a nice place to start. From Figure 8 one can infer several training utterances:

- password reset
- reset my password
- locked account
- my account is locked
- login problem
- can't login

The process of inferring user utterances from keywords is heavily biased! No matter how clever you are, you can't help but include the words you've already seen. You are unlikely to come up with variations like "Unlock my account" or "I get an error when logging in" – the

latter of which has zero overlap with the previous keywords, as well as having a completely different grammatical structure. A keyword-based approach is very likely to have training gaps like these. We'll explore the impact of these gaps in the upcoming section on training data variety.

After reviewing their production logs, collecting data from a mock user interface, and interviewing their SMEs, FICTITIOUS INC has gathered a great set of user phrases that can become training data. Each utterance needs to be associated to a single intent to become training data for the virtual assistant. Let's look at how FICTITIOUS INC can find the intents in the data they collected.

6.2.4 Organizing training data into intents

FICTITIOUS INC has collected a variety of actual and potential user utterances from each of their data sources. Now they are ready to organize them by intent. Initially they did a top-down intent creation based on historical user needs. In this section they will do a bottoms-up intent creation based on the utterances users actually use.

A bottoms-up intent discovery will produce different results than top-down

Discovering user needs is a critical part of the virtual assistant planning process. These needs will align with the intents but may not match one-to-one. Beware of biasing yourself when you create the intents. Try to find the intents present in the data, rather than force-fitting the data into the intents.

Table 5 shows a sample of the utterances FICTITIOUS INC collected. Take a few minutes to review the utterances and to organize them into intents.

Table 5 Example utterances collected from FICTITIOUS INC mock user interface

reset my password	i'm locked out of my account
my account is locked	close my account
where are you located	sign up for fictitious inc rewards
what time are you open	directions to fictitious inc
store hours	i need a job
when do you close tonight	can i have a password reset
i need to open an account	i need your schedule
how many points do i have left	do you have a store downtown?
can't login	is this thing on?
i forgot my password and need to reset it	

Please note there are no perfect answers. Every time I do an intent mapping exercise like this I am amazed at the variations and alternatives that people come up with. I've included my mapping of these utterances below in Figure 9. Please don't peek until you've at least thought about the intents!

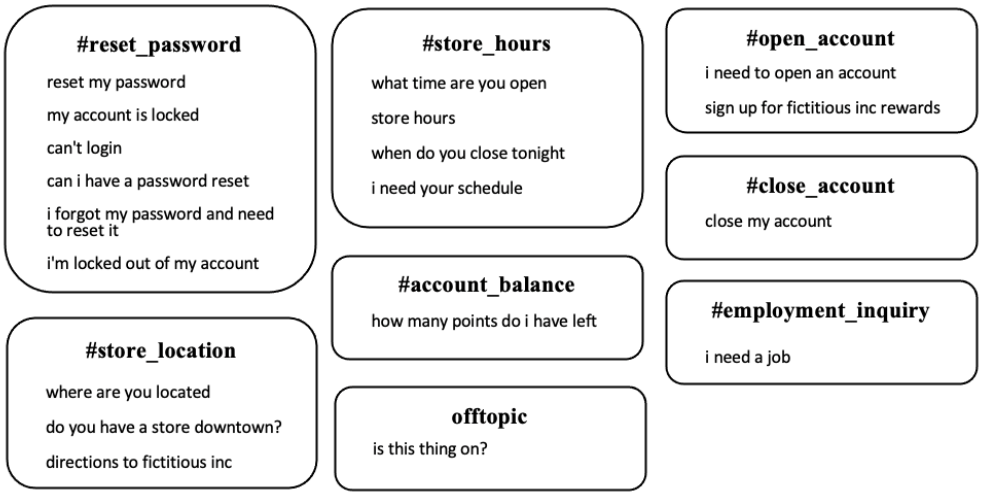


Figure 9 One possible organization of the utterances from Table 5.

Compare this organization to your own. What differences did you see? Here's a couple differences I've gather from performing this exercise with various audiences:

- **Reset password vs locked account vs login problems**: It's reasonable to define these as separate intents, even if they share an answer.
- **Hours vs open/close times**: Requests for opening and closing times are different requests. I combined them because the most common "hours" requests are ambiguous ("store hours" and "schedule" do not fit cleanly into an open or close intent)

There's no One True Way to organize data into intents. Reasonable people can and will differ on intent definitions. The precise criteria are not so important as the fact that your whole team agrees on the definitions and that these definitions lead to satisfied users. A virtual assistant trained with data in disagreement will not perform well. Whenever your team does not agree on a definition, have them work together until a consensus can be found. Then train your virtual assistant on that consensus data.

While there are no perfect answers in the intent mapping exercise, there are mappings that will work less well than others. The distribution of intents generally follows a power-law distribution. You may also be familiar with this as a Pareto distribution[3] or more simply the "80-20" rule. The core idea as it relates to assistants is that 80% of the utterances fall into 20% of the intents. FICTITIOUS INC's data fits this pattern as demonstrated in Figure 10.

[3] https://en.wikipedia.org/wiki/Pareto_distribution

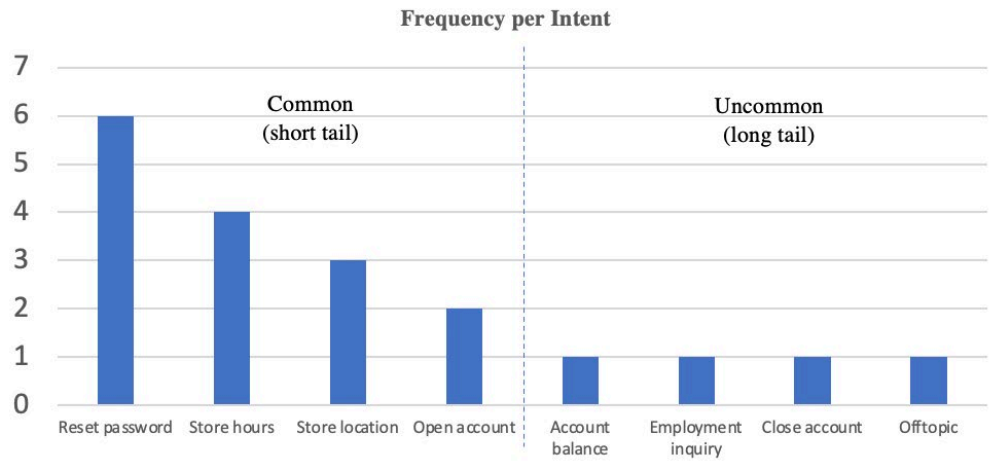


Figure 10 Distribution of the number of examples per intent in an intent mapping exercise. There will be common and uncommon intents. Focus on the most common intents first.

In some literature the common intents are called the "short tail" and the remaining intents are the "long tail". The sweet spot for virtual assistant value is to focus on the short tail questions. In the example diagram, the short tail covers somewhere between the first six and eight intents. Focus on training these common intents first.

Exercise

Take the virtual assistant from Chapter 2, export the training data, and delete all of the intent labels. Then shuffle the data and organize it into intents. If you are pressed for time, feel free to use just a subset of the data.

When you are done, compare your intents to the original intents. The odds are that you'll have both some different intents as well as some different utterances within the intents.

The larger your dataset, the more likely you are to have differences of opinion when labeling the data with intents.

6.3 Assessing if you have the right training data

The quality of the training data sets an upper limit on the accuracy of a virtual assistant. FICTITIOUS INC should inspect the quality of their training data before training their assistant. There are three criteria training data must meet to produce optimum results. I refer to them as the three V's:

- Variety
- Volume
- Veracity

The training data is a "ground truth" for the model and must follow all of these characteristics for the model to be successful. If your training data is missing one or more of these three V's, your assistant will not be as accurate as it could be. These inaccuracies will lead to a poor experience for your users and increased costs to your organization.

Let's assess FICTITIOUS INC's training data against each of the V's in turn, starting with Variety.

6.3.1 Training data variety

An assistant should be trained with data that has variety *representative sample* of what the assistant will see in production. FICTITIOUS INC has a wide demographic range of customers, each of whom may phrase their requests differently. This variety should be captured in FICTITIOUS INC's training data. Otherwise, their assistant may not recognize common phrasing patterns, which will lead to bad outcomes.

A famous example of a non-representative sample with negative consequences

If a sample is biased – *not* representative of the whole – it can have disastrous consequences. One famous example is the 1948 US Presidential election. The Chicago Daily Tribune's polls predicted a decisive victory for the Republican challenger Thomas Dewey. Instead, incumbent Harry Truman won in a landslide. A retrospective analysis showed that the Tribune's polls were done entirely over the phone - phones were not present in every household yet and were more often in wealthy, Republican households[4].

There is an extensive literature on how to avoid sampling bias. Suffice it to say you should carefully inspect your sampling process for any potential biases. A good sample has a similar distribution of characteristics as the broader population. A truly random sampling can achieve that. As demonstrated in the Tribune example "random" and "convenient" are two different things. The Tribune team surely thought they were doing a random sampling!

Let's compare and contrast the variety from two different data collection strategies. To keep it simple, we'll focus on a single intent: password resets. In the first strategy, FICTITIOUS INC will skimp on data collection time and just ask their SMEs how users ask for password resets. We'll call this the "fabricated" data set. In the second strategy, FICTITIOUS INC will collect actual user utterances from production logs and a mock user interface. We'll call this the "representative" data set. These two data sets are shown in Table 6.

Table 6 FICTITIOUS INC password reset data sets from two different strategies: Fabricate (or infer) what users *might* say, and using actual and representative things users *do* say

Fabricated data set (Strategy 1)	Representative data set (Strategy 2)
I need to reset my password	I'm locked out of my account
I want to reset my password	I can't remember my login info
Password reset	I have a pw issue
How do I reset my password	I forgot my password

[4] https://medium.com/@ODSC/dewey-defeats-truman-how-sampling-bias-can-ruin-your-model-f4f67989709e

Help me reset my password Need help resetting my password I forgot my password	Reset password Can't get past the login screen

There is some nominal variety in the fabricated examples, but they are largely identical from the classifier's point of view. Every statement includes "my password" and almost all include "reset my password". There is almost no variety in the verbs. (Did you notice that the SME was biased by the wording in our question?)

In contrast, the representative data set has rich variety. It has twice as many unique words (23 vs 13) and each example uses a different verb. What if FICTITIOUS INC trained two different virtual assistants – one the fabricated data set and one on the representative data set. Which do you think would perform better?

Try it out! There's no need to trust your intuition on this one.

I performed this experiment and will show the results below. I created two copies of an existing assistant added a `#reset_password` intent[5]. I performed two different experiments of the new `#reset_password` intent using the different FICTITIOUS INC data sets:

- Experiment 1: Train on fabricated user utterances, test on real user utterances
- Experiment 2: Train on representative utterances, test on fabricated utterances

The experiment results are summarized in Table 7.

Table 7 Summary of experiments

Experiment	Training Data	Test Data	Accuracy on Test Data	Average Confidence on Test Data
Experiment 1	7 Fabricated utterances	6 Varied & Representative utterances	33%	0.39
Experiment 2	6 Varied & Representative utterances	7 Fabricated utterances[6]	100%	0.91

Table 7 shows the power of variety in training data and underscores the need for a good data collection exercise. If FICTITIOUS INC had gone to production with only their fabricated `#reset_password` utterances, they would miss almost two-thirds of user password reset requests! This would have a disastrous effect on their success metrics! The assistant with FICTITIOUS INC's fabricated data performs so poorly that it's worth exploring in detail as shown in Table 8.

[5] Full experiment setup: I created a new skill in Watson Assistant, imported the Banking intents from the Content Catalog, and tested using the version parameter value '2020-04-01'. I added the training examples one-by-one in the order they are listed in the data table. Other virtual assistant platforms and classifiers learn at different rates.

[6] This test is a little unfair – it's easy to correctly predict the set of fabricated utterances, precisely because they have so little variety.

Table 8 Experiment 1 results: Train an assistant with fabricated training and test on actual, representative training data.

Test Utterance	Top Intent	Top Intent Confidence	#reset_password Confidence
I'm locked out of my account	other	0.37	n/a
I can't remember my login info	#reset_password[7]	0.13	0.13
I have a pw issue	other	0.17	0.14
I forgot my password	#reset_password	1.00	1.00
Reset password	#reset_password	0.97	0.97
Can't get past the login screen	other	0.09	0.08

Table 8 shows just how little the assistant learns from fabricated data. The utterance "I forgot my password" was classified with perfect confidence because it appeared in both the training and test data. The remaining training phrases included the words "reset" and "password", so it is not surprising that it classified "Reset password" correctly. After those two phrases, the assistant was never even close. Without being trained on varied data, FICTITIOUS INC's assistant has little chance of correctly predicting the intent in varied utterances.

Data collection from real users can be costly but generates excellent data variety!

The training data collection techniques shown in this chapter can be time-consuming. There is a strong temptation among virtual assistant builders to skimp on data collection and have subject matter experts fabricate example data. Many subject matter experts believe they can generate a similar sampling of training data as would be found in a data collection process. These people are well-intentioned but generally wrong. The "distance" between their fabricated sample and the real-life distribution is proportional to the size of the accuracy problem you'll have using their data!

If you cannot find good training data you can start with fabricated data and the insight of your SMEs, but make sure to set aside time in the near future to improve the variety in your training data. Keep track of which training samples are from actual users and which examples are fabricated. Over the life of the virtual assistant, you should remove the fabricated examples and replace them with actual end-user examples.

Variety is clearly important to an accurate assistant. FICTITIOUS INC's assistant worked much better when trained with a variety of data. Let's examine their training data for the next V – volume.

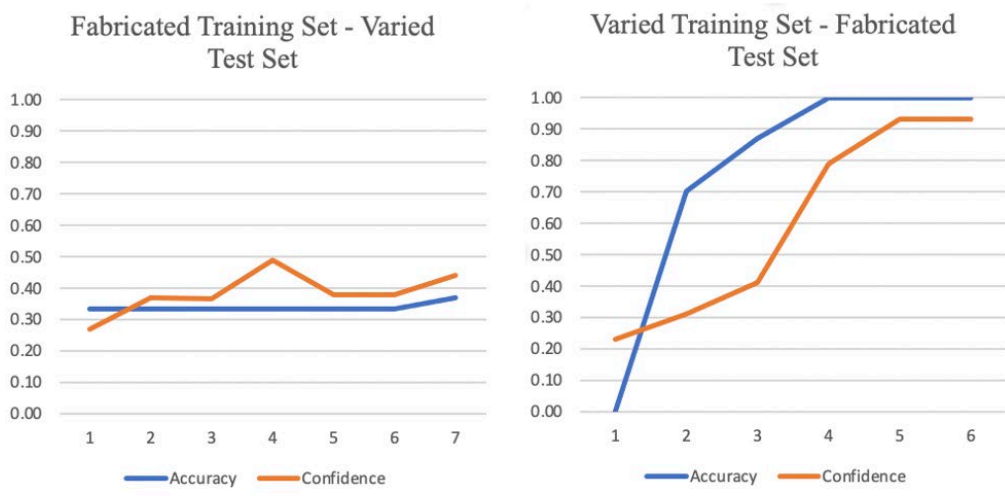
[7] I scored this as a miss. Most virtual assistant platforms discard predictions below a confidence threshold, and 0.13 is a very low confidence.

6.3.2 Training data volume

All classifiers require a volume of training data to train successfully. Given the nearly infinite ways natural language can express an intent (in conversational assistants) or a classification (in event classifiers), we should not expect a classifier to be able to extrapolate to all of them from one or two examples. How many training examples will FICTITIOUS INC need?

Commercial virtual assistant platforms vary in their recommendations, but they generally work best with ten or more examples per intent[8]. The sample and built-in intents from commercial platforms are often trained on many more examples. The sample code FICTITIOUS INC used in Chapter 2 was based on Watson Assistant's Content Catalog, which included twenty examples per intent.

In the previous section we saw how increasing the variety of training data affected the accuracy of FICTITIOUS INC's assistant. We can see how training data volume affects the assistant by running the same experiment in multiple iterations, starting with one piece of training data and adding one additional training in each successive iteration. The training data volume effect is shown in Figure 11.



1. Accuracy and confidence are almost completely unaffected by training data that has volume but not variety.

2. Accuracy and confidence increase, then plateau, when training data increases in volume *and* variety.

Figure 11 FICTITIOUS INC's accuracy on the #reset_password intent, based on the volume of training data. Diagram 1 uses a fabricated training set with little variety. Diagram 2 uses a training set with high variety.

Figure 11 shows that training data volume does not always have a significant impact on accuracy.

[8] Virtual assistant platforms make differing recommendations on the specific data volume you need to train your assistant. Training data volume requirements have steadily decreased as virtual assistant platform builders improve their algorithms.

- In diagram 1, the training data had little variety. Each new piece of training data contributed little variety and thus had little impact on the assistant's performance.
- In diagram 2, the training data had high variety. Each new piece of training data added significant variety and the assistant learned from every example[9].

Figure 11 shows that training data volume is a means to an end. Artificially inflating training data size by adding many similar examples does not help the assistant. FICTITIOUS INC's experiment shows that a small volume of varied examples can outperform a large volume of non-varied examples.

The most important part of training data volume is to get a high enough volume to cover the variety you expect the assistant to encounter and so that patterns can emerge. Training generally improves with more examples (as long as they follow the other two V's: variety and veracity).

Can different intents have different training volumes?

Yes! As shown in this chapter, capturing the right variety is more important than hitting a specific volume target. Some intents will have good variety with fewer examples than other intents.

Still, it's good to have approximate volume balance across your intents, meaning a similar number of examples for each intent. Each training example creates some "gravity" towards that intent. An intent with several times more examples than the average per intent will have too much gravity, causing over-selection of that intent. Similarly, an intent with far fewer examples than average is likely to be under-selected.

An intent with "too many" examples will have a large positive bias and cause a large negative bias in every other intent. The mathematics are outlined in Chapter 11. In short, the number of examples in a given intent influences the positive bias towards that intent. For a selected intent, the number of examples in *every other intent* influences the negative bias towards the selected intent.

We've seen that training data variety and volume are both important drivers of virtual assistant accuracy. Let's look at the final V: veracity.

6.3.3 Training data veracity

Veracity derives from the Latin word "verax", meaning true[10]. You may have heard the phrase "Garbage in, garbage out". This phrase certainly applies to training data! The virtual assistant learns directly from training data. If that data is not true (not correct), then the assistant will not learn the right lessons from the training data.

An important aspect of veracity is that the training data is unambiguous. Every training data sample should match to one intent – no more and no less. The FICTITIOUS INC `#reset_password` utterances were all completely unambiguous in intent. Let's examine

[9] The shape of the accuracy curve in this experiment is affected by the order the examples are added. When the first and only training example was "Reset Password", the fabricated test set was predicted with 100% accuracy and 76% confidence. This underscores how important it is for the training data variety to be similar to the test set variety!

[10] https://www.merriam-webster.com/dictionary/veracity

some ambiguous utterances FICTITIOUS INC found in their production logs. These utterances are listed in Table 9

Table 9 Additional FICTITIOUS INC training utterances, some include ambiguity

utterance
account
Oh I'm so glad to talk to someone, I need to reset my password
I need to reset my password and verify my points balance

Each of these utterances suffers from some ambiguity. Let's see if or how FICTITIOUS INC can use these statements when training their assistant.

AMBIGUITY: NO INTENT MATCHED

If a FICTITIOUS INC user started their conversation with the single word "account", what should the assistant do? The user may ultimately want to open an account, close their account, modify the account, or check their account balance. There's no way to know which of these they wanted based on the utterance alone! The first type of ambiguity is a statement does not match any intent at all. This type of statement cannot be used as a training example for any intent because there is no intent that can be inferred from the statement by itself.

Though FICTITIOUS INC cannot use the utterance "account" as training data, they can create and use variations of that utterance that unambiguously indicate a single intent, as shown in Table 10.

Table 10 How FICTITIOUS INC's assistant should treat short utterances

Ambiguous (handle at runtime)	Unambiguous (use in training)
account	Open account Close my account Account balance

Users will ask ambiguous questions. Ambiguity should be handled at runtime, not during training. At runtime, your assistant can respond to ambiguity by asking a disambiguation question. If the user says "account", the FICTITIOUS INC assistant can disambiguate by asking "What type of account service do you need?". An assistant is most accurate when it is trained on unambiguous examples. This training will help it deal with ambiguity at runtime! You should not train with statements that don't clearly identify an intent and should instead let a disambiguation flow clarify them[11].

[11] This is brilliantly explained in Best Practice #5 of "Best Practices for Building a Natural Language Classifier" by Cari Jacobs https://medium.com/ibm-watson/best-practices-for-building-a-natural-language-classifier-part-two-286588014752

Do virtual assistant users really use one-word utterances?

Yes! This is surprisingly common. Users are familiar with keyword-based automated telephone systems, where a single word may advance you to the next menu option. In text-based virtual assistants, some users treat the dialog input like a search query box. Single-word utterances are often ambiguous.

AMBIGUITY: ONE INTENT MATCHED (LOOSELY)

The second ambiguous example was "Oh I'm so glad to talk to someone, I need to reset my password". At first, this seems like an unambiguous password reset request. But this statement has a lot of "filler" that does not contribute to the intent. It would be just as common to see an utterance "Oh I'm so glad to talk to someone, I need to check my points balance" or "Oh I'm so glad to talk to someone, I need to close my account". The opening "Oh I'm so glad to talk to someone" provides no contextual information to the assistant.

Filler adds ambiguity and should be omitted from training data. The classifier in the assistant learns from every single word in the training data, so precision is important.

Sources of filler

You are especially likely to find extraneous content in transcripts from human-to-human interactions. Humans are social creatures - we just can't help ourselves. When training a conversational assistant, remove filler from the training data.

FICTITIOUS INC can remove the filler before using these utterances to train their assistant. The modified ambiguous and unambiguous statements are shown in Table 11.

Table 11 How FICTITIOUS INC's assistant should treat verbose utterances

Ambiguous (handle at runtime)	Unambiguous (use in training)
Oh I'm so glad to talk to someone, I need to reset my password Oh I'm so glad to talk to someone, I need to check my points balance	I need to reset my password[12] I need to check my points balance

AMBIGUITY: MULTIPLE INTENT MATCHED

The utterance "I need to reset my password and verify my points balance" demonstrated the last form of ambiguity: when an utterance truly matches multiple intents. This user utterance describes two intents: `#reset_password` and `#check_balance`. Users may ask your virtual assistant such compound questions, but the training phase is generally not the correct place to handle them.

[12] Arguably, the segment "I need to" has some ambiguity, as it could be part of an utterance in multiple intents. "I need to" or even "I need" could be removed from the training examples, as long as you are consistent and remove it from every intent. I prefer to keep that phrase since different intents can have different levels of urgency

My virtual assistant platform lets me train on compound questions. Why shouldn't I?

Some virtual assistant platforms allow you to train compound classes like `#reset_password_and_check_balance`, however this is generally not a scalable solution.

The first problem is that the new compound classes are going to be similar to each other as well as the component classes.

The second problem is that the number of possible compound intents grows exponentially with the number of component intents. Providing enough training data for compound intents is generally a non-starter.

The best way to use a compound utterance in your training data is to split it up into two component utterances. The utterance "I need to reset my password and verify my points balance" can become "I need to reset my password" and "I need to verify my points balance", each of which is an unambiguous example for their respective intent. These new unambiguous examples are suitable for training the assistant.

Table 12 Compound statements are too ambiguous for training. Break them up into unambiguous training examples.

Ambiguous (handle at runtime)	Unambiguous (use in training)
I need to reset my password and verify my points balance	I need to reset my password I need to verify my points balance

VERACITY SUMMARY

The training data for a virtual assistant should always have veracity. Every piece of training data should unambiguously match one intent. Unambiguous training data provides clear patterns for your assistant to learn from. Users may ask ambiguous questions to your assistant, but the correct place to handle ambiguity is at runtime. You can always deal with an ambiguous user utterance by asking a follow-up clarifying question.

Veracity rule of thumb

If you cannot identify the right intent for an utterance in less than fifteen seconds, that utterance is too ambiguous for use as training data!

6.4 Summary

- An assistant is trained by example – not programmed. The assistant is shown utterances that match each intent and then infers patterns from that data during the training process.
- Training data should be collected from a source as close as possible to the virtual assistant. The best data source is production logs which provide actual and representative user phrasing, the least accurate source is expert intuition.
- Training data must include a variety representative of actual user data. Getting the right variety of data is more important than creating a large volume of training data.

- Training data must be unambiguously correct.

7

How accurate is your assistant?

This chapter covers:

- Collecting test data for your assistant
- Assessing the accuracy of your assistant
- Selecting the best accuracy metric(s) to use for your assistant

Virtual assistants make predictions based on the way they are trained. How can you tell if this training is working well? You shouldn't release a virtual assistant into the wild if you don't know how well it works! You need to be able to tell if you are making the assistant smarter or dumber when you change the way you train it.

FICTITIOUS INC wants to assess their virtual assistant's accuracy will be when they put it into production. The best way to test a virtual assistant's accuracy is to see how well it predicts intents in production. This poses an interesting conundrum. They don't want to go to production without reasonable accuracy, but they won't know the assistant's true accuracy until it is in production.

The best way to handle this conundrum is to train and test your assistant iteratively, as shown in Figure 1. The virtuous cycle of Gather Data, Train, Test, and Improve, provides a repeatable methodology to build and improve your virtual assistant. It's ok for FICTITIOUS INC to start with imperfect data because they can continually improve it. You have to start somewhere!

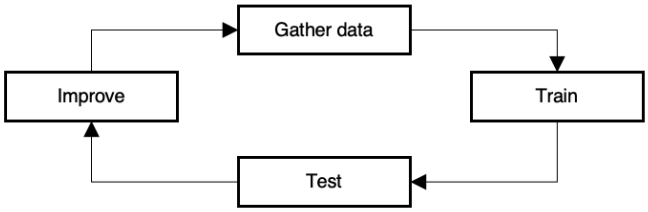


Figure 1 The virtuous cycle of testing how accurate your assistant is.

In Chapter 6 we covered how to gather data and train your assistant. In this chapter, we will walk through how to test your assistant's accuracy. An accuracy test should not just tell you how accurate your assistant is but where and why it is inaccurate, including showing any error patterns. This chapter gives you the tools to assess your assistant's accuracy so that you understand its strengths and weaknesses. This information helps you identify focal points for improvement.

7.1 Testing a virtual assistant for accuracy

A virtual assistant session starts with user input. The virtual assistant first classifies that input then decides the appropriate response action, and finally responds to the input. The user may respond to the system, starting the cycle anew. Figure 2 shows how execution flows in a virtual assistant.

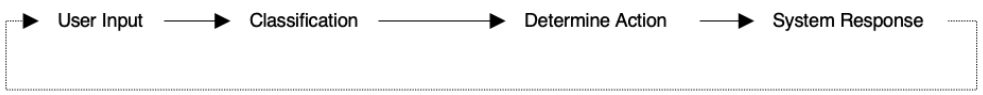


Figure 2 Execution flow in a virtual assistant

Figure 3 depicts an annotated FICTITIOUS INC conversation. A user may say "I'm locked out of my account" and the assistant will classify this utterance to the `#reset_password` intent. The assistant uses the intent to select a dialog response. The entire conversation cycle needs to be tested. In this chapter, we will focus entirely on testing the classification process so that we can determine how accurate the assistant is.

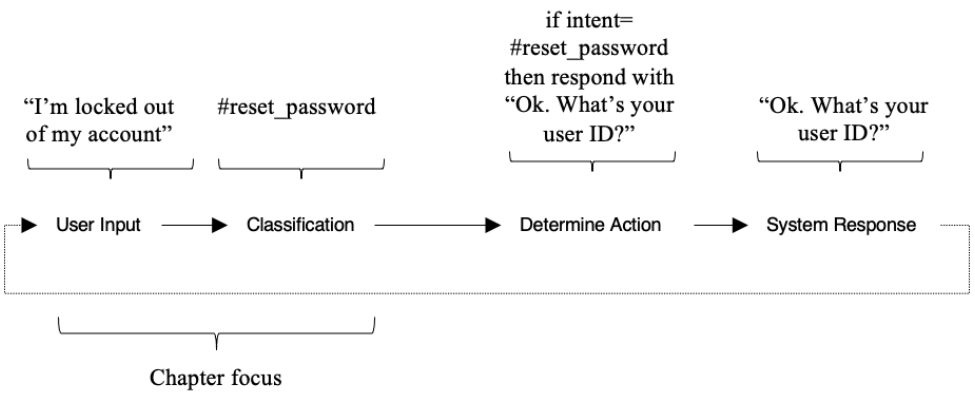


Figure 3 In this chapter we will focus on testing the classification process. This tells us how accurate the assistant is.

FICTITIOUS INC wants to measure how accurate their assistant will be when they deploy it to production. Let's first define accuracy. The simplest definition of accuracy is the ratio of correct predictions to total predictions:

$$\text{Accuracy} = \text{correct predictions} / \text{total prediction}$$

When FICTITIOUS INC's assistant classifies the utterance "I'm locked out of my account" as a `#reset_password` intent, it is correct. When the assistant classifies the utterance "Where can I apply for a job?" as the `#store_location` intent, it is incorrect.

Accuracy is literally an equation of "right" and "wrong". FICTITIOUS INC's assistant is right when it predicts the correct intent for a user utterance and is wrong when it predicts an incorrect intent for a user utterance.

FICTITIOUS INC wants a detailed picture of their assistant's accuracy. They want to know what parts of their assistant work well and what parts do not work well. They assume that some intents will be more accurate than others. They want to be able to find intents with major accuracy problems so they can fix them. They also want specific insights into any accuracy problems they find.

Let's explore how to test the accuracy of FICTITIOUS INC's assistant.

7.1.1 Testing a single utterance

FICTITIOUS INC can test the accuracy of their assistant one utterance at a time. Some virtual assistant providers let you access the classifier independently of the dialog engine, and some do not. The classifier can be tested independently of the rest of the assistant, even if the classification is coupled to the dialog, by looking only at the classification and

ignoring the dialog[1]. In Figure 4 FICTITIOUS INC uses their virtual assistant testing interface to see how the assistant classifies the utterance "What time are you open".

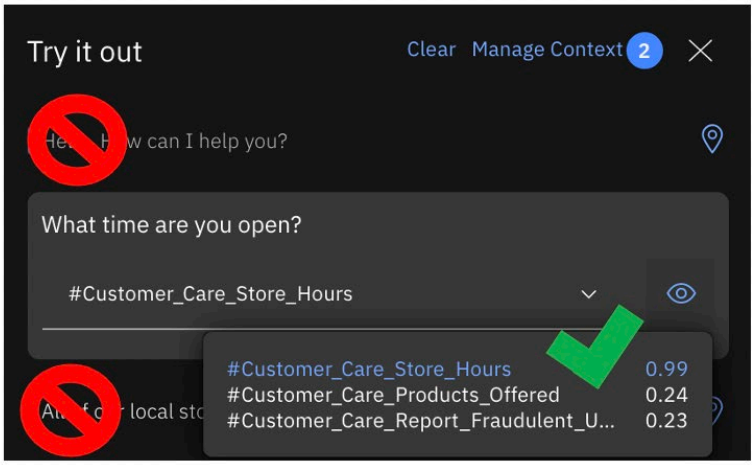


Figure 4 If your virtual assistant couples the classification and dialog response, you can still manually test the classifier by ignoring the dialog response.

FICTITIOUS INC can test their classifier manually in a testing interface, testing one utterance at a time. This is fine for quick tests, but FICTITIOUS INC can expect to test thousands of utterances over the life of their assistant. They will want to automate these tests by using an API.

Every virtual assistant platform I surveyed for this book exposes an API for testing the assistant's classifier. The API will require an utterance for input and will provide an intent as output. Depending on your virtual assistant platform, you may need to provide additional configuration parameters such as username, password, version, or other identifiers. Code Listing 1 demonstrates an exemplary API call.

Code Listing 1. Using an API to test the classification of "What time are you open?"

```
curl -X POST -u "{your_username}:{your_password}" --header "Content-Type:application/json"
       --data "{\"input\": {\"text\": \"What time are you open?\"}}"
       "{your_url}/message?version=2020-04-01"
```

The input and output from this API call are shown in Figure 5.

[1]Some of the major platforms bundle these two functions together, especially those with a "low-code" model. Some of the platforms keep these as separate functions. Your platform's API documentation will tell you whether these functions are bundled or separate.

API Input
Provide the utterance

```
{
   "input": {
      "text": "What time are you open?"
   }
}
```

API Output
Extract the intent

```
{
   "input": {
      "text": "What time are you open?"
   },
   "intents": [
      {
         "intent": "Customer_Care_Store_Hours",
         "confidence": 0.9874794483184814
      }
   ],
   "entities": [],
   "output": {
      "text": [
         "All of our local stores are open from 9am to 9pm."
      ]
   }
}
```

Figure 5 Using a virtual assistant's API to test a classifier one utterance at a time

Your virtual assistant platform may provide a Software Development Kit (SDK) which makes it easier to interact with their classification APIs. If they don't provide an SDK, you can write your own scripts to interact with these APIs. Having code that calls the classification API is the first step to automation.

The second step in automating classification tests is calling the API in a repeatable fashion. FICTITIOUS INC created a shell script `classify.sh` to simply making a single classification call. The contents of this script are highly dependent on their virtual assistant platform, but the script should do the following:

1. Take two input parameters: an utterance and its expected intent
2. Initialize a connection to the virtual assistant
3. Call the assistant's classification API to classify the utterance
4. Extract the predicted intent and compare the predicted intent to the expected intent
5. Output four parameters: the utterance, the expected intent, the predicted intent, and the correctness of the predicted intent.

The desired output from this script is shown in Table 1.

Table 1 Example output format for an automated classification test

Utterance	Actual Intent	Predicted Intent	Prediction Assessment
What time do you open?	`#store_hours`	`#store_hours`	Correct

The four pieces of information in Table 1 form the basis of all accuracy measurements for the virtual assistant.

FICTITIOUS INC can get some specific insights testing one utterance at a time. This is a great stepping stone to achieving their accuracy testing goals. Testing one utterance at a time is not a scalable solution, but it helps to automate accuracy testing.

7.1.2 Testing multiple utterances

FICTITIOUS INC wants to find patterns of errors. To find patterns of errors they need to test many utterances at once. Now that they have automated a process to test a single utterance, they can expand that process to test multiple utterances. FICTITIOUS INC can revise their `classify.sh` script to test multiple utterances at a time. They are essentially inserting a `for` loop into the script. The pseudocode of this script has the following steps:

1. Take one input: a comma-separated values (CSV) file with two columns, each row containing an utterance and its expected intent
2. Initialize a connection to the virtual assistant
3. For every row in the CSV file:
4. Call the assistant's classification API to classify the utterance
5. Extract the predicted intent and compare the predicted intent to the expected intent
6. Output four parameters: the utterance, the expected intent, the predicted intent, and the correctness of the predicted intent.

With this new script, FICTITIOUS INC can run an accuracy test on an entire spreadsheet's worth of data at once. For their first multiple utterance test, they create an input file with four rows, each row with an utterance and its intent, and pass this file to this new script. The output from the script is listed in Table 2.

Table 2 Testing four utterances in a single test with FICTITIOUS INC's automated script.

Utterance	Actual Intent	Predicted Intent	Prediction Assessment
I'm locked out of my account	`#reset_password`	`#reset_password`	Correct
Where is the Elm store?	`#store_location`	`#store_location`	Correct
Can I get a job?	`#employment_inquiry`	`#employment_inquiry`	Correct
Where can I apply for a job?	`#employment_inquiry`	`#store_location`	Incorrect

The ability to test a batch of utterances at once is more useful for assessing the assistant's accuracy than testing one utterance at a time. By reading this tabular report FICTITIOUS INC can extract the following insights:

- Their assistant was 75% accurate in this small test.
- Their assistant never made a mistake on `#reset_password`.
- When the correct answer was `#store_location`, the assistant did not make any mistakes.
- When the correct answer was `#employment_inquiry`, the assistant seemed *confused,* sometimes correctly selecting `#employment_inquiry` and sometimes incorrectly selecting `#store_location`.

Perhaps the most useful insight from this data is the potential confusion between `#employment_inquiry` and `#store_location`. For virtual assistants, *confusion* is defined as a pattern of errors between a group of two or more intents. Knowing that the assistant is 75% accurate in this test is interesting but knowing the source of the errors is actionable. When FICTITIOUS INC improves their assistant, they need to address the biggest sources of confusion first.

Confusion is when an assistant makes frequent errors between a group of two or more intents.

By carefully reading the result data, FICTITIOUS INC can extract useful insights including potential confusion. This gets increasingly difficult as the test gets larger. More test samples will add more rows to the table. The number of potential confusions explodes as more intents are added. Since any pair of intents can be confused, the number of potential confusions is proportional to the square of the number of intents. Fortunately, there is a method for visualizing confusion that is even easier to read than a table. FICTITIOUS INC can use a *confusion matrix*.

A confusion matrix is a grid with the following characteristics:

- The first dimension lists the correct intents
- The second dimension lists the predicted intents
- Each square in the grid is the number of times the correct and predicted intent combination occurred
- Any number off of the main diagonal represents confusion.

Optionally, the squares are color-coded, such that the opacity matches the relative frequency of that occurrence. This makes the sources of confusion easier to spot. See FICTITIOUS INC's first confusion matrix in Figure 6.

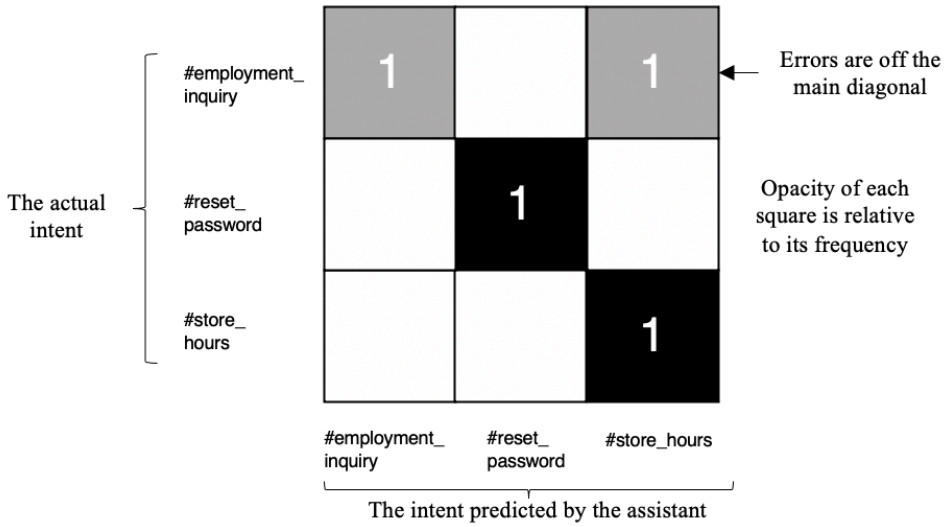


Figure 6 A confusion matrix visualizes accuracy testing results.

With the confusion matrix, FICTITIOUS INC's insights jump right off the page! The matrix is easily scannable. Most of the color opacity is on the main diagonal and it's easy to find the intent pair with confusion.

FICTITIOUS INC can now test multiple utterances, create a table of the test results, and visualize the results with a confusion matrix. These tools are foundational in FICTITIOUS INC's quest to assess the accuracy of their assistant. This first test data set only included four utterances, not nearly enough to give a reliable estimate of the assistant's accuracy. FICTITIOUS INC needs to build a good test data set so they can understand how accurate their assistant will be when they deploy to production.

7.1.3 Selecting a test data set

The most useful data set for testing a virtual assistant is data from that assistant in production. With production data, you can extract a set of data that has both volume and variety representative of what the assistant receives.

However, FICTITIOUS INC has not yet gone to production. They can only simulate a production test using data they *think* will be representative of production data. The accuracy from this test will be as predictive of their eventual production performance as their synthetic data is reflective of the eventual production data. Collecting good test data is just as important as collecting good training data.

The true test of classifier accuracy comes from how well it classifies data *it was not trained on*. This means that a classifier actually requires two data sets: a training data set and a test data set. Fortunately, these data sets can both be drawn from the same source and using the same methodology.

Can the test data overlap the training data?

Some overlap in the two data sets is ok, but it is best if you can minimize the overlap. One of the most important outcomes from a blind test is understanding how well it classifies data it was NOT trained on. (We can generally assume it will classify training data well.)

One notable exception to this is voice-driven conversational assistants. In these assistants, many user utterances are short (possibly only one word), because users are frequently terse with automated voice assistants. Such an assistant is likely to include the most common short phrases into both training and test sets.

FICTITIOUS INC's first test only used four utterances, of which the assistant predicted three correct intents. Can FICTITIOUS INC assume that their assistant is 75% accurate? Probably not. FICTITIOUS INC has eighteen intents in their assistant and most were not included in the test! With only four pieces of test data, FICTITIOUS INC cannot make strong assumptions about their assistant's accuracy.

In production, FICTITIOUS INC's assistant might be 95% accurate or 20%. Because of their small test set size, they cannot rule out either. The more representative test data they add, the more likely the accuracy score is to reflect actual production accuracy. At the very least, they need to add test data for each of their intents. Figure 7 demonstrates the "trustworthiness" a test's accuracy score has based on a volume of representative data.

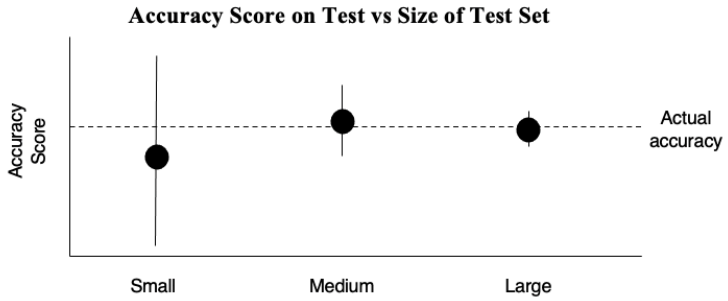


Figure 7 Assuming representative variety in the test set, the larger the test set, the more precise the accuracy score is. This plot shows accuracy scores for three different FICTITIOUS INC test sets and a confidence interval for each score.

FICTITIOUS INC wants to measure how accurate their assistant *will be* when they go to production, which means they need to test on data that they *will see* in production. Having enough test data volume is critical. Figure 7 showed us how test data volume affected the trustworthiness of the test score. Since test data can be gathered the same way as training data, we can gather one set of data and divide it into two sets.

In a perfect world, the test data set would be much larger than the training set, or at least equal in size. Assuming the test data set is representative of the data the assistant will encounter in production, the larger the test data set is, the more confident you can feel in the test results. Small test sets are susceptible to sampling bias. Figure 8 demonstrates the tradeoff between relative test set and training set sizes.

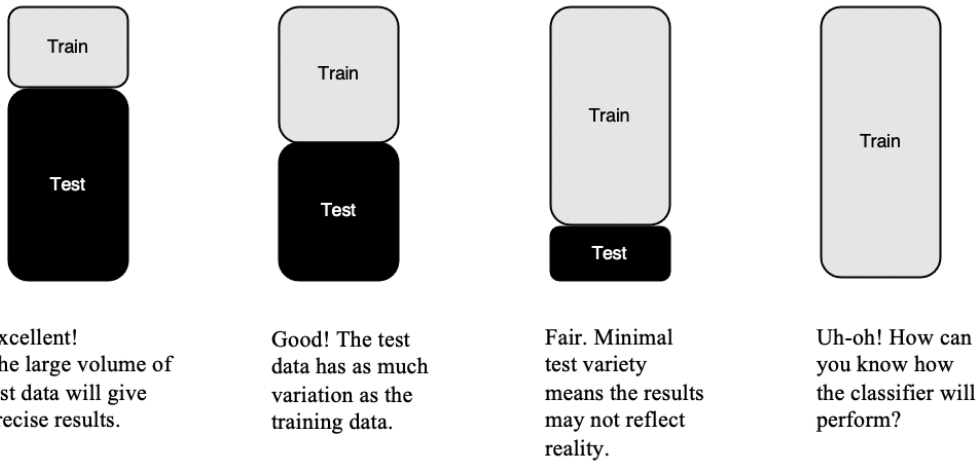


Figure 8 The more test data you have, the less likely the test data set is susceptible to sampling bias, and the more you can trust the results from an accuracy test. You can create a test set by "holding out" examples from the training set and moving them to a test set.

However, most virtual assistant builders struggle to find enough data for the training set alone. A good heuristic for training data volume is to have ten to twelve examples per intent. FICTITIOUS INC's assistant has eighteen intents which thus requires 180 training data samples. An equivalent test data set would have another 180 data samples, bringing the total to 360 data samples (twenty per intent). Ideally, none of these examples should be fabricated – they should all be collected from users or production logs. Most initial data collection exercises don't achieve this volume. Twenty representative, non-fabricated examples per intent is a lot when creating a new conversational assistant.

In data we trust

Data is king when testing your assistant. The quality of your test data set directly influences how much you can trust your accuracy measurement.

FICTITIOUS INC can only trust the accuracy score from an accuracy test if the test data is representative of the eventual production data. In Figure 9 I've plotted several examples of `#reset_password` utterances, organized by similarity. If FICTITIOUS INC runs an accuracy test based on examples in Box 1, the results from the test will not be informative. The

examples in Box 1 are all clumped together, with minimal variety, and do not represent production variety. In contrast, the examples in Box 2 capture a wide range of variety. Box 2 is representative of all the ways users will ask for their password to be reset. An accuracy test based on data in Box 2 will be much more useful than a test based on Box 1 data because it will more precisely reflect the assistant's true accuracy.

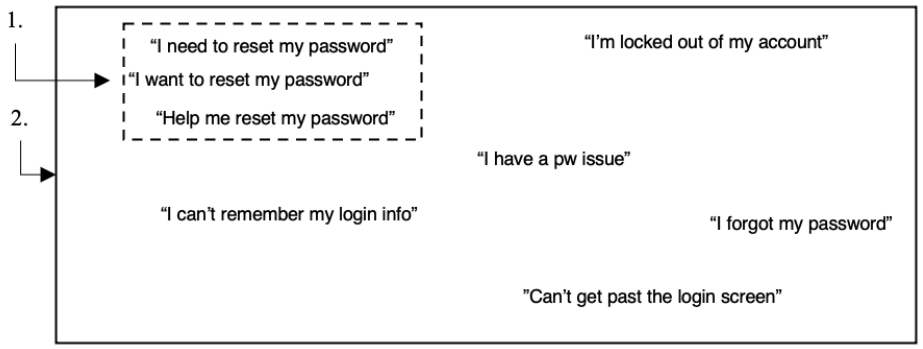


Figure 9 A synthetic test set (Box 1) may not capture the full range of variety from users (Box 2). Accuracy scores based on non-representative variety will be misleading!

FICTITIOUS INC can start with an imperfect test set and expand it over time. The first time they build a virtual assistant they may not have a large test set when they first try to assess the assistant's accuracy. Even if they are not able to build a dedicated test set right away, they will still be able to test their virtual assistant for patterns of confusion. Let's explore the testing methodologies they can use.

7.2 Testing methodologies

FICTITIOUS INC has two testing goals:

1. Estimate how accurate their assistant *will be* in production, as precisely as they can.
2. Find the most common sources of confusion in their assistant, showing them the most common patterns of errors.

FICTITIOUS INC has two different test methodologies they can use, depending on their volume of test data.

- **Blind test:** "Hold out" a subset of the available data for use as a test set. The assistant is never trained with the test set data; the test data is only used for testing. Accuracy comes from how many correct predictions are made on the test set.
- **k-folds cross-validation test**: Have no dedicated test data set, have only the training data set. Train *k* different "temporary" assistants, and for each one "hold out" a different subset of data for a test set. Accuracy comes from averaging the results from each of the multiple classifiers.

Blind test results are predictive of production accuracy and will show where the assistant is truly confused. A k-folds test can only tell where an assistant is *probably* confused and cannot be used to predict production accuracy. The simplest decision of which test to use is based on the volume of available test data, depicted in Figure 10.

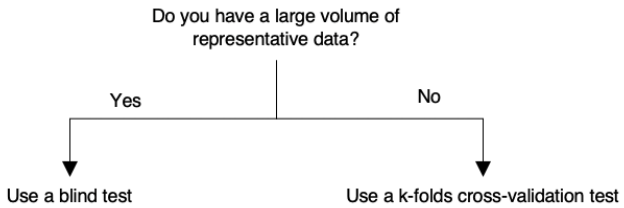


Figure 10 The volume of representative test data you have helps you select the test methodology to use.

The best case is that FICTITIOUS INC has enough testing data to run a blind test. With the blind test, they can achieve both of their testing goals. If they do not have enough data, they can use the k-folds test as their backup plan. Its ability to find probable confusion sources is a useful step in FICTITIOUS INC's accuracy evaluation process.

Let's explore these testing methods in turn, starting with blind testing.

7.2.1 Blind testing

Why is it called a "blind" test? The "blind test" uses "blind data". "Blind data" means that the classifier has not seen that data during the assistant's training. We can generally assume that the volume of training data is significantly smaller than the number of input variations the assistant will see in production. From this, we can infer that for most of the data the assistant encounters will not have been included in training. A blind test is valuable specifically because it shows how well the assistant predicts data it has not been explicitly trained on.

The true test of classifier accuracy comes from how well it classifies data *it was not trained on*.

A **blind test** follows a few simple steps for each utterance in the test data:

1. Get the classifier's predicted intent for the utterance
2. Get the actual intent for the utterance (sometimes called the "golden" intent)
3. Compare the predicted vs actual intent and score the assistant

FICTITIOUS INC's first time training a new intent was when they added a `#reset_password` intent to an assistant populated with several intents from a sample data catalog. The

assistant was accurate at predicting each of the sample intents, which let FICTITIOUS INC isolate the effect of training a single new intent.

FICTITIOUS INC first created their `#reset_password` intent based on synthetic data. They asked themselves how users might ask to have their passwords reset and recorded several example statements that they used in a training data set. A little later they found some example `#reset_password` utterances from other data sources and used these as a test set. The training and test data sets for `#reset_password` are available in Table 3.

Table 3 The data sets used in FICTITIOUS INC's first attempt at the #reset_password intent. FICTITIOUS INC fabricated a training set based on intuition, then tested it against data collected from actual end-users.

Training set (fabricated)	Test set (from production logs)
I need to reset my password I want to reset my password Password reset How do I reset my password Help me reset my password Need help resetting my password I forgot my password	Reset password I'm locked out of my account I can't remember my login info I have a pw issue I forgot my password Can't get past the login screen

FICTITIOUS INC trained their assistant by adding the new `#reset_password` data from the first column in Table 3, then ran a blind test using only the data in Table 3's second column. This test was useful for them to determine the accuracy of their `#reset_password` intent. After the assistant was trained, FICTITIOUS INC tested it on each utterance in the test set, then calculated how many times the correct intent was predicted. In that test only two of the six test data were predicted correctly, yielding a 33.3% accuracy. Table 4 gives details for this first blind test.

Table 4 Blind test results

Utterance	Top Intent	Confidence	Assessment
I'm locked out of my account	other	0.37	Incorrect
I can't remember my login info	`#reset_password`	0.13	Incorrect (confidence is too low)
I have a pw issue	other	0.17	Incorrect
I forgot my password	`#reset_password`	1.00	Correct (was also part of the training data)
Reset password	`#reset_password`	0.97	Correct
Can't get past the login screen	other	0.09	Incorrect

This blind test results give several insights, leading to the conclusion that the `#reset_password` training data variety and volume is insufficient:

- The 33.3% accuracy tells us the assistant is not very accurate at identifying password

reset requests.

- All the utterances that do not include the word "password" are classified incorrectly. This tells us the training data needs more variety.
- The overall low confidence of the predictions (except those containing the word "password") suggest the training data needs more volume.
- Only one utterance that did not overlap with the training data was predicted correctly. This is another signal that the training data volume and variety are insufficient.

This was a very small blind test. The assistant was trained on many intents, but we only tested one of those intents. This helped demonstrate how the training process affected a single intent. For most blind tests we prefer to look at all of the intents at once.

Let's run another blind test for FICTITIOUS INC. For simplicity, let's restrict the assistant to only five intents: `#appointments`, `#employment_inquiry`, `#reset_password`, `#store_hours`, and `#store_location`. This test will use training and test sets of equal size: 50 total training utterances and 50 total test utterances.

Recreate this blind test

You can recreate this blind test by following these steps:

1. Load the Chapter 2 demo code into Watson Assistant
2. Remove all but five of the intents
3. Export the training data into a CSV file
4. Remove the rest of the intents from the assistant
5. Split the CSV file into two equal parts, one for training and one for testing, each with 10 examples of each of the five intents
6. Import the training data back into Watson Assistant
7. Test the assistant using the test data set

This is an ideal test scenario – the training data volume was adequate (10 examples per intent), and the test data was as large as the training data. This suggests that any insight from the blind test will be instructive for how the assistant will perform in production. Since the test data has 50 rows, only the summary metrics from the blind test are shown in Table 5.

Table 5 Accuracy metrics for each of the intents in the blind test

Intent	Accuracy Score[2]
`#appointments`	0.78
`#employment_inquiry`	0.89
`#reset_password`	0.90

[2]The individual intents are technically using an F1 score. For the sake of this exercise, you can think of it as accuracy.

`#store_hours`	0.89
`#store_location`	0.7
`Total`	0.82

The first thing we can see in Table 5 is the overall accuracy - the assistant is 82% accurate in this test. Further, three of the intents are quite high performing with accuracy at 89% or 90%. The overall accuracy is brought down by the performance of the `#store_location` intent with 70% accuracy and the `#appointments` intent with 78% accuracy. FICTITIOUS INC will want to investigate this intent first. The fastest way is to review the confusion matrix from this test, seen in Figure 11.

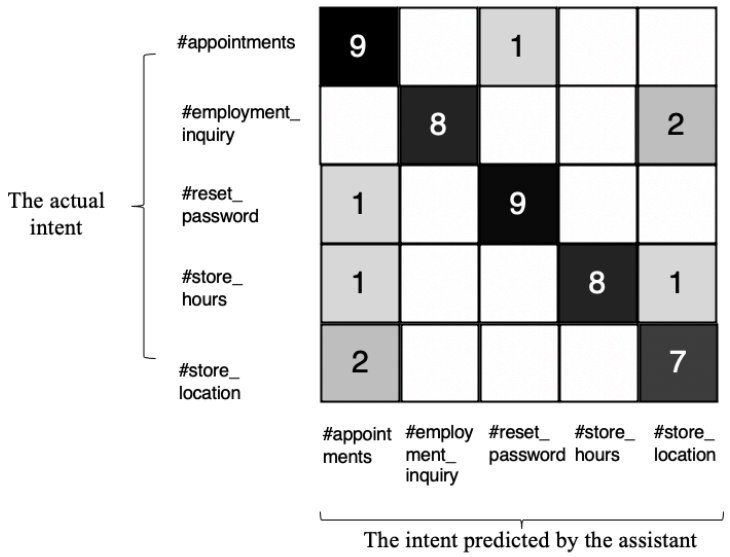


Figure 11 FICTITIOUS INC blind test confusion matrix given five intents with an equal volume of training and test data.

The confusion matrix in Figure 11 immediately surfaces some error patterns. Almost every error by the assistant is found in the `#appointments` and `#store_location` *columns*, meaning the assistant is over-predicting these intents. Four of the errors were when `#appointments` was predicted, and three of the errors were when `#store_location` was predicted. This suggests what went wrong in the test, but we don't have to guess. With Table 6 we can inspect all nine errors from the blind test to search for additional insights.

Table 6 Exact errors made by the FICTITIOUS INC assistant in their blind test

Utterance	Actual intent	Predicted intent
Want to change my visit	#appointments	#reset_password
Where is your careers page?	#employment_inquiry	#store_location
Hire me please	#employment_inquiry	#store_location
I have had the same pin used to contact telephone customer support for a long time and would like to change it. could you please help me to set this up.	#reset_password	#appointments
What are the business hours of the store nearest to me?	#store_hours	#store_location
At what hour can I swing by?	#store_hours	#appointments
I need help with find a store	#store_location	#appointments
Restaurants nearby please	#store_location	(no intent)[3]
I want to know about a store	#store_location	#appointments

FICTITIOUS INC has achieved both of its accuracy testing goals. They know that for these five intents, the assistant will have approximately 82% accuracy. Further, they know that the bulk of their problems come from just two of their intents. Reviewing these two intents is an actionable next step to improve their assistant. These testing goals were only achieved because they had enough data to create a test set, and that the test set was representative of production variety.

Blind test results are only actionable if the test data is representative of production

Any insight from a blind test is actionable when the test data comes from people using the assistant.

Running a blind test on fabricated user utterances may be a waste of time – garbage in, garbage out. It is unlikely that your fabricated user utterances will resemble the distribution of actual utterances in production.

The most valuable outcome from a blind test is understanding how your assistant performs against real user input.

A blind test gives you the best understanding of how accurate your assistant will be, particularly when you use production data in the test set. Since FICTITIOUS INC is not in production, what would they do if they didn't have enough data for a blind test? There's still hope, they can use a k-folds cross-validation test.

7.2.2 k-folds cross-validation test

A **k-folds cross-validation test** is an alternative way to simulate how an assistant may perform in production. If FICTITIOUS INC did not have production data to do a real blind

[3]This error is why the numbers in the confusion matrix only added up to 49. For simplicity, the "confusion" with "no intent" was omitted from the diagram.

test, they could still examine their assistant with a k-folds cross-validation test. As with all simulations, there is a probability that findings will not be representative of the reality the assistant will see in production. K-folds testing is best suited to assessing how "confusing" the assistant's training data is. This estimate of confusion can be used to hypothesize if the assistant will be confused in production as well.

k-folds testing shuffles all of the training data into *k* separate segments. Every piece of training data appears in exactly one of the *k* segments. For each of the *k* folds, a different segment is held out as a test data set, and the other *k*-1 segments are used to train a temporary assistant. In each fold, a blind test is executed on a temporary assistant. The k-folds result is the aggregated result of the *k* blind tests. Figure 12 demonstrates a k-folds test with *k*=5.

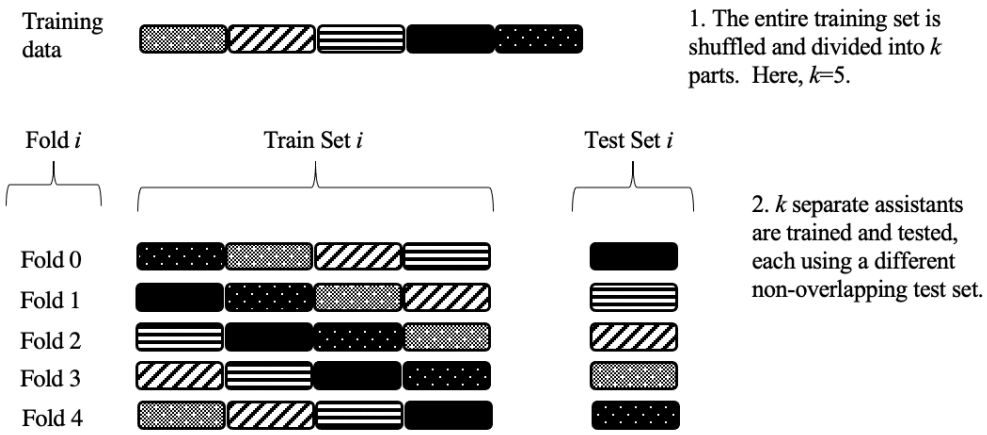


Figure 12 A k-folds cross-validation visualization

Why do they call it k-folds?

The k-folds name comes from two concepts. The first part is *k*, and *k* is just a number. I usually use *k*=3 or *k*=5 or *k*=10. The second part is 'fold', referring to an arrangement of data. When you apply k-folds to your training data you are creating *k* different arrangements of your training data.

If you were really good at origami, maybe you could put your training data on paper and fold it *k* ways!

Let's explore a k-folds test with a small volume of FICTITIOUS INC's data. Table 7 demonstrates a *k*=3 k-folds test with six utterances and two expected intents. In each fold, the assistant is trained on four utterances and tested on two utterances. The k-fold test shows many errors.

Table 7 k-folds test detailed results on a tiny training set

Data #	Utterance	Actual intent	Fold 0	Fold 1	Fold 2	Predicted intent
0	locked out of my account	#reset_password	Test	Train	Train	No intent predicted
1	can you help me reset my password	#reset_password	Test	Train	Train	#employment_inquiry
2	I need to log in and I can't	#reset_password	Train	Test	Train	#employment_inquiry
3	Help me apply for a job	#employment_inquiry	Train	Test	Train	#employment_inquiry
4	I need a job application	#employment_inquiry	Train	Train	Test	#employment_inquiry
5	can I work for you	#employment_inquiry	Train	Train	Test	#reset_password

The total k-folds score in Table 7 is 33.3%, with `#reset_password` scoring at 0. The specific errors all have reasonable explanations:

- In Fold 0, `#reset_password` only had one example to `#employment_inquiry`'s three.
- In Fold 2, `#employment_inquiry` only had one example to `#reset_password`'s three.
- Data 0 has no overlap with any other word in the training data. When Data 0 is blind data, the assistant has no patterns to match against. This prediction is wrong – an intent should have been found but wasn't.
- Data 1 has overlapping words and phrases ("can", "you", "help me") in `#employment_inquiry`, and no overlap to `#reset_password`. Data 2 has the same problem with "I need". These predictions are "confused" – an intent was selected, but it was the wrong intent.
- Data 5 has a similar problem, but in reverse, where "can" and "you" are more indicative of `#reset_password` in this small training set. This prediction is also confused.

Critical thinking exercise

Can you explain why the k-folds test got Data 3 and Data 4 correct?

The k-folds results can also be plotted in a confusion matrix, as in Figure 13.

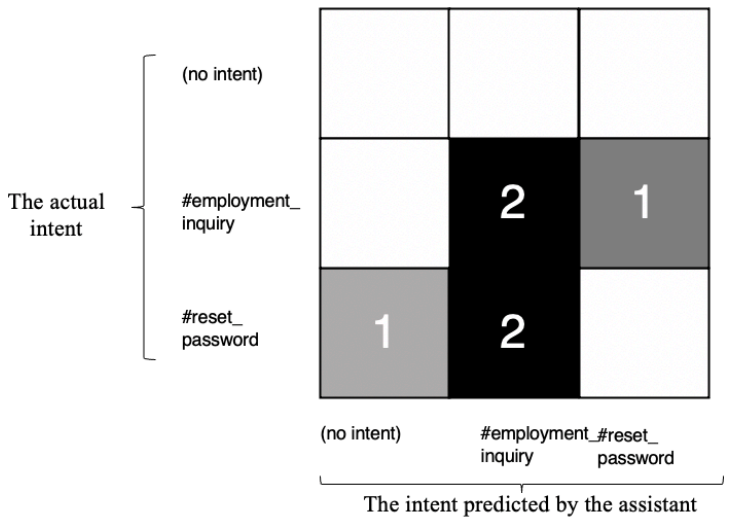


Figure 13 Confusion matrix for FICTITIOUS INC's small k-folds test. The main diagonal is almost empty, suggesting the assistant is highly confused.

This k-folds test shows that there is more confusion between the intents than accurate predictions. (Three predictions were confused, two were correct.) This poor performance is ultimately caused by using such a tiny volume of training data, but let's pretend for a minute we didn't know that. When there is such high confusion between intents, the intents and the training data should be completely reviewed.

Reasoning about individual k-folds results

One interesting challenge with k-folds is that many virtual assistant platforms perfectly remember their training data. For instance, such a platform trained that "unlock my account" means `#reset_password` will not just predict "unlock my account" means `#reset_password`, it will predict `#reset_password` with 100% confidence.

In other words, your k-folds results may show the wrong intent predicted for "unlock my account". The synthetic assistant created by the fold won't have that example. But your real assistant will have that example! Not only will your real assistant predict "unlock my account" correctly, but it may also predict variations of that phrase correctly: for example "unlock my online account", "help me unlock my account", "can my account be unlocked".

This makes reasoning about k-folds results tricky!

First, we should confirm that the intent definitions are not overlapping. If the intent definitions themselves overlap, the solution may be to revise the intent definitions themselves. In FICTITIOUS INC's case, there is no overlap between the concepts of resetting a password and applying for a job.

Secondly, the training data should be evaluated against the three V's. The six-example training data set is evaluated in Table 8.

Table 8 Evaluating small training data set against the three V's.

V	#reset_password	#employment_inquiry
Variety	**Excellent**. Wide variety in the training samples, with little overlap in keywords.	**Good**. Includes multiple verbs. Important words "job" and "apply" are repeated.
Volume	**Poor**. Well below the recommended 10 examples per intent.	**Poor**. Well below the recommended 10 examples per intent.
Veracity	**Excellent**. Each is a very clear example of the intent.	**Excellent**. Each is a very clear example of the intent.

This detailed evaluation of k-folds results is important. If we had looked only at the summary and seen the 0 score for `#reset_password`, we might have tried to "fix" that intent. In fact, we could get a much higher k-folds score by using all of the synthetic and non-varying examples. If every training utterance for `#reset_password` contained the words "reset" and/or "password", the k-folds test would not have made any mistakes on `#reset_password`. Removing the variety to "improve" a k-folds score could hurt the assistant!

k-folds test scores can be artificially inflated by removing variety from your training data. But this will hurt your assistant when it's in production!

With this fundamental understanding of k-folds testing, FICTITIOUS INC is ready to run a larger k-folds test. They will test the same assistant they ran the blind test on, which has five intents and ten examples of each. They will run a `k=5` k-folds test where the data will be sliced into five segments with 10 utterances in each segment. They will have five temporary models, each trained on four of those segments (40 utterances) and tested on the remaining hold-out segment (10 utterances).

FICTITIOUS INC wants to compare the results of this k-folds test to their blind test. In both tests, the assistant had the same training set of 50 utterances. First, they look at the overall test scores in Table 9.

Table 9 Accuracy scores for FICTITIOUS INC data in k-folds and blind tests. k-folds correctly predicted some patterns, like #store_hours being the most accurate intent, but missed other patterns, like #reset_password and #store_location being inaccurate.

Intent	k-folds Score	Blind Score
`#appointments`	0.86	0.78

#employment_inquiry	0.74	0.89
#reset_password	0.78	0.90
#store_hours	1.0	0.89
#store_location	0.6	0.7
Total	0.78	0.82

The tabular results show some interesting findings:

- Both tests suggest that `#store_location` is the least accurate intent.
- Both tests suggest that `#store_hours` is highly accurate.
- The k-folds test suggests that the `#appointments` intent is one of the most accurate intents and the blind test suggests `#appointments` is nearly the least accurate.
- The tests also disagree about the relative accuracy of `#reset_password` and `#employment_inquiry`.

Despite these differences, the overall accuracy score is similar between the two tests (78% vs 82%). It's intriguing to get approximately the same answer for what appears like completely different reasons. FICTITIOUS INC should go to the confusion matrices in Figure 14 to investigate the results more completely.

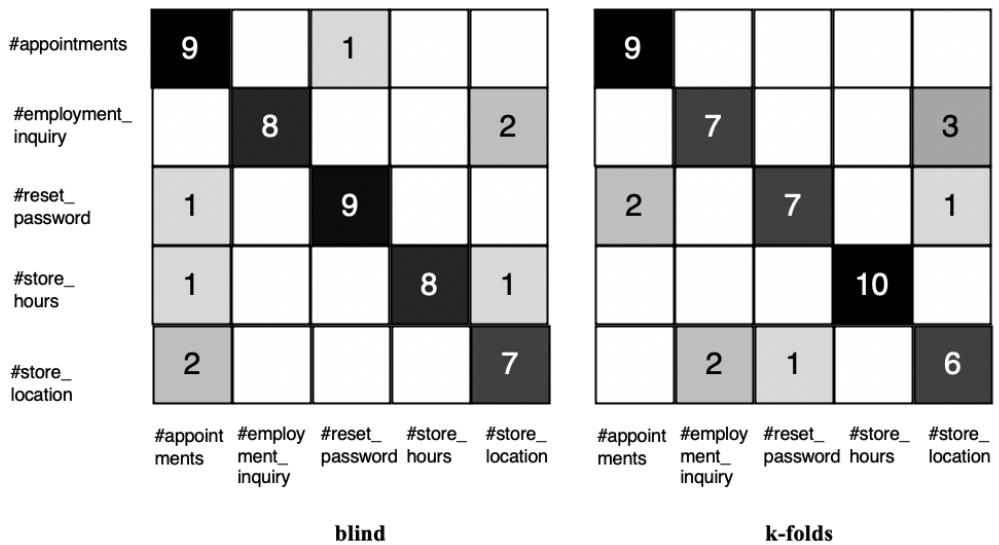


Figure 14 Comparing FICTITIOUS INC blind test and k-folds confusion matrices. The actual intents are on the vertical axis and the predicted intents are on the horizontal access.

Figure 14's confusion matrix shows much more confusion between `#employment_inquiry` and `#store_location`, along with possible confusion between `#reset_password` and `#appointments`. The k-folds test shows four additional error patterns:

- Over-prediction of `#appointments`.
- Over-prediction of `#store_location`.
- Under-prediction of `#employment_inquiry`.
- Under-prediction of `#reset_password`.

The first three error patterns also appeared in the blind test. FICTITIOUS INC can trust these insights. What if FICTITIOUS INC had not run the blind test – could they still trust the fourth insight? Do they really have a problem with the #reset_password intent? They can explore the #reset_password errors in Table 10.

Table 10 Exploration of the #reset_password under-predictions in FICTITIOUS INC's k-folds test

#	Utterance	Actual intent	Predicted intent
1	Wanna change my pass	#reset_password	#appointments
2	Locate pin	#reset_password	#store_location
3	I need to talk to customer service, but I've lost my password. can you help me?	#reset_password	#appointments

From Table 10, the utterances in errors 1 and 2 are unambiguous. The utterance in error 3 is arguably a compound utterance with both `#reset_password` and `#appointments` intents. All three utterances add useful variety to the `#reset_password` training data. Except for the the compound utterance, the `#reset_password` training data looks better than the k-folds results might suggest. FICTITIOUS INC's k-folds score for `#reset_password` was lower than in the blind test precisely because that intent has lots of variety in its training examples (a good thing!).

k-folds testing requires some interpretation and extrapolation. For any given misclassified utterance in the k-folds test, you can't sure if the classifier was wrong because of a real problem, or because the classifier needed that utterance to learn a pattern! k-folds testing can find potential error patterns and sources of confusion, but they may not precisely reflect the true accuracy of the assistant.

Run your own k-folds test!

There are plenty of software libraries that help you run k-folds cross-validation tests.

You can build your own k-folds cross-validation test using the scikit-learn library, documented at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html.

Botium has a general cross-validation framework with adapters for Watson Assistant, Google Dialogflow, Wit.ai, and Amazon Lex. Read more at https://medium.com/@floriantreml/tutorial-benchmark-your-chatbot-on-watson-dialogflow-wit-ai-and-more-92885b4fbd48.

Watson Assistant users can use the open-source WA-Testing-Tool library at https://github.com/cognitive-catalyst/WA-Testing-Tool. This Python tool can run k-folds cross-validation and other tests, generating a variety of charts and diagrams. (The examples in this book were generated with WA-Testing-Tool.)

Note that you may get slightly different results each time you run k-folds. This is because most tools fold the data randomly. The overall scores will different between runs but the error patterns should be stable. Look more closely at the overall patterns than the specific predictions.

Given these caveats, it is important not to optimize for a k-folds score. Always remember that a k-folds test is a synthetic test and cannot replace a blind test. When I review virtual assistants that do not have a blind test set, I am suspicious of k-folds scores greater than 0.9. Such a high k-folds can indicate that the training data has low variety. (If the production data also has low variety, that is fine, but production data usually has high variety!). I'm more comfortable with k-folds scores between 0.70 and 0.89. A score in that range suggests that the training data has some variety while still defining clear intents. A score lower than 0.70 may indicate that many intents are poorly defined, or insufficiently trained.

Do not optimize for a high k-folds score. Use k-folds only to find patterns of errors.

k-folds and blind tests are both useful for assessing the accuracy of your virtual assistant. Both of these tests can provide summary and detailed metrics. I have generally referred to all of these metrics as accuracy. There are other metrics we can use to assess virtual assistant accuracy performance. Let's look at these now.

7.3 Select the right accuracy metric for the job

FICTITIOUS INC wants to use the results from blind tests to get a deep understanding of how well their virtual assistant performs, and to determine how best to improve their bottom line with their assistant. They tell you that some errors are more important than other errors. There are two errors they are very concerned about:

- **"Opt-out" being under-detected**: If a user gives any indication that they want to leave the virtual assistant, and speak to a human, they should be immediately routed to a human. Failure to detect this condition leads to angry users who are likely to stop being FICTITIOUS INC customers. FICTITIOUS INC does not want to miss `#opt_out` intents.
- **"Report fraud" being over-detected.** Any conversation touching on fraud is immediately transferred to a high-cost fraud specialist. FICTITIOUS INC does not

want to incur an extra cost from conversations mistakenly routed due to an errant `#report_fraud` intent detection.

FICTITIOUS INC considers all other errors equal. They want a report on their assistant's "accuracy" which takes these into account. There's more than one way to measure accuracy. First, let's review the simplest possible definition:

$$\text{Accuracy} = \text{correct predictions} / \text{total prediction}$$

For the blind and k-folds test examples in this chapter, the overall scores I provided for each test used this simple accuracy metric. The blind test score of 0.82 came from predicting 41 out of 50 test utterances correctly, and the k-folds test score of 0.78 came from predicting 39 out of 50 test utterances correctly.

But FICTITIOUS INC doesn't just want to know how many predictions were right and wrong. It matters to them *how* the assistant was wrong. There are two types of errors they are worried about: under-selection of a correct intent and over-selection of an incorrect intent. Let's examine these error types through the sample data shown in Table 11.

Table 11 Exemplary test results for exploring error types. Data #0 shows an under-selection of #employment_inquiry and an over-selection of #reset_password.

#	Utterance	Actual intent	Predicted intent
0	can I work for you	`#employment_inquiry`	`#reset_password`
1	I need a job application	`#employment_inquiry`	`#employment_inquiry`

If accuracy is our metric, Table 11's Data #0 ("can I work for you") is one error. But from a detailed point of view, Data #0 contains **two** errors:

1. **Over-selection** of the `#reset_password` intent. The predicted intent is a *false positive* of `#reset_password` – that intent was predicted but should not have been.
2. **Under-selection** of the `#employment_inquiry` intent. The predicted intent is a *false negative* of `#employment_inquiry` – that intent was not predicted but should have been.

Similarly, if accuracy is our metric, Table 11's Data #1 ("I need a job application") is one correct prediction. But from a detailed point of view, Data #1 contains **two** correct predictions:

1. **Positive** prediction of the `#employment_inquiry` intent. The predicted intent is a *true positive* of `#employment_inquiry` – that intent was predicted, and it should have been.
2. **Negative** prediction of the `#reset_password` intent. The predicted intent is a *true negative* of `#reset_password` – that intent was not predicted, and it should have been predicted.

With these building blocks, we can build the precise metrics that FICTITIOUS INC requires. When you are worried about over-selection of an intent, you are most concerned about false

positives for that intent. The metric that considers false positives is called *precision*. When you are worried about under-selection of an intent, you are most concerned with false negatives for that intent. The metric that considers false negatives is called *recall*.

The equations for precision and recall are given below:

Precision: (True Positives) / (True Positives + False Positives)

Recall: (True Positives) / (True Positives + False Negatives)

Two ways to be wrong but only one way to be right?

It feels unfair that you don't get any credit for true negatives in the precision and recall equations. True negatives are just not that useful in assessing performance.

Consider an assistant with 20 possible intents. When it makes a single prediction, it will have either one true positive (the predicted intent was the actual intent) or one false positive with one false negative (the predicted intent and the actual intent). Every other intent will be a true negative - it will have either 18 or 19 true negatives! The math is much simpler and useful when you omit the true negatives.

You can look at precision and recall on an intent-by-intent basis. If the different error types are not important, you can blend precision and recall into a single metric. That metric is called *F1*, and it is the harmonic mean of precision and recall[4]. The equation for F1 is shown below.

*F1: (2 * Precision * Recall) / (Precision + Recall)*

In the blind test earlier in this chapter I used a sleight-of-hand, giving each intent an "accuracy" score that was actually its F1 score. That blind test result is reproduced in Table 12 with precision and recall included.

Table 12 Precision, recall, and F1 scores for FICTITIOUS INC's blind test.

Intent	Precision	Recall	F1
#appointments	0.90	0.69	0.78
#employment_inquiry	0.80	1.0	0.89
#reset_password	0.90	0.90	0.90
#store_hours	0.80	1.0	0.89
#store_location	0.70	0.70	0.70

Including precision and recall gives us more detailed information to fix the assistant, especially when compared to the confusion matrix. We can see from Table 12 that

[4] F1 is a specific form of an F-score. There are other F-scores including F0.5 and F2. Each F-score uses a different weighting for precision and recall. F1 gives equal weight to precision and recall, F0.5 gives more weight to precision, and F2 gives more weight to recall. I find it sufficient to look at just recall, precision, and F1.

`#store_location` is over-predicted based on its lower precision score. In contrast, `#appointment`'s low recall shows it is under-predicted.

FICTITIOUS INC is concerned about under-selection `#opt_out` and over-selection of `#report_fraud`. When they extend their assistant to include these intents, they should focus on the recall of `#opt_out` and the precision of `#report_fraud`. For the other intents, FICTITIOUS INC can look at the F1 scores.

An intent with low precision is over-selected. An intent with low recall is under-selected.

Precision and recall are naturally in tension with each other. Adding training data to an intent will usually improve its recall but may decrease its precision. Removing training data from an intent may improve its precision at the cost of decreasing recall. The first step to improving your assistant is understanding what kind of problem you have. Accuracy, precision, recall, and F1 are all useful in making that assessment.

FICTITIOUS INC should test their assistant's accuracy via an iterative approach, with steps for gathering data, training, testing, and improving, as in Figure 14. The first iteration of this cycle will not be perfect – and that's fine! It's a safe assumption that multiple iterations will be required, and virtual assistant builders should plan to continue iterating after their first production release.

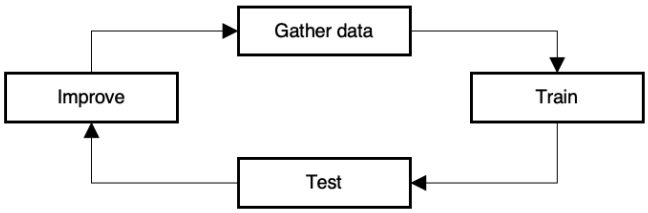


Figure 15 Testing and improving your classifier is a virtuous cycle. Assume you will need multiple iterations of the cycle and plan for some iterations after you go to production.

If you are unable or unwilling to find real user data from production – NOT just synthetic data from subject matter experts – you should stop iterating after a couple of cycles. Each cycle has diminishing returns and there is no reason to optimize an assistant on synthetic data when it is highly probable that you'll have to optimize again once you have production data. When you understand how accurate your assistant is, you're ready to test the rest of the virtual assistant solution.

7.4 Summary

- Blind testing with data from production users provides the most useful assessment of how accurate your assistant is.

- k-folds cross-validation testing can be used to identify potential accuracy problems in your assistant and is appropriate if you don't have production data to test with.
- Accuracy is a good general metric, but you can use precision and recall when you are concerned with over-selection or under-selection of a specific intent, respectively.

8

How to test your dialog flows

This chapter covers:

- Testing each dialog flow in your virtual assistant
- Automating dialog tests
- Gathering unbiased feedback on the user's experience with your assistant
- Stress testing your assistant with concurrent user tests

FICTITIOUS INC has designed a virtual assistant to handle their most frequent customer service inquiries. They formed a dream team to design and implement conversational flows for each of the major intents their virtual assistant will handle. Now FICTITIOUS INC needs to test their virtual assistant and make sure it works as well as they want it to.

The first conversational flow FICTITIOUS INC will test is for password resets. The reset password design is charted in Figure 1.

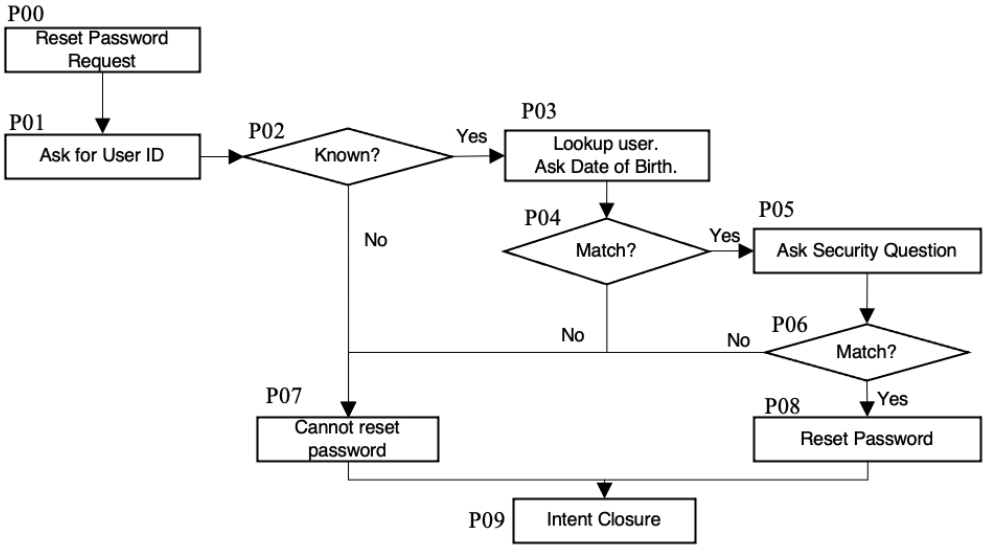


Figure 1 FICTITIOUS INC wants to verify the implementation of this Reset Password conversational flow.

FICTITIOUS INC has already evaluated how accurate their assistant was at identifying the intent in a user's question, including the ability to identify the `#reset_password` intent. A user who wants their password reset will start the conversation with an utterance indicating their intent. The system will execute several additional steps in a process flow, including asking follow-up questions, before ultimately servicing the `#reset_password` intent. In this chapter, FICTITIOUS INC will learn how to test those process flow steps.

In this chapter, we will learn how to test a dialog process flow. We'll start by functionally testing dialog responses for a single path in the process flow, using both manual and automated approaches. Then we will functionally test all paths through the process flow. Once the process flow works functionally, we will test the process flow for both user experience and system performance.

8.1 Functionally testing a dialog flow

FICTITIOUS INC's first goal in testing the password reset process flow is to make sure the user can get through all of the steps in resetting their password. They will first test the happy path – the simplest case where the user can get their password reset. This path has the following conversational elements:

1. The system acknowledges the `#reset_password` intent and asks for the user ID.
2. The system verifies the user ID exists and asks for the user's date of birth.
3. The system verifies the date of birth is accurate and asks their security question.
4. The system verifies the security question response and resets the password.

In each step of this process, the virtual assistant follows the steps in Figure 2. The virtual assistant session starts with user input. The virtual assistant first classifies that input then decides the appropriate response action, and finally responds to the input. The process repeats as long as the user keeps interacting with the system. For FICTITIOUS INC's password reset process, there will be four cycles.

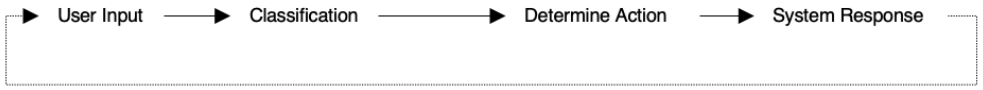


Figure 2 Execution flow in a virtual assistant

Figure 3 shows how the virtual assistant components work together on the first cycle of FICTITIOUS INC's password reset process flow. The user says, "I'm locked out of my account", the classifier assigns the intent `#reset_password`, the dialog manager finds the action for the `#reset_password` condition, and the response "Ok. What's your user ID?" is sent to the user.

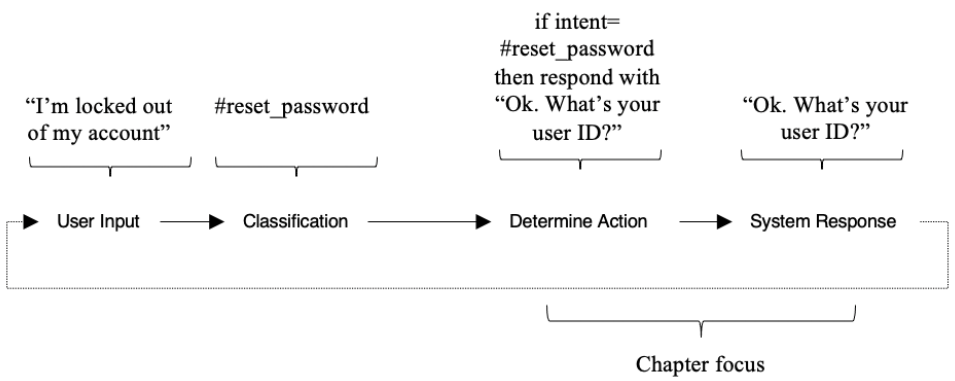


Figure 3 Execution flow in a virtual assistant and the two major testing focuses

As seen in Figure 3 the system executes several steps internally before responding to the user. The user will only see the inputs and outputs. Process flow testing should only use these inputs and outputs, as shown in Table 1. When debugging the solution, a virtual assistant builder can review the internal steps, but the process flow testing should just verify the outputs.

Table 1 Happy path process flow for a password reset

Step #	Input	Output
1	I'm locked out of my account.	Ok, I can help you reset your password. What's your User ID?

2	afreed123	I found your user ID. What's your date of birth?
3	12/25/1900	What's your favorite book?
4	Creating Virtual Assistants	I reset your password. Can I help you with anything else?

Let's test this process flow in FICTITIOUS INC's virtual assistant.

8.1.1 Manually testing a conversation flow

The simplest way FICTITIOUS INC can test their virtual assistant is through the testing interface provided by their virtual assistant platform. Most virtual assistant platforms include some form of a graphical interface for the virtual assistant. Some virtual assistant platforms include multiple interfaces:

- **"Developer" interface**: This interface is intended for developers to use as they build and unit test the virtual assistant. This interface provides debugging information which may include the intents, entities, and specific dialog nodes that were used in generating a system response. This debugging information is NOT shared with end-users in the deployed virtual assistant.
- **Sample user interface**: This graphical user interface is production-ready. It renders system responses exactly as they will be displayed to the user.

Either interface can be used to functionally test the dialog in a process flow. FICTITIOUS INC's reset password flow is demonstrated in a developer interface in Figure 4.

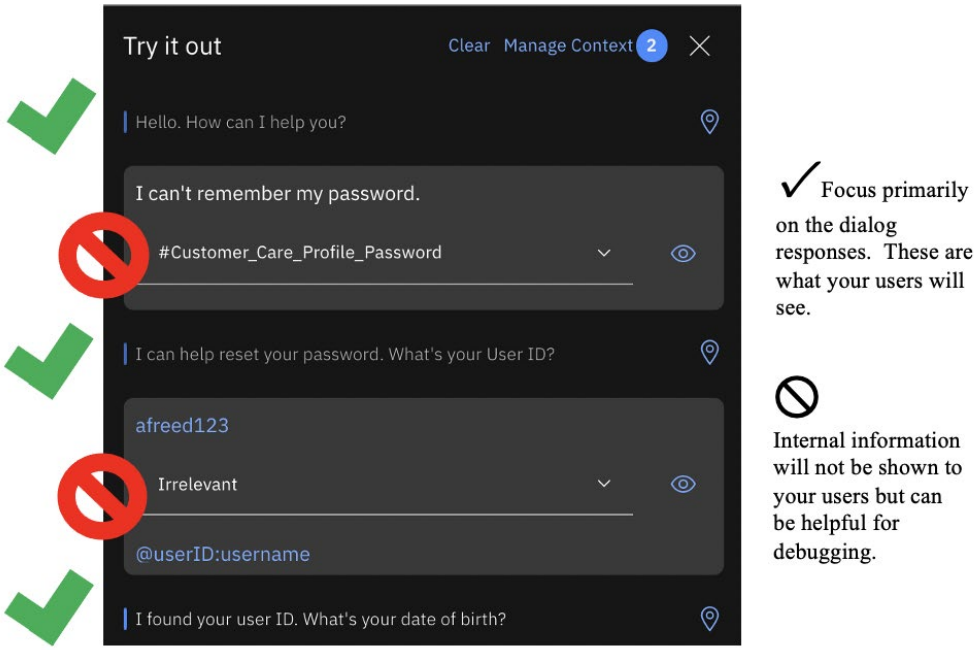


Figure 4 Use your virtual assistant platform's testing interface to manually test dialog responses. The testing interface may give you more information than you need. End-users do not know or care about the name of the intents or entities your system uses to understand their input.

FICTITIOUS INC can use either interface to test their password reset flow. Their developers will probably prefer the development interface and the rich information it provides. Their testers will probably prefer to use the same interface that the end-users will consume.

Regardless of the testing interface used, manual testing is simple to execute but does not scale very well. Manually entering four inputs to reset a password is fine once, but FICTITIOUS INC has multiple paths through their reset password process flow. And FICTITIOUS INC is implementing 17 more intents! The more complex their virtual assistant grows, the longer it will take to manually test all of the process flows.

FICTITIOUS INC won't just test a process flow once. They'll re-test their assistant each time they make significant changes to it. Over the life of their assistant, this could be thousands of tests. FICTITIOUS INC will want to automate as many of these tests as they can. Let's look at how they can automate testing the `#reset_password` process flow.

8.1.2 Automating a conversation flow test

FICTITIOUS INC will want to create test scripts for each of their process flows and be able to run these scripts at any time. They can do this by writing test scripts that use the API exposed by their virtual assistant.

Every virtual assistant platform I surveyed for this book exposes an API for testing the assistant's dialog flow. The API will require an utterance for input and will provide at least a dialog response as output. Depending on your virtual assistant platform, you may need to provide additional configuration parameters such as username, password, version, or other identifiers. Code Listing 1 demonstrates an exemplary API call.

Code Listing 1. Using an API to test the dialog response to of "I forgot my password"

```
curl -X POST -u "{your_username}:{your_password}" --header "Content-Type:application/json"
       --data "{\"input\": {\"text\": \"I forgot my password\"}}"
       "{your_url}/message?version=2020-04-01"
```

The input and output from this API call are shown in Figure 5.

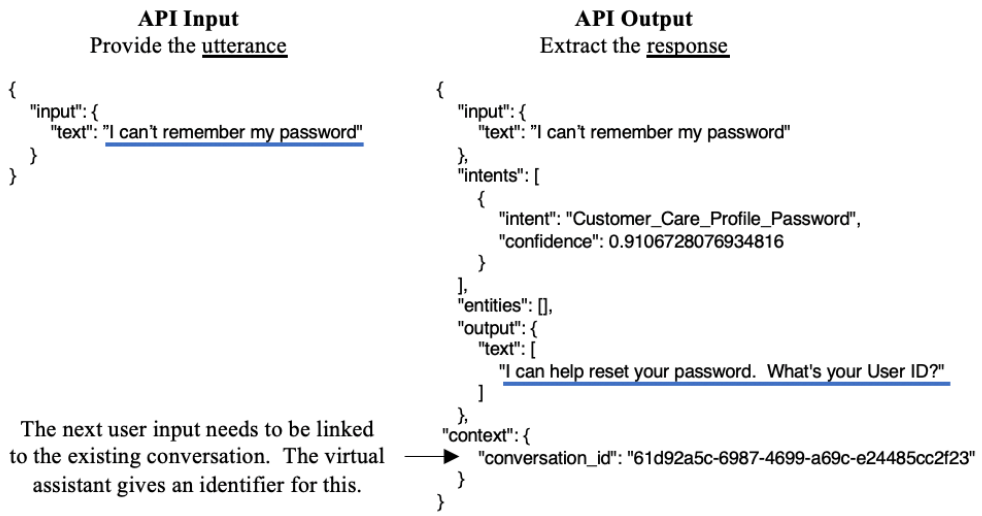


Figure 5 Using a virtual assistant's API to test the assistant's dialog response to the user's first utterance

Your virtual assistant platform may provide a Software Development Kit (SDK) which makes it easier to interact with their dialog APIs. If they don't provide an SDK, you can write your own scripts to interact with these APIs. Having code that calls the dialog API is the first step to automation.

The second step in automating dialog tests is calling the API in a repeatable fashion. FICTITIOUS INC created a shell script `dialog.py` to test a single dialog response. The contents of this script are highly dependent on their virtual assistant platform, but the script should do the following:

1. Take two input parameters: an utterance and the expected response
2. Initialize a connection to the virtual assistant

3. Call the assistant's dialog API to get the system response
4. Extract the dialog response and compare it to the expected response
5. Output four parameters: the utterance, the expected response, the actual response, and the correctness of the response.

The desired output from this script is shown in Table 2. The four pieces of information in Table 2 form the basis for verifying all of the virtual assistant's dialog responses.

Table 2 Example output format for an automated classification test

Utterance	Expected Response	Actual Response	Response Assessment
I forgot my password	I can help reset your password. What's your User ID?	I can help reset your password. What's your User ID?	Correct

FICTITIOUS INC's reset password flow is implemented in a multi-step conversation. After the assistant recognizes a `#reset_password` intent, the assistant asks the user for their user ID. The user's response to this question will be their second input in the conversation. When testing this dialog flow via an API, FICTITIOUS INC will need to link this second input to the first input, so the assistant treats it as belonging to the same conversation.

The specific linkage mechanism varies by conversational platform. One common pattern is for conversations to be linked by a *conversation ID*. Figure 6 demonstrates an API request and response for the second turn of a password reset conversation, linked by conversation ID.

API Input
Provide the utterance linked to the existing conversation

```
{
   "input": {
      "text": "afreed123"
   },
 "context": {
      "conversation_id": "61d92a5c-6987-4699-
a69c-e24485cc2f23"
   }
}
```

API Output
Extract the next conversational response

```
{
   "input": {
      "text": "afreed123"
   },
   "intents": [],
   "entities": [
      {
         "entity": "userID",
         "location": [0, 9],
         "value": "username",
      }
   ],
   "output": {
      "text": [
         "I found your user ID. What's your date of birth?"
      ]
   },
   "context": {
      "conversation_id": "61d92a5c-6987-4699-a69c-e24485cc2f23"
   }
}
```

The second reset_password question is driven by finding an entity, not an intent.

Figure 6 Testing FICTITIOUS INC's password reset flow through an API, continuing the conversation with a user

response to the question "What's your user ID?"

The specific implementation for continuing a conversation via API varies by virtual assistant platform. One possible implementation is shown in Code Listing 2, where each subsequent call to the virtual assistant dialog API uses the `conversation_id` from the previous call[1].

Code Listing 2. Using an API to test the second password reset process flow step ("What's your User ID?")

```
curl -X POST -u "{your_username}:{your_password}" --header "Content-Type:application/json"
       --data "{\"input\": {\"text\":
       \"afreed123\"},\"context\":{\"conversation_id\":\"xxxxxxxx-xxxx-xxxx-xxxx-
       xxxxxxxxxxxxx\"}}" "{your_url}/message?version=2020-04-01"
```

FICTITIOUS INC wants to test multiple messages in a single conversation. Now that they know how to test a single message with their assistant's API, they can expand their `dialog.py` script to test an entire conversation. The script has similar steps as when it tested a single message but has a `for` loop with an iteration for each message. The pseudocode of this script has the following steps:

1. Take one input: a comma-separated values (CSV) file with two columns[2], each row containing an utterance and its expected response
2. Initialize a connection to the virtual assistant
3. For every row in the CSV file:
    a) Call the assistant's dialog API to get the dialog response to the given input utterance
    b) Extract the actual dialog response and compare it to the expected dialog response
    c) Output four parameters: the utterance, the expected response, the actual response, and the correctness of the response.

A pseudocode version of `dialog.py` is found in Code Listing 3.

Code Listing 3. Example dialog.py contents

```
import json
import sys

# Script is called with a test data filename as parameter
all_test_data = json.load(open(sys.argv[1]))

# Pseudocode – this will vary per virtual assistant platform
assistant = new Assistant(your_username, your_password, your_url)
context = {}
assistant.message("")

for test_data in all_test_data:
    result = assistant.message(test_data['input'], context)
```

[1] In this system, an API call that does not include a conversation_id is assumed to initiate a brand new conversation.
[2] Other common formats include JSON or XML. These formats make it easier to expand the tests with additional input context or additional output verification parameters.

```
    expected_response = test_data['output']
    actual_response   = result['text']
    context = result['context']
    if(actual_response != expected_response3):
        print("Expected {} but got {}".format(expected_response, actual_response))
        return -1

print("Passed the test")
return 0
```

The `dialog.py` script can be invoked with a test case file like `reset_password_happy_path.json` as found in Code Listing 4.

Code Listing 4. Example reset_password_happy_path.json contents

```
[
  {
    "input":"I'm locked out of my account",
    "output":"Ok, I can help you reset your password.  What's your User ID?"
  },
  {
    "input":"afreed123",
    "output":"I found your user ID. What's your date of birth?"
  },
  {
    "input":"12/25/1900",
    "output":"What's your favorite book?"
  },
  {
    "input":"Creating Virtual Assistants",
    "output":"I reset your password.  Can I help you with anything else?"
  }
]
```

With this new script, FICTITIOUS INC can execute any individual conversation test in seconds. FICTITIOUS INC can create a file for each conversation scenario they wish to test and automate the execution of all of them. They can use these tests to verify the functionality they've built into their assistant, and they can use these tests as a regression suite to run any time they change their assistant in the future.

An even faster path to automating your dialog tests

This chapter shows you how to implement your own automated dialog tests. You can tie these tests into testing frameworks like JUnit and unittest. There are also existing tools you can use for even faster time-to-value.

Botium has a dialog testing framework with adapters for many of the most common virtual assistant platforms. This service also offers integration to continuous improvement/deployment pipelines. Read more at https://www.botium.ai.

[3]Exact string matching is brittle since messages may frequently change. You may update this condition to use partial string matching or to only check the final dialog response.

Watson Assistant users can use the open-source WA-Testing-Tool library at https://github.com/cognitive-catalyst/WA-Testing-Tool. This Python tool can run dialog tests from CSV or JSON files. (The examples in this book were patterned off of WA-Testing-Tool.)

Now that FICTITIOUS INC can test any conversational flow, either manually or automatically, they are ready to test all of their conversational flows. They have so far just functionally tested "happy path" of their password reset process flow. Let's look at testing the other paths through their password reset flow.

8.1.3 Testing the dialog flowchart

FICTITIOUS INC had drawn a flow chart for each of the process flows their virtual assistant supports. These process flows were useful during the design and build time so that every member of their dream team knew how the assistant would work. These process flows are also invaluable during the testing phase and can be used to build functional test cases. FICTITIOUS INC's password reset dialog flowchart is displayed in Figure 7.

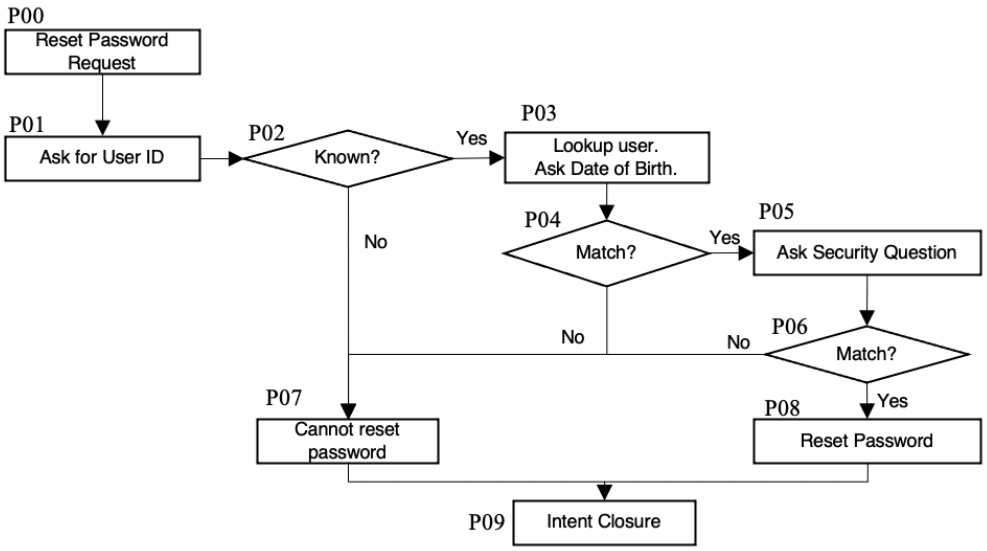


Figure 7 FICTITIOUS INC's dialog tree for the reset password flow.

This process flow includes several rule conditions. Each of these rule conditions affects the process flow and needs to be tested. The five rule conditions are shown in Table 3.

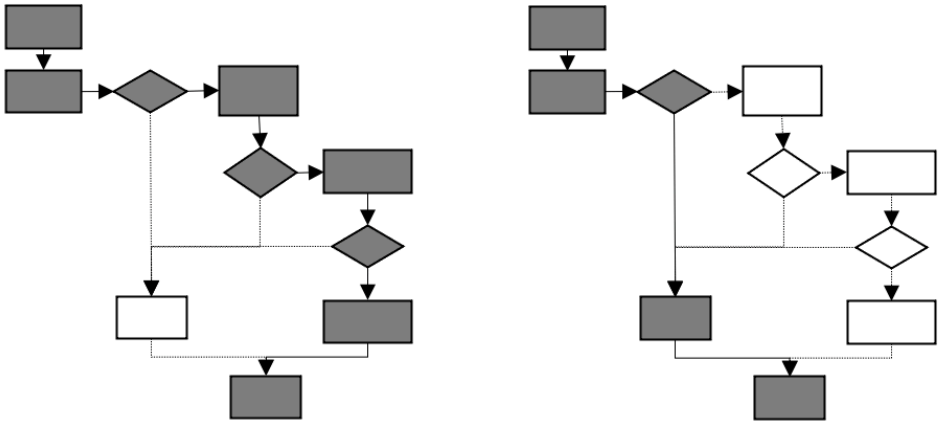
Table 3 Dialog rule conditions in the Reset Password process flow

Node	Rule Condition	Response
P01	`if intent==#reset_password`	"I can help reset your password. What's your User ID?"
P03	`if user_id is not null`	"I found your user ID. What's your date of birth?"
P05	`if input_dob==user_id_dob`	Ask the user's specific security question
P07	`if user_id is null or security_match==false`	"I'm sorry, I can't reset your password"
P08	`if security_match==true`	"I reset your password. Can I help you with anything else?"

FICTITIOUS INC wants to test all of the password reset flow paths. This means testing each of the conditional branches in the process flow. There are four paths[4] through the reset password flowchart in Figure 7:

- Happy Path – The user's password is reset. (P00,P01,P02,P03,P04,P05,P06,P08,P09)
- Unhappy Path – Security question incorrect. (P00,P01,P02,P03,P04,P05,P06,P07,P09)
- Unhappy Path – User date of birth is incorrect. (P00,P01,P02,P03,P04,P07,P09)
- Unhappy Path – The user ID is unknown (P00,P01,P02,P07,P09)

Two of FICTITIOUS INC's password reset conversation flows are shown in Figure 8.



1. Happy Path Test

2. Unhappy Path Test (one of several)

Figure 8 Visual representation of individual test cases for the reset password flow. Each test case can only test a handful of dialog nodes and conditions. The untested flows in each test case use dashed lines and unfilled boxes.

[4]There are five conditions but only four paths for the test case. The condition that does not get an individual dialog test case is that the intent is truly #reset_password. The intent matching is tested in the accuracy tests. (See Chapter 7).

For FICTITIOUS INC's `#reset_password` process flow, the number of possible paths through the flow is finite. FICTITIOUS INC can get complete functional test coverage of this process flow by enumerating the paths creating test cases for each. An example test case is demonstrated in Table 4.

Table 4 Example test case for the reset password "happy path". If automated, the test can verify each response produced by the system.

Step	Test Action	Expected Response
1	Enter "reset my password"	System responds, "I can help reset your password. What's your User ID?"
2	Enter "afreed123"	System responds, "I found your user ID. What's your date of birth?"
3	Enter "12/25/1900"	System responds, "What's your favorite book?"
4	Enter "Creating Virtual Assistants"	System responds, "I reset your password. Can I help you with anything else?"
5	End test	

Testing a process flow for a virtual assistant includes running many tests like these. The biggest challenge in testing dialog flows is the sheer number of possible paths through your virtual assistant. The number of test cases for a process flow can grow exponentially with the length of the flow, depending on how many paths there are through the flow and how intertwined those paths are.

Test cases can easily be written from a well-diagrammed process flow like FICTITIOUS INC's password reset flow. In this diagram, each of the most common error conditions is captured, such as user ID or date of birth validation failures. These "unhappy" paths can be traced on the flow diagram. What about the conditions that don't quite fit on the flow diagram? How can FICTITIOUS INC test those?

8.1.4 Testing the unexpected error paths

FICTITIOUS INC created dialog flowcharts for each of their major intents and used these for testing. The flowcharts showed the most expected paths through each dialog flow. However, there are unexpected paths that can be common across all process flows, and FICTITIOUS INC did not include these on their flow charts. These paths are caused by errors that could occur on nearly any line of dialog. As such, they are often not included in dialog flowcharts. They still need to be tested.

These unexpected paths include:

1. The assistant has identified the **wrong intent** from the user's utterance
2. The user input was **not understood**
3. The user tried to **opt-out** of the assistant

Let's explore how FICTITIOUS INC can test these.

WRONG INTENT

FICTITIOUS INC's team has already tested how accurate their assistant is at identifying the user's intent. No matter how accurate their assistant is, there is still a chance it will make mistakes. Any time the assistant interprets open-ended user input, there is a chance the assistant will interpret that input incorrectly. In Figure 9 the assistant selects the wrong intent for the user's problem, by selecting the `#reset_password` intent instead of `#employment_inquiry`. Will the user be able to correct the assistant and get to the flow they need, or will the assistant stubbornly continue? FICTITIOUS INC needs to test the user experience to see how classification mistakes will impact the user.

In Figure 9 the user's initial utterance is misclassified. The statement "I can't find the job application site" was classified as the `#reset_password` intent with 0.4 confidence, a fairly low value, instead of being classified as #employment_inquiry. The user gets frustrated by this misclassification.

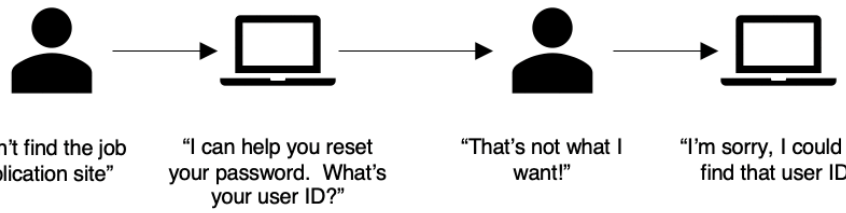


Figure 9 If your assistant selects the wrong intent, does the user have any way to bring the conversation back on track? It's good to test what will happen to a user when the wrong intent is selected. Users should be able to correct the virtual assistant if it gets the intent wrong, or they should at least be able to escape a process flow.

FICTITIOUS INC can reduce the number of process flows that start based on the wrong intent in multiple ways:

- Explicitly confirm the intent with the user before starting the process flow. FICTITIOUS INC could start the `#reset_password` flow with the confirmation question "Would you like to reset your password?". This lengthens the conversation by one question but assures that the system understands the user's input.
- The assistant can automatically start a process flow only if the confidence of the intent prediction is high. If the intent confidence is not high, the assistant can ask the user to confirm the predicted intent, or the assistant can offer a default response like "I'm sorry, I didn't understand that".

These approaches are demonstrated for FICTITIOUS INC's `#reset_password` flow in Figure 10. FICTITIOUS INC may decide to automatically initiate the `#reset_password` flow if that intent is detected with 0.75 confidence, to confirm the intent if the confidence is between 0.5

and 0.75 confidence, and to ask the user to restate their utterance if the intent is less than 0.5 confidence.

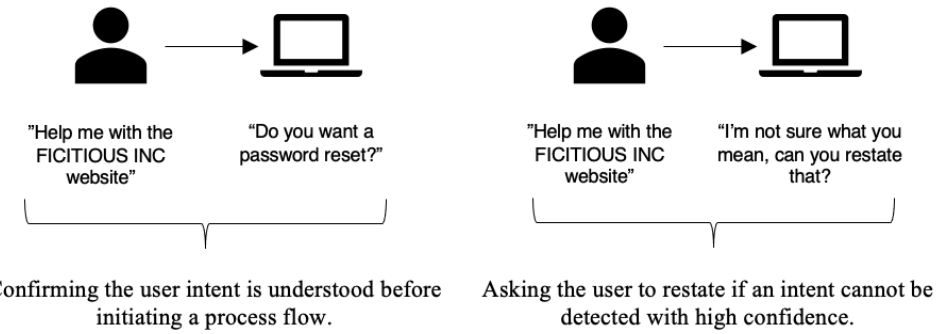


Figure 10 FICTITIOUS INC has multiple ways to make sure a process flow is only started with a well-understood intent.

Regardless of which design approach FICTITIOUS INC uses, they should test what happens when the assistant does not recognize the correct intent with high confidence.

Mistakes happen!

User input can be misunderstood by the assistant, or users may just say "the wrong thing". Your assistant should be designed with the assumption that the classifier will make some mistakes, and there should be resolution mechanisms built into the assistant or the application that embeds the assistant. Your virtual assistant testing plan must include testing this functionality.

"I DIDN'T UNDERSTAND"

FICTITIOUS INC's `#reset_password` flow had included four user inputs: the initial utterance, a user ID, a date of birth, and a security question answer. In their flowchart, they defined a success path and a failure path for each of these inputs. For instance, the user may provide a valid user ID (success) or an invalid ID (failure). But what happens if the input is not a user ID at all?

At any point in a conversation, a virtual assistant may simply misunderstand the user. The user may say something completely unexpected. Text messages can be corrupted by a bad auto-correct, or the user resting something on their keyboard. Voice messages can be corrupted by speech-to-text errors or by insidious background noise. An assistant should be adaptive in case the user input does not match expectations.

Virtual assistants are generally coded with rules to handle input matching various conditions. For instance, the question "What is your User ID?" may be coded to expect a single alphanumeric word in response, since user IDs don't contain spaces. If the input is anything else, the virtual assistant may use a default or fallback condition to respond. This "anything else" response can be implemented generically so that it can be used on any dialog

node. Figure 11 illustrates a fallback condition that responds to unexpected input by first saying "I'm sorry, I didn't understand" and then repeating the question.

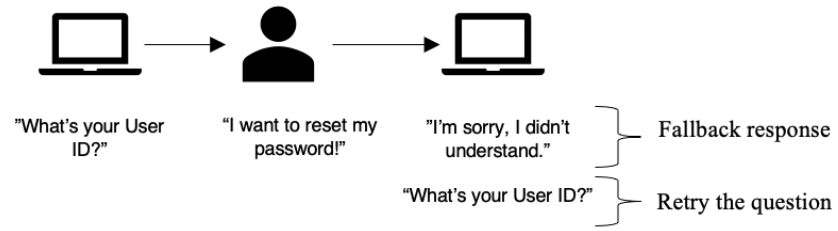


Figure 11 If the user says something unexpected, they may get a fallback response like "I'm sorry, I didn't understand." The assistant can ask the original question again.

In some virtual assistant platforms, this fallback condition can encapsulate additional logic, by treating a misunderstood utterance as a "digression" or "interruption"[5]. FICTITIOUS INC can use this kind of digression to implement logic that retries any misunderstood question only once. The first time the user is misunderstood, the assistant replies "I'm sorry, I didn't understand" and repeats the last question. The second time the user is misunderstood, the assistant replies "I'm sorry, I still didn't understand" and directs the user to an alternate resolution, as shown in Figure 12. The fallback condition needs to keep track of conversational context, to determine if it is being invoked for the first or second time in a conversation.

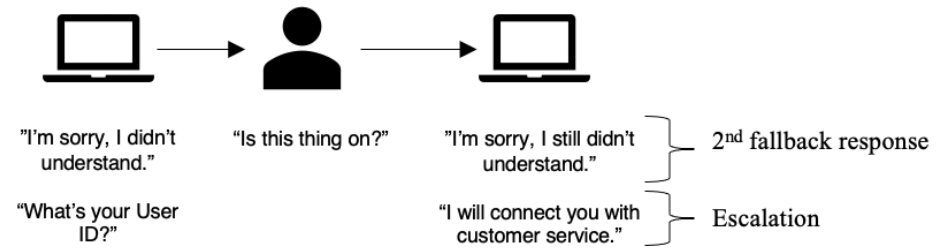


Figure 12 A fallback response can use context from earlier in the conversation. FICTITIOUS INC can code their assistant such that if the user is misunderstood twice, the assistant plays a different fallback message as well as escalating the user out of the conversation.

[5] Virtual assistant platforms advertise digressions as a way for the user to switch topics in the middle of a conversation. It turns out that they are also a convenient way to implement a miniature dialog flow that's initiated when a user is misunderstood. When a digression flow finishes, it returns the conversation to the place it was before the digression occurred.

When this kind of fallback logic is implemented in a global condition, how should it be tested? Should FICTITIOUS INC update their flowcharts to show this conditional logic on every branch?

Most virtual assistant builders do not include this fallback logic on every single diagram – it adds a lot of clutter. Similarly, most builders do not write a test case of this fallback logic on every single dialog condition. FICTITIOUS INC is well-served by testing this fallback logic in the dialog branches where it is most likely to occur. For most of their process flows, including password resets, this includes the first two or three branches in the dialog flow.

OPT-OUT

FICTITIOUS INC wants users to be able to opt-out of their virtual assistant. If the user ever says something like "get me out of here" or "I want to speak to a human", the virtual assistant will help the user get the human assistance they desire[6]. An exemplary opt-out dialog flow is depicted in Figure 13. Since FICTITIOUS INC wants this to always be an option, they've chosen not to include it on their flowcharts.

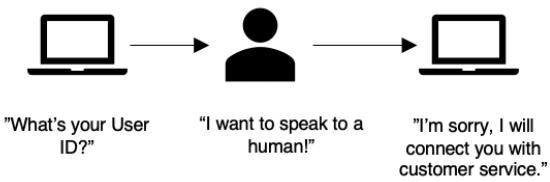


Figure 13 FICTITIOUS INC allows users to "opt-out" of the virtual assistant at any time.

Since opt-out is possible on any dialog node, should FICTITIOUS INC write a test case opting out of every specific dialog node? Probably not – there is a point of diminishing returns in testing a global function like opt-out. The longer a user converses with the system, the less likely they are to opt-out. This means the most important dialog nodes to test are the first few nodes. For password resets, FICTITIOUS INC should especially verify that the user can opt-out of the intent capture (the first message in the conversation) and the user ID capture (the first message in the reset password flow).

Letting the user opt-out is good manners

FICTITIOUS INC may be very proud of their virtual assistant, and it may save them a lot of money compared to fielding inquiries with human customer service reps. But it's a violation of a social contract with your users to ignore them if they explicitly tell you they don't want to talk to a machine.

[6] In another clever use of digressions, opt-out is frequently implemented as a digression that never returns. Using a digression is one way to make the opt-out logic globally available.

In practice, almost all end-user opt-out requests occur in the first two or three messages in the conversation. If they make it to the fourth message, there's a good chance they'll stick with the assistant.

Rather than preventing opt-outs, focus your energy on improving your assistant to the point that users don't want to opt-out of it.

FICTITIOUS INC has now tested all functional aspects of the password reset process flow. They have verified that the virtual assistant faithfully implements the flowcharts they built, that the assistant can handle an unexpected utterance, and that the assistant lets users opt-out. Now that the assistant works well functionally, they should test the non-functional aspects as well.

8.2 Non-functionally testing a dialog flow

It's important to make sure that your virtual assistant works functionally. How well does it work for your users? You can't be sure until you've done some user experience testing.

8.2.1 User experience testing

FICTITIOUS INC has designed their assistant to satisfy their users' most common needs. They designed dialog flows intending to help their users get what they need as quickly as possible. But even if they have executed the design phase to the best of their abilities, there's still a lot of time between the design, build, and test phases. There's a difference between whiteboarding a dialog flow and actually experiencing it through a text or voice interface. When you test the design the way the user will actually use it, you'll find things that don't work as well as you thought they would on some diagram.

There's something even better than pretending to be a new end-user. Go find some actual new end-users! There's no substitute for the fresh perspective from new users. Just like your subject matter experts can't usually predict the full range of data end-users will send to the system, they can't usually predict how the entire virtual assistant experience will work for end-users. Pretending to be an end-user is a cheap but inferior substitute; don't be afraid to get the real thing.

There's no substitute for the fresh perspective from new users.

FICTITIOUS INC can find some friendly faces to help in this phase. Their developers can recruit people from nearby teams, or even friends and family. New end-users will not be biased by the thought processes used in the assistant's design. When these users test the assistant, FICTITIOUS INC should give these users only the amount of guidance they plan to give to production end-users, to avoid biasing these users in the areas where their feedback and fresh perspective is the most important.

Figure 14 contrasts two different ways FICTITIOUS INC can introduce their assistant to new users. The way they introduce the assistant significantly affects the way the users will consume the system.

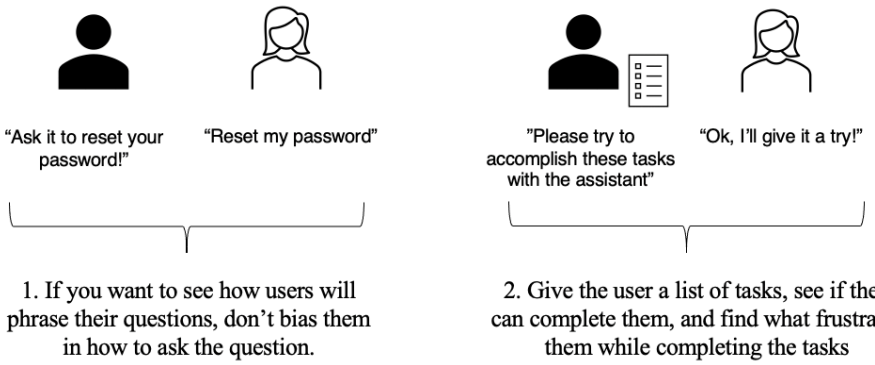


Figure 14 The instructions you give your end-users will affect the experience they have and the feedback you give.

Design your user experience test so that you get the feedback you need the most. I never want to pass up an opportunity to find out how users ask their initial questions, so I prefer not to give too much advice on what to ask. Users may spend a little time "playing" around (asking the assistant "tell me a joke!") but they will quickly discover what's "in-bounds" for the assistant.

Don't forget to observe how users react to the dialog

In every conversational assistant I've built I have found some dialog that seemed okay during the design phase but felt wrong once I acted like a user. This is especially true in voice conversational systems, where some seemingly important caveat turned a system response from useful into an interminable bore. With a careful eye (or ear!) you can often find some rough edges that you can smooth to improve the user experience.

Most virtual assistant builders will want to test that users can both get into a process flow as well as execute it to completion. You can give the user specific instructions which may include the initial question you want the user to ask. This will take less time and give you feedback on the process flow itself, but you won't find out how well the intents are designed. You can instead give the user a set of open-ended tasks. This will help you test how well you can identify intents and how well the process flows for these intents work. Table 5 has sample instructions FICTITIOUS INC can give their new users, based on what part of the user experience they will focus their testing on.

Table 5 Specific and open-ended instructions for specific user experience tests

Process Flow	Specific Instructions	Open-Ended Task
Store Hours and Store Location	Ask the assistant where they are located and when they are open.	Plan a trip to FICTITIOUS INC with the assistant.
Reset Password	Ask the assistant to reset your password.	The FICTITIOUS INC website gave you an error "Incorrect username or password". Resolve this by using the assistant.
Employment inquiry	Ask the assistant for a job application.	You're looking to earn some extra cash in your spare time. Apply to FICTITIOUS INC.

The differences between approaches in Table 5 are stark. Giving specific instructions to the users heavily biases their experience but ensures that you will definitely test the process flow you intend. For the "Store Hours and Store Location" tests, the specific instructions will certainly execute the `#store_hours` and `#store_locations` process flows. The open-ended version of this test asks a user to "plan a trip" – they may not specifically ask for both hours or locations, and they may not ask for either! Depending on the feedback FICTITIOUS INC needs, they can adjust the instructions to be more specific or more open-ended. For instance, they might instead say "Plan a trip to FICTITIOUS INC with the assistant, *including the when and where*."

User experience testing takes time, but it's worth it!

Allow some time to review the feedback from this user experience testing. The feedback from this test is the closest you will get to post-production feedback. This feedback can be worth its weight in gold! Take the most significant feedback and incorporate it into your assistant before going into production. This can ensure a much smoother rollout when your assistant goes to production.

8.2.2 Load Testing

After FICTITIOUS INC has tested the functionality and user experience of each intent for a single user, FICTITIOUS INC needs to make sure the assistant works at scale. FICTITIOUS INC needs to make sure their assistant responds quickly and correctly when multiple consumers use the assistant at once. **Load testing** is when a software system is tested with many simultaneous users[7].

FICTITIOUS INC should have a plan for how many concurrent users they expect to serve in production. They can estimate this number from existing support channels. For instance, if 500 unique users visit the FICTITIOUS INC website in an hour, they could assume half of the users will use the chat. If each chat takes 2 minutes to complete, FICTITIOUS INC can assume an average load of approximately 10 simultaneous users. (250 users times 2 minutes per user chat divided by 60 minutes is just under 10 simultaneous users, assuming

[7]The strain on the system from serving multiple users at once is called the "load". Hence the name "load testing".

a random distribution.) FICTITIOUS INC would then load test their system with 10 concurrent tests.

FICTITIOUS INC may be able to have 10 different humans testing their system. Perhaps they can throw a pizza party while their team works on manual simultaneous tests of the assistant. However, this approach will not scale well. The more concurrent users they need to test, the harder it is to test manually, and the more valuable automated tests become. Ideally, FICTITIOUS INC has already automated their functional testing, and they can use these same automated functional tests to do the load testing as well.

Load testing can be an expensive proposition. If FICTITIOUS INC's virtual assistant is hosted on a cloud provider, FICTITIOUS INC could run up a hefty cloud bill since the test traffic looks "real" to the provider. They should check their cloud plan details to see if or how they will be charged. Still, it may give FICTITIOUS INC good peace of mind to verify that their cloud provider can handle FICTITIOUS INC's production level.

If FICTITIOUS INC is self-hosting their assistant on their premises, they should certainly stress test their assistant with some concurrent testing. This ensures that the virtual assistant has been allocated the right infrastructure and that this infrastructure has been properly configured, to handle production loads.

The first thing FICTITIOUS INC should do in the load test is verify that each individual test runs as expected. FICTITIOUS INC verified the functionality of their reset password flow by making sure it identified the intent, asked for and verified the user ID, asked for and verified the date of birth, asked for and verified a security question, and finally reset the password. This flow should work the same way whether one user is using the assistant or many. Figure 15 depicts what could happen in an unsuccessful load test.

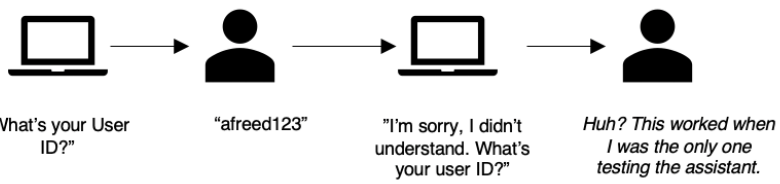


Figure 15 FICTITIOUS INC's process flows should work the same way whether there is one or many simultaneous users. Load testing verifies that the system performs correctly under concurrent usage.

As long as the system is performing correctly for simultaneous users, FICTITIOUS INC should also verify how quickly the system responds to the user. The time between the user's message and the system's response is called **response time**. Users expect responses in a few seconds at most. If the system takes longer than that to respond, they will get frustrated and possibly leave the assistant. Figure 16 shows the user's perspective when response time is high.

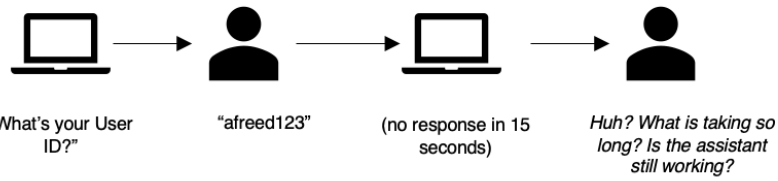


Figure 16 If the assistant responds slowly, users will wonder what is going on. Load testing ensures that the assistant is responsive enough to provide a good user experience.

Ways to mitigate slow responses from your assistant

Ideally, your virtual assistant will always respond to the user in a few seconds or less. This may not always be possible. The assistant may slow down under heavy usage, or the assistant may interact with slow external systems. FICTITIOUS INC's password reset service might always take 30 seconds to execute.

A good user experience can be preserved if the assistant indicates to the user that progress is being made and that the assistant has not failed. Assistants should always announce when they are starting a process that will take more than a few seconds. FICTITIOUS INC's assistant could say "Please wait up to 30 seconds while I reset your password".

Depending on the specific type of assistant, there are additional ways to let the user know the assistant is still active. For a text-based conversational assistant, the user interface can display a temporary "thinking" icon or message, like chat programs that show when a user is typing. For a voice-based assistant, the assistant can play music during a waiting period instead of staying silent.

FICTITIOUS INC should include all of the intents in their load test. Ideally, their load test will simulate the distribution of requests they expect during production. If password resets are the most common intent, using 30% of expected call volume, then 30% of the tests in the load test should test the password reset flow. Using a representative variety of tests is important. If FICTITIOUS INC only load tests the simplest intents, they will not properly stress test the system.

Load testing tools

Loadrunner and JMeter are two popular load testing tools. Either tool can test your virtual assistant by exercising the assistant's APIs.

Once you've successfully verified your assistant's functionality, user experience, and performance under multiple users – congratulations! It's time to deploy your assistant out to some real users!

8.3 Summary

- Each dialog flow should be tested thoroughly to assure it works as designed. The

flows can be tested manually or via automation.

- The assistant should be resilient if the user does something unexpected or if the assistant "makes a mistake".
- Feedback from users who didn't design or build the assistant is invaluable and should be collected carefully.
- The assistant should be tested to ensure it responds correctly and quickly with multiple simultaneous users.

9

How to deploy and manage

This chapter covers:

- Tracking changes to your virtual assistant over time using a source control management solution like Git
- Managing multiple versions of your virtual assistant at once
- Safely deploying code to production

The day FICTITIOUS INC decided to build a virtual assistant was the happiest day of John Doe's life. John, a developer at FICTITIOUS INC, could not wait to get started building. John raced ahead of the rest of his project team, spending nights and weekends developing the virtual assistant. John's project team was happy about his enthusiasm, but nobody on the team was exactly sure what John was up to all the time.

Communication with John was a struggle. He was slow to respond to instant messages and emails. Occasionally a team member would walk to John's desk find him furiously typing new code on his workstation. They would ask John how the latest feature was going, and John would give a lengthy soliloquy. By the time the team member got back to their own desk, they had forgotten what John told them.

Early in the development phase, this eccentricity did not bother the rest of the FICTITIOUS INC project team. John Doe was pumping out code, he seemed generally happy, and the team had a vague sense he was making progress. But as FICTITIOUS INC readied to functionally test the virtual assistant, this lack of transparency was causing strife for the project team. Nobody knew exactly what features were done and which features were working. The virtual assistant changed multiple times per day, but nobody was sure how or why.

This chapter is the story of how FICTITIOUS INC escaped this Wild West scenario and how they established transparent development and change control procedures. In this chapter, you will learn how to manage changes to your virtual assistant in a transparent

way, and how to track which version of your virtual assistant each member of your team is using.

9.1 Where to store your code

FICTITIOUS INC's development team currently consists of just one developer – John Doe. Let's track their evolution of source code management.

9.1.1 The Wild West approach

As the first developer on the FICTITIOUS INC virtual assistant project, John has not worried about sharing his code with anybody. John is only worried that the hard drive on his laptop may die someday. Thus, John tries to remember to save a copy of the virtual assistant code to a shared drive about once a day (or whenever he remembers)[1].

John does not have any hard and fast rules about how he backs up the virtual assistant code. He usually changes filenames to indicate the date of the backup, or to add a note about the state of the code. Sometimes he makes a mistake when he backs up the code, overwriting a previous version of the assistant. John feels a little guilty when this happens, but not too bad. He knows in his head what state the code is in … mostly. His backup strategy can be seen in Figure 1.

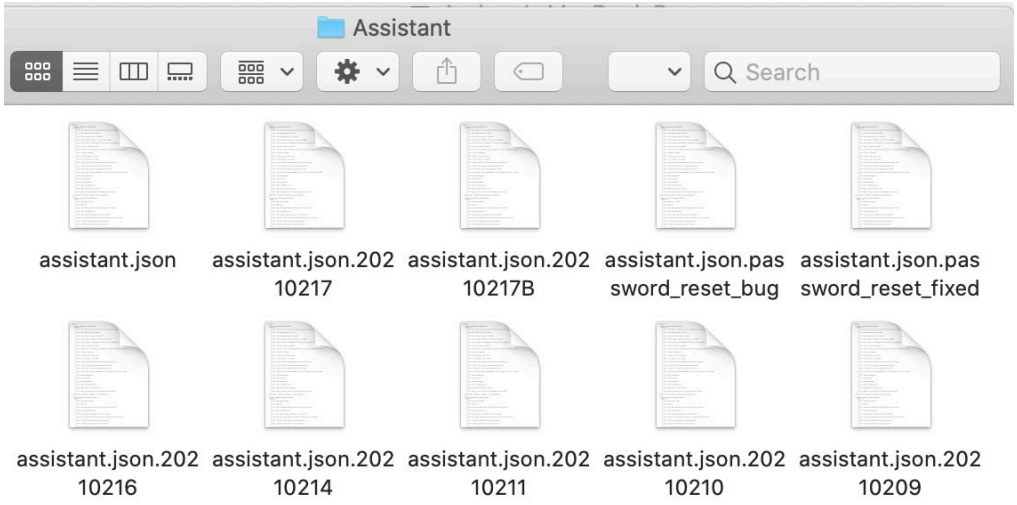


Figure 1 A quick and dirty way of managing versions of a virtual assistant. What do these file names mean? Which one is the latest version, and which versions are obsolete? Does the file assistant.json work?

[1] Many virtual assistant platforms store virtual assistant code online rather than on physical workstations. The source code management problems in this section still apply. When I use a hosted virtual assistant platform, I still download a copy of the code to my workstation to review it.

This backup strategy makes it hard to work with John. John's team can find his files on the shared drive. But when someone wants to test the virtual assistant, which version should they use? They can guess by choosing a file based on the file name or date, but they have no assurances what is actually in that file. They don't know what's in the file, if it works well, or if it contains features and functions they are interested in.

FICTITIOUS INC's team could just go ask John, but we can see that John is a bit unreliable and hard to get a hold of. And what if John is out sick – how can anyone know which version of the code to use?

FICTITIOUS INC has a handful of problems:

- They don't know which files represent the latest versions of the code.
- They don't know what's in any particular version of the code.
- They don't know which files represent stable versions of the code.

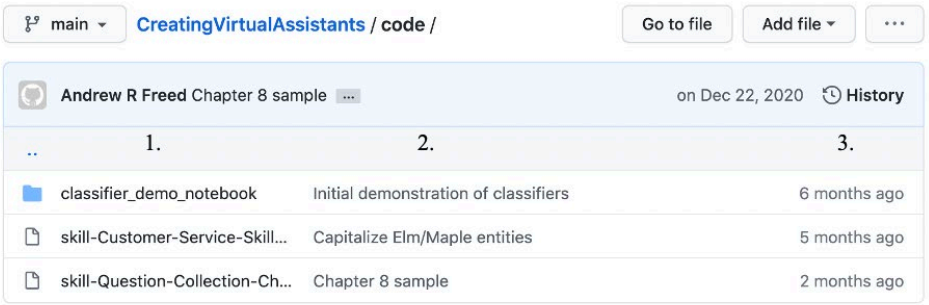
Let's see how FICTITIOUS INC can address these problems with source control.

9.1.2 Using source control for code

Source control is the practice of tracking and managing changes to a series of files, particularly source code files. John Doe's ad-hoc process barely meets this definition. There are many software solutions that offer robust source control management capabilities. At a minimum, a source control solution should be able to provide a complete list of all the files it is tracking, the file contents, all the changes made to those files, who made the changes, and why. This information is stored in a source code *repository.*

There are many possible source control platforms. At the time of this book's writing, the most popular source control platform is Git[2]. Git has a powerful command-line client, and there are many third-party user interfaces as well as hosting platforms. GitHub[3] is a popular hosting platform for using Git. The source code used in this book is stored in a GitHub repository[4]. The home page in GitHub for that source code is shown in Figure 2.

[2] https://git-scm.com/
[3] https://github.com/
[4] https://github.com/andrewrfreed/CreatingVirtualAssistants



1. Every file tracked in source control
2. Reason for last update
3. Last update date for each file

Figure 2 Summary view of the source code for Creating Virtual Assistants as seen in the book's GitHub repository.

Anyone who connects to a GitHub repository gets all of this information. They can browse the repository and view each of the files. They can immediately see when each file was last updated. They can even download all of the files to their own computer using Git's *clone* command.

This solves FICTITIOUS INC's first problem – identifying the latest version of the code. Git goes above and beyond this requirement by tracking all versions of the code. Git also supports viewing and downloading any version of code stored in its repository. No more walking down the hall and asking John which version is which – FICTITIOUS INC can easily find the version they want in Git. Now let's look at how FICTITIOUS INC can use source control to solve their second problem – what's in a given version of code.

By default, source control systems will show you the most recent version of every file in the repository. It's often useful to see what changed in a file and why. Figure 3 shows a history of the source code I used for the demo in Chapter 2.

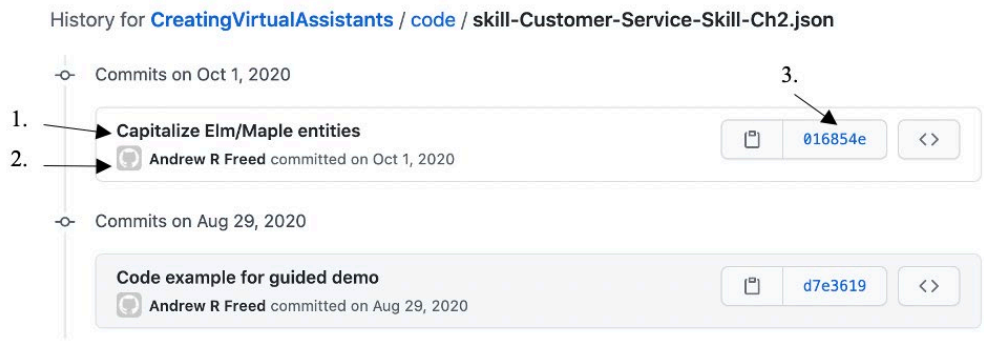


Figure 3 Using a source control management application lets you see what changed, when, and why. On October 1, I updated the source code for Chapter 2 to include the capitalization of the Elm and Maple entities.

In Figure 3, several pieces of information are immediately available:

- The latest copy of `skill-Customer-Service-Skill-Ch2.json` was saved on October 1, 2020, by Andrew Freed.
- The reason that `skill-Customer-Service-Skill-Ch2.json` was updated was to capitalize the Elm and Maple entities.
- The contents of `skill-Customer-Service-Skill-Ch2.json` after the change are stored in version 016854e[5]. The contents of this file before the change are stored in version d7e3619. Each version of the file belongs to a different *commit*[6].

This solves part of FICTITIOUS INC's second problem – determining what each version of the code means. The descriptive summaries attached to Git commits describe the evolution of code. The commit histories in a source repository can be combined into a single stream called a *commit log*. If John was using Git, FICTITIOUS INC could review the commit log to find out what was in the code – they wouldn't have to walk down the hall to ask John. Figure 4 shows an exemplary commit log for a single file in this book's GitHub repository.

[5]Git calls these "commit hashes". For technical reasons beyond the scope of this book, versions are referred to by hash rather than by a number. Using numbers for versions is much harder than you might expect. You'll need to take my word on this one.
[6]Multiple source control systems use the term 'commit'. A commit contains one or more files that were changed and stored to the repository at the same time.

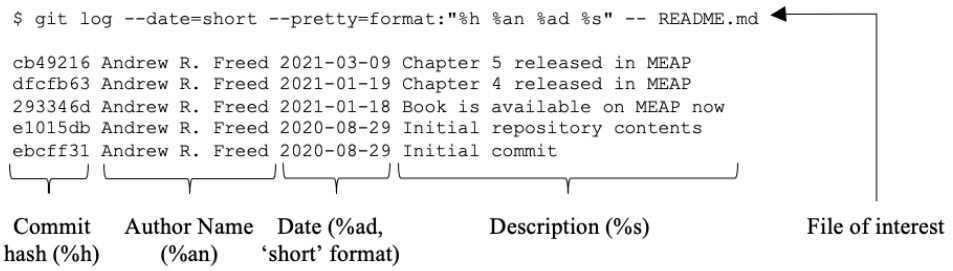


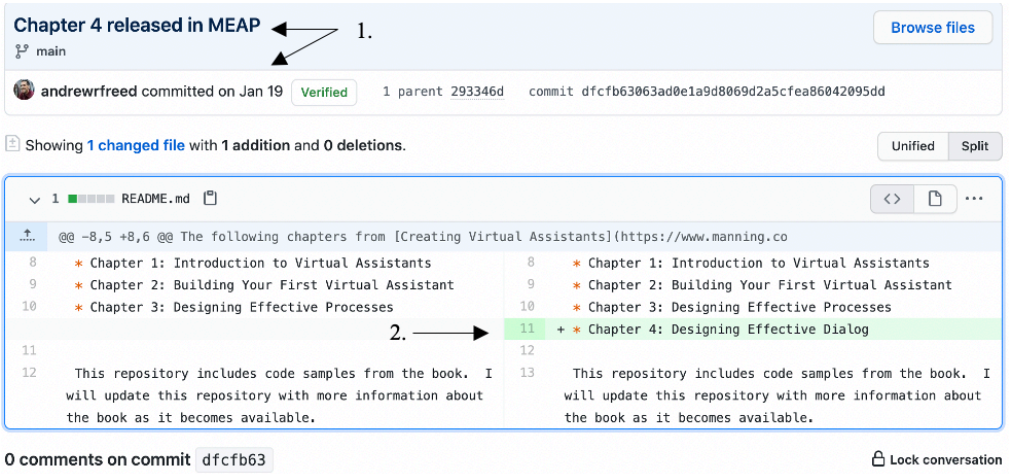
Figure 4 Commit log for one file in this book's GitHub repository.

Use descriptive commit messages!

A commit log is much more useful if it contains a detailed description of the changes. The message "Capitalize Elm/Maple entities" is much more useful than a message of "fixed bug".

Commit messages are a good way to share what was done and why. Developers, take the time to write descriptive commit messages. It helps the whole team! (And, it will help "future you" too!)

A commit log may not be enough information, so source control also provides an additional level of detail. Figure 4 demonstrates a Git view that compares two versions of the same file. This viewer can be used to drill down into a specific change, to see how it was done. Reviewing changes at this detailed level serves two purposes. First, a reviewer can determine if the change summary matches the actual changes. Second, a reviewer can provide feedback on the changes, through a process called code review.



1. Summary of changes
2. Line-for-line comparison showing the exact changes

Figure 5 Differences between two versions of the same file in a Git repository. This view shows the "before" and "after" versions side-by-side and annotates the differences between them. In this case, the change was to add one line of text as denoted by the "+" sign. (Lines that are removed are marked with "-" sign.)

Drilling down into this level of detail fully solves FICTITIOUS INC's second problem. The line-level detail shows them exactly what each version of the code means. Before using source control, members at FICTITIOUS INC had to walk down to John's office and ask, "what's new in `assistant.json` today?" Now they just review the commit log!

FICTITIOUS INC's final challenge was identifying a stable version of the code. By convention, the latest stable code in a Git repository should be found in the `main` branch[7]. FICTITIOUS INC has multiple ways they can establish this convention. The first and simplest path is shown in Figure 5.

[7] A branch represents a line of development. Every source code repository has at least one branch, and in Git the default branch is called 'main'. A full explanation of Git branching is beyond the scope of this book.

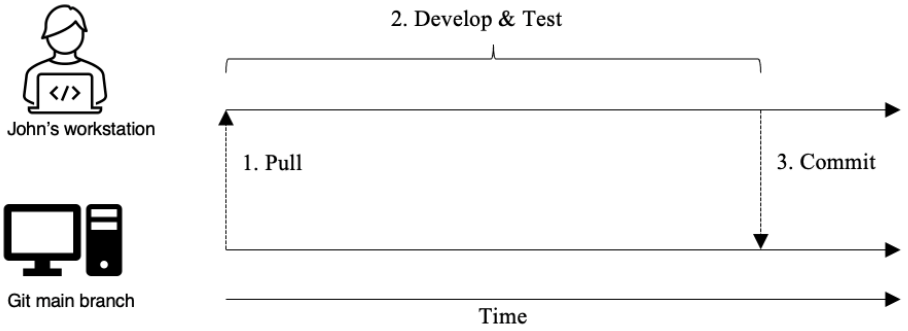


Figure 6 Simplest possible Git development model. John pulls the latest code from the Git repository, makes some changes, and when the changes are stable, he commits them back to the repository.

This simple development process ensures that the `main` branch always has stable code. The steps in the process are:

1. John gets the latest stable code from the Git repository.
2. John makes one or more changes to the code, then tests the code to ensure it works well and is stable.
3. John commits the updated code to the Git repository.

The best part of this development process is its simplicity. There are only three steps, and the `main` branch always has stable code. However, there are a handful of drawbacks to this approach as well:

- Nobody can see the code John is working on until he commits it. Since John can't commit code until he's sure it is stable, significant time may elapse before anyone else can test or review it.
- This puts a significant burden on John. It's difficult for him to share any work unless he shares files another way.
- Interim versions of John's code are not stored in the Git repository.

These shortcomings can be addressed by modifying the development process. When John starts new development work, he can create a new *branch* in the repository for the specific feature he is working on. John can safely commit interim updates to this branch, and only commit code to the `main` branch when he's sure it is stable. This updated process is depicted in Figure 6.

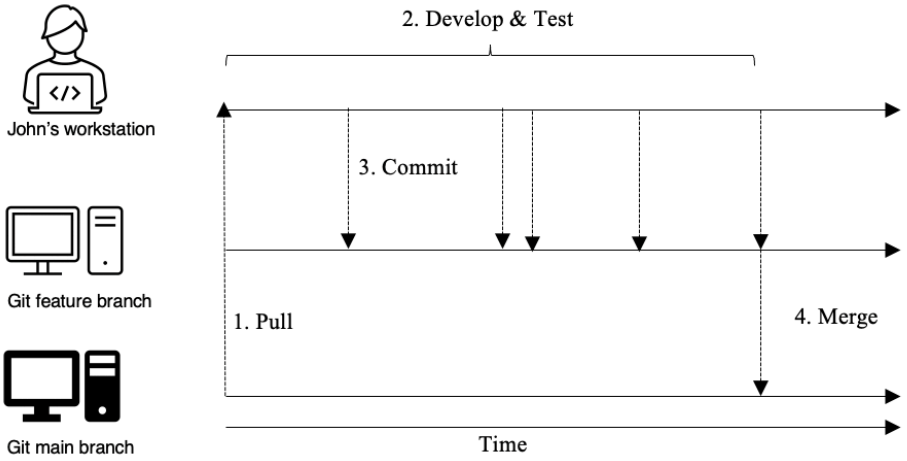


Figure 7 Enhanced development process. John's interim updates are stored in Git in a working space called a "feature branch". Other team members can access his code through this branch. Only stable changes are moved to the main branch.

The steps in this new process are now:

1. John gets the latest stable code from the Git repository (same as before).
2. John makes one or more changes to the code and tests them
3. At any time, John can commit code to his feature branch. This allows other team members to access it through the Git repository.
4. When his new code is stable, John merges his code into the `main` branch.

FICTITIOUS INC can use this process to elegantly answer two of their biggest questions:

- **What's the latest version of John's code?** John's feature branch contains his latest code.
- **What's the latest stable version of the team's code?** The `main` branch contains the latest stable code for the entire team.

Choose the right size process and workflow for your team

The processes demonstrated in this book work well for teams with a small number of developers. Git workflows can be very simple if the number of developers is one! As your team increases in size, additional rules or coordination processes will be required. Git is a powerful and highly configurable system and can support a variety of complex processes. These processes are out of the scope of this book. There are many good Git books available on the market.

With Git, FICTITIOUS INC can fully track the lineage and status of their virtual assistant source code. They can find any version of their virtual assistant code and understand what it means. Now they need a process for testing and sharing this code.

9.2 Where to run your code

FICTITIOUS INC's development process evolved as they moved John's code from John's laptop to a shared network drive, to a Git repository. This made it easy to track the code as it was developed. FICTITIOUS INC doesn't just want to store code, they want to run it! In this section, we'll see how FICTITIOUS INC evolves the runtime environment for their virtual assistant code.

9.2.1 Development environment

At the beginning of the virtual assistant process, John Doe is working by himself. John writes code and tests the virtual assistant in the same place. Figure 7 shows John's working process. As John writes code, it runs in a *development* environment.

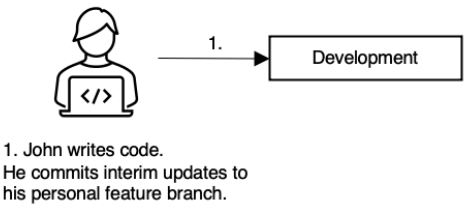


Figure 8 The simplest runtime environment. There is one version of the code and it is used by one developer.

Depending on FICTITIOUS INC's virtual assistant platform, the development environment may run on his workstation, on a FICTITIOUS INC server, or on a cloud. Regardless of platform, a development environment is where new features and functionality are first built. The development process often requires exploration and may have a "two steps forward, one step back" ethos. Because of this, the development environment should be expected to have the latest code but not necessarily stable code.

Latest code vs stable code

When new code is written, it often does not function correctly the first time. This "latest" code may take experimentation or effort to repair a piece of code. The development environment is the place where code repair happens. Once the code is repaired it is declared stable and ready for sharing with others.

John can store his unstable code on his workstation or in a personal feature branch[8] on the Git repository. Eventually, John will stabilize his code and it will be ready to share with his team for them to test. John knows to move his stable code into the `main` branch and his team knows to look in that branch for stable code. But where will they run this new code? If they use John's development environment, John is not able to do any further development

[8] A feature branch is a temporary or "short-lived" line of development used to isolate changes from the main branch until those changes are stable.

work. Similarly, John's team cannot test code on the development server while he is using it.

9.2.2 Test environment

FICTITIOUS INC has two different environmental needs. John needs a space with potentially unstable code to do his development. John's broader team needs a space with stable code for testing the assistant. FICTITIOUS INC's new *test* environment is shown in Figure 8.

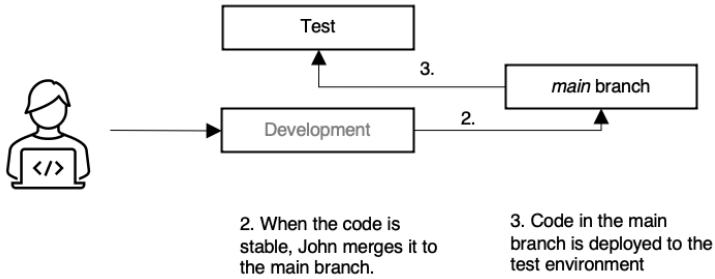


Figure 9 Development process with separate environments for development and test.

Development and test environments should at minimum be logically separated. Table 1 includes other common isolation requirements based on virtual assistant type.

Table 1 Comparing development and test environment isolation requirements in cloud-based and on-premise virtual assistants.

Requirement	Cloud-based virtual assistant	On-premise virtual assistant
Isolation between development and test is achieved by	Using a separate service or instance (depending on the platform's terminology).	Using separate hardware.
Access control is achieved by	Using different access control lists per service or instance.	Using different access control lists per machine.

The test environment is literally called a "higher environment"[9], and in higher environments processes are more strict. In this development process, every code change is first built and tested in a development environment. John never writes code directly on the test environment. John only works in a development environment, committing interim code to his feature branch and stable code to the `main` branch. The only code source for the test environment is the `main` branch. Code may be *developed* in the development environment, but it may only be *deployed* in the test environment.

The code deployment process varies by virtual assistant platform. In some platforms, a virtual assistant is deployed by importing a single file through the virtual assistant provider's

[9] In Figure 8, the test environment is intentionally drawn above (higher than) the development environment.

user interface or API. In other virtual assistant platforms, several source code files may need to be compiled and built into one or more packages that are deployed to a server. There is significant variation between the major virtual assistant platforms in the deployment process.

Automating deployment

Regardless of the virtual assistant platform, it is best if the deployment process can be automated. (Automated deployments are referred to as Continuous Integration/Continuous Deployment, or CI/CD.) An automated continuous integration process for virtual assistants could look like:

1. Wait for code to be merged into the `main` branch.
2. Extract the latest code from the `main` branch.
3. Build and/or package the code into a deployable bundle.
4. Deploy the bundle to the test environment.

The development process will repeat many times over the life of the virtual assistant. John will develop code in the development environment, and it will be tested (when ready) in the test environment. Any time someone proposes a change, like fixing a defect, it will first go through the development environment and then to the test environment. Eventually, FICTITIOUS INC will build enough features, with sufficient quality, that they are ready to deploy the assistant to production. How will FICTITIOUS INC do that?

9.2.3 Production environment

You may have guessed already that FICTITIOUS INC can't repurpose their development or test environments. FICTITIOUS INC needs a third environment – a *production* environment. This environment is shown in Figure 9.

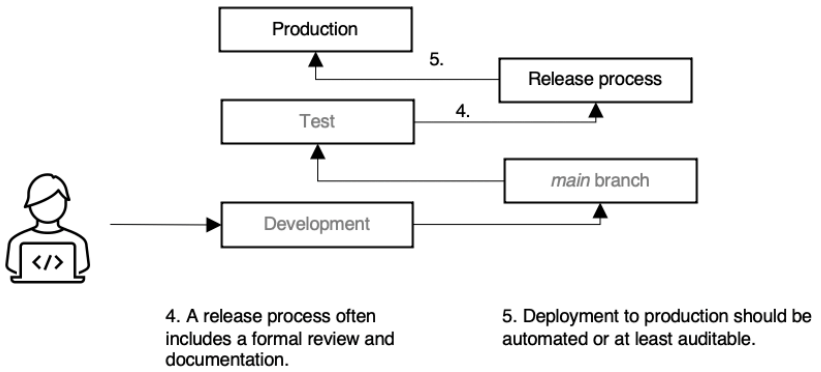


Figure 10 FICTITIOUS INC adds a production environment.

FICTITIOUS INC's production environment is an even higher environment than the test environment. The production environment must be logically separated from development and test environments, and physical separation is a good idea if FICTITIOUS INC can afford it.

Production needs additional separation from development and test

Logical separation of production from other environments is a must and physical separation is an extra safeguard. Many companies go even further and enforce a "separation of duties". In a separation of duties approach, a person who has access to a development environment cannot have access to a production environment, and vice-versa.

Most virtual assistant builders have some release process they go through before deploying to production[10]. At a minimum, the release process should include some quality assurance, such as the assistant passing a series of manual or automated tests. Additionally, the process should include logging the specific version of the code that is deployed to production[11].

FICTITIOUS INC should deploy to their production environment with the same tools they use to deploy to the test environment. Ideally, the deployment process should be automated. If it is not automated, it should at least be carefully monitored. It's one thing to make a mistake when deploying to the test environment – it's inconvenient for your testers. It's another thing to break the production environment – now nobody can use your virtual assistant!

FICTITIOUS INC has achieved a wonderful milestone the first time they deploy to production. Virtual assistants are never done the first time they are deployed to production. FICTITIOUS INC will be making many updates to production over the life of their virtual assistant. How can they manage this process?

9.2.4 After the first production deployment

FICTITIOUS INC followed a relatively lightweight development process for their first production release. FICTITIOUS INC only had one stream of development by definition since they did not have a virtual assistant in production. The development stream only included new feature work. After FICTITIOUS INC takes their assistant to production, they have a second stream of changes, for improving the existing functionality. These two views are shown in Figure 10.

[10] This chapter describes a basic development and release process. I have seen many processes much more complex than the ones I describe here. The important thing is to establish *some* process and make sure it is understood and followed by the entire team.

[11] Perhaps the simplest way to record the code deployed to production is to note the Git commit hash of main and store it in a document. A better way is to use Git's "tag" feature to mark the specific commit with an easily found identifier like "release-1.0".

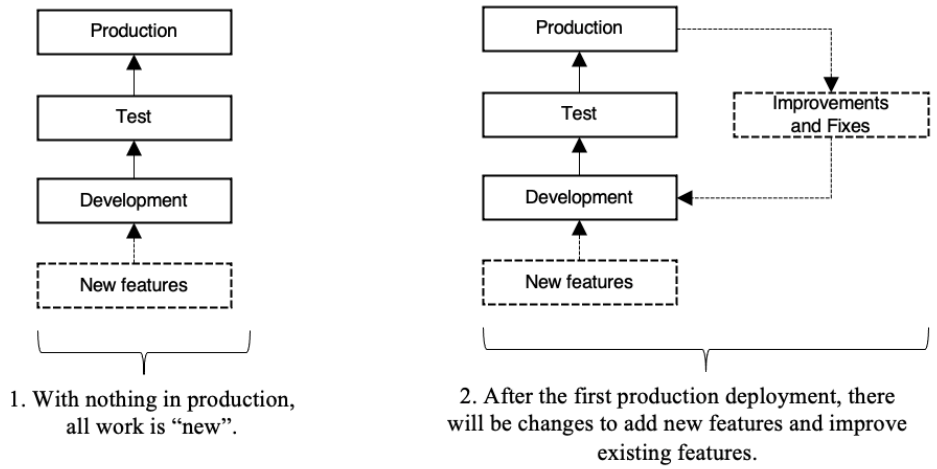


Figure 11 Once an assistant is in production, the nature of development changes. New feature development needs to be balanced with improving existing features.

FICTITIOUS INC has several release planning options for how to handle these two different modes of input.

"ONE THING AT A TIME" RELEASE PLANNING

The first way FICTITIOUS INC can treat the two competing streams of "improvement" and "feature development" is to make the streams take turns. For every new release of their virtual assistant, they can focus on either adding features or improving existing features. One version of this approach is shown in Figure 11.

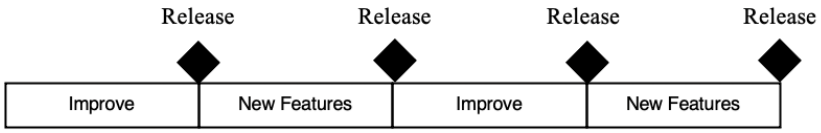


Figure 12 "One thing at a time" release planning. Each release includes only new features or only improvement of existing features.

The virtue of this approach is its simplicity. FICTITIOUS INC can use their existing Development, Test, and Production environments as-is. The entire team can focus entirely on improving the assistant, or entirely on new features, without worrying if one of those efforts is impacting the other. FICTITIOUS INC can even vary the length of the iterations, perhaps following each two-week cycle of improvement with a four-week cycle of new feature development.

The downside of this approach is its rigidity. If FICTITIOUS INC wants to do an improvement in the middle of a new feature iteration, they have to wait for the next improvement cycle. FICTITIOUS INC could instead consider a less rigid release planning approach.

"TWO THINGS AT A TIME" RELEASE PLANNING

Rather than isolating improvement and feature development into separate releases, FICTITIOUS INC can do them both at the same time. Each release of their assistant can contain a mixture of new features and improvements to existing functionality. This release model is shown in Figure 12.

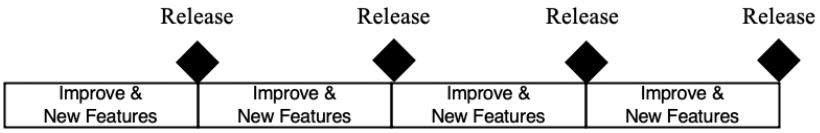


Figure 13 Improving the assistant while adding new features. Each release iteration contains some improvements and some new features.

Like the "one thing at a time" approach, FICTITIOUS INC can use this approach with their existing Development, Test, and Production environments. The team can vary the blend in each release iteration – some may be heavier on improvements, some on new features.

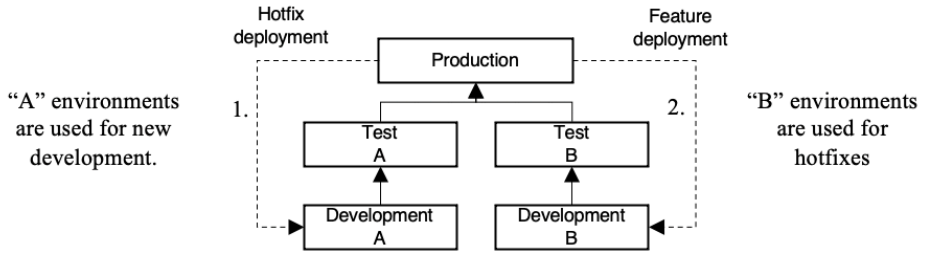
The best part of this approach is that it is very flexible. FICTITIOUS INC can incorporate an improvement into any release iteration. If we assume four-week iterations, then any particular improvement or bug fix is only four weeks or less from being deployed to production. I favor short iterations – short iterations limit the number of changes going into a single deployment and thus reduce risk.

However, even this flexible approach may not be enough for FICTITIOUS INC. FICTITIOUS INC's assistant is probably not production-ready in the middle of an iteration. If a critical bug is found, waiting several weeks for it to be production-ready so that a "hotfix"[12] can be deployed may not be an option. Fortunately, there's one more release planning approach FICTITIOUS INC can consider.

SEPARATE DEVELOPMENT AND "HOTFIX" ENVIRONMENTS

FICTITIOUS INC can adjust either of the two previous approaches to release planning and still be ready to quickly deploy hotfixes. The only cost is that they will need additional development and test environments, with the new development and test environments solely dedicated to hotfixes. These environments and the associated workflow are shown in Figure 13.

[12] My definition of a hotfix: a critical fix that must be deployed as soon as possible, or sooner. For instance, if your legal department says "fix this or we'll get sued."



1. When hotfixes are deployed to production, they are merged back into the "A" environments.

2. When new features are deployed to production, the "B" environments are overwritten with the new production code

Figure 14 Complete separation of development and "hotfix" environments.

Development follows the same path as before for feature and improvement work. Changes are propagated from development to test to production. The hotfix environments are generally idle and are only actively used when a hotfix is being developed and tested. When a hotfix is deployed to production, it must be merged back into the main development branch. When a regular release is deployed to production, the hotfix environments must be refreshed with the latest production code.

Caveats for the multiple workstreams and environments approach

The "separate development and hotfix environments" approach is very doable in Git, but a bit beyond the scope of this book. Research the "Gitflow workflow" to find out how to set up the Git branching for this approach. The other two approaches can be achieved with the minimalist branching scheme described in this chapter.

The multiple workstreams approach can be more difficult to do in "low-code" virtual assistant platforms. If your virtual assistant "code" is stored in a JSON file, it can be hard to merge changes from one branch to another, and you may instead need to duplicate the changes in multiple branches. This difficulty can be mitigated by keeping hotfixes as small as possible.

There are many possible variations of these release planning approaches. You can always start with a simple approach and evolve into a more sophisticated approach when you need it. The release planning approaches are summarized in Table 2.

Table 2 Comparison of release planning approaches

	One thing at a time	Two things at a time	Separate streams
Release contents	One of: features or improvements	Features and improvements	Development stream is mostly features. Hotfix stream is only fixes.

# of environments	Three: One development, one test, one production	Three: One development, one test, one production	Five: Two development, two test, one production
Resource utilization	High – all environments are in frequent use.	High – all environments are in frequent use.	Hotfix stream is mostly idle.
Hotfix delivery timing	Deliver at the end of the current iteration.	Deliver at the end of the current iteration.	Delivery as soon as the fix can be built and tested.
Analysis and improvement	Improvements are isolated iterations – the analysis is performed against a stable base.	Improvements are made while new features are being added. Iterative analysis may be skewed by the new features.	Improvements can be isolated if they are done on the hotfix stream.

9.3 Using source control for other assets

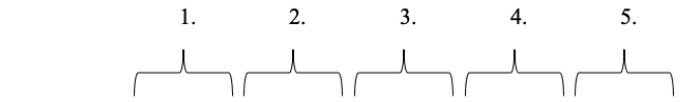
Source control is not just a good idea for source code, it's a good idea for any asset related to a virtual assistant. You can use the same source control repository for all of your virtual assistant artifacts.

For instance, both the training data and test data for your virtual assistant should be tracked in source control.

Training data: The training data for your virtual assistant will evolve as you add new intents and improve existing intents. Changes to your assistant's training data can have a big impact – positively or negatively – on your assistant and it's important to be able to trace a variation in the assistant's accuracy back to the change(s) that caused it. Training data is commonly stored in comma-separated value (CSV) files which are easily tracked in Git.

Test data: The longer an assistant is in production, the more opportunities you have to collect test data. It's important to add more test data in the early days of your virtual assistant, especially if you had a small amount of test data to start with. You will likely run many different accuracy tests against your assistant over its lifetime. When the accuracy changes, you'll want to be able to track it back to the training and test data used in the experiment.

Figure 14 demonstrates how FICTITIOUS INC tracks the accuracy of their assistant over time, and how they know exactly which version of their training and test data they used for each test. By tracking these data sources so closely they can extrapolate why the assistant's accuracy went up (or down) based on how the data changed.



Assistant Overall Accuracy	80%	77%	87%	85%	88%
Training Data Version	1	2	3	3	4
Test Data Version	1	1	1	2	2

1. The initial accuracy assessment is stored as a baseline
2. New training data made the assistant less accurate
3. New training data improved the assistant's accuracy
4. New test data showed the assistant was less accurate than expected.
5. The training data is improved to increase the assistant's accuracy

Figure 15 Evolution of FICTITIOUS INC's virtual assistant accuracy, associated with the version of training and test data used.

The revision history in Git can be a great reminder to keep your test data fresh. If your Git repository shows your test data was last updated `11 months ago`, it's probably time to refresh your test data!

Training and test data are probably the most important non-code assets to store in source control, but it's also a good idea to store any scripts you use to test or deploy your assistant in the source control repository as well.

9.4 Summary

- Source control helps you manage changes to your virtual assistant. It can show you the who, what, and why behind every change.
- Use separate development, test, and production environments. Each environment has a specific purpose.
- Source control is not just for code – use it to version your training data, test data, deployment scripts, and more!

10

How to improve your assistant

This chapter covers:

- Examining and deciphering where your assistant needs improvement
- Finding where your assistant is failing according to your success metric and rectifying these failures.
- Improving the assistant where it has the highest inaccuracy.
- Motivating virtual assistant owners for continuous improvement.

FICTITIOUS INC has deployed their virtual assistant to production, but they are not achieving the success metrics they outlined for the solution. The virtual assistant was supposed to reduce the burden on other customer service channels, but these channels have not seen a significant reduction in user activity. FICTITIOUS INC knows how to troubleshoot their traditional applications but does not know where to start troubleshooting their virtual assistant.

FICTITIOUS INC needs to quickly drill down into WHY their assistant is not performing well. They need to find out if their conversational flow does not work for users, or if the intent mapping they have done does not work, or if there is some other core problem with their assistant.

FICTITIOUS INC is in good company. Deploying a virtual assistant to production is not the end – it is only the beginning! Figure 1 demonstrates the continuous improvement in a virtual assistant's lifecycle. Continuous improvement is broadly applicable in software projects, and it is especially applicable for virtual assistants.

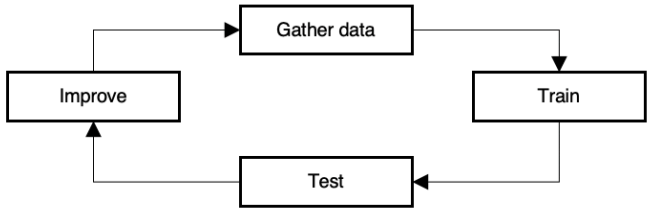


Figure 1 Improvement is part of a continuous cycle in the life of a virtual assistant. This cycle continues even after an assistant is deployed to production!

This cycle does not stop for FICTITIOUS INC when they deploy their assistant. The first improvement cycle after deploying to production is the most informative. This is where FICTITIOUS INC will learn which of their assumptions were correct and which ones need to be revisited.

Deploying a virtual assistant to production is not the end – it is only the beginning!

In this chapter, we will learn how FICTITIOUS INC can identify where their assistant needs the most improvement. FICTITIOUS INC has chosen "successful containment" as their key success metric and we will use that to drive our investigation. *Containment* for virtual assistants is the percentage of conversations handled entirely by the virtual assistant. (A conversation that is not escalated to a human is "contained"). FICTITIOUS INC's "successful containment" modifies this definition: only conversations that finish at least process flow are "successfully contained".

With successful containment in mind, we will use a data-driven approach to evaluate their virtual assistant, including the dialog flows and intent identification. We will conduct a single evaluation of FICTITIOUS INC's virtual assistant. FICTITIOUS INC will need to evaluate their virtual assistant many times over its lifetime. Let's start by looking for the first improvement FICTITIOUS INC needs to make.

> "Change is the only constant in life."
>
> **Heraclitus, Greek philosopher**

10.1 Using a success metric to determine where to start improvements

Analyzing a virtual assistant can feel like a daunting process. There are many different types of analyses. Where should FICTITIOUS INC begin their analysis? Analysis should be centered on a success metric. This success metric forms a guiding principle for all analysis

and improvement. Any potential analysis or improvement work should be prioritized based on how it impacts a success metric.

FICTITIOUS INC's chosen success metric is "successful containment". "Successful containment" is better aligned with their users' needs better than "containment". If a user quits a conversation before getting an answer, that conversation is contained, but FICTITIOUS INC does not consider the conversation a success. Table 1 contrasts containment and successful containment.

Table 1 Sample scenarios and how they are measured. FICTITIOUS INC will use "successful containment".

Interaction ends with...	Containment	Successful Containment
The assistant suggests escalating the conversation to a human agent.	Not contained	Not contained
The user demands a human agent before a process flow completes.	Not contained	Not contained
The user quits the chat in the middle of a process flow.	Contained	Not contained
The user starts and finishes a process flow successfully.	Contained	Contained

FICTITIOUS INC will use three data point to start the analysis of their assistant: overall successful containment, volume by intent, and successful containment by intent. These data points enable analysis of each intent. From these data points we can find which intents are having the largest impact on the overall successful containment.

To simplify the analysis, we will only consider five of FICTITIOUS INC's intents. These intents and their associated metrics are shown in Table 2. Based on this table, which intent would you explore first?

Table 2 FICTITIOUS INC's metrics for conversation volume and successful containment, broken down per intent.

Intent	Volume	Contained	Uncontained	% Volume	Containment
`#appointments`	50	15	35	10%	30%
`#employment_inquiry`	50	40	10	10%	80%
`#reset_password`	200	80	120	40%	40%
`#store_hours`	100	95	5	20%	95%
`#store_location`	100	90	10	20%	90%
Total	500	320	180	100%	64%

`#appointments` has the lowest overall containment at 30%, but it is a low-volume intent. And `#reset_password` is the largest source of uncontained conversations, comprising two-thirds of the total uncontained conversations. If FICTITIOUS INC can fix what's wrong in

those two intents, their virtual assistant will have much higher containment and thus be more successful. Since `#reset_password` has the biggest problem, FICTITIOUS INC should start there.

10.1.1 Improving the first flow to fix containment problems

Solving problems is easier when you know what the specific problems actually are. FICTITIOUS INC has identified the `#reset_password` flow as the biggest source of non-contained conversations. This is the most complex of FICTITIOUS INC's process flows, and that's probably not a coincidence. Let's reacquaint ourselves with FICTITIOUS INC's `#reset_password` flow in Figure 2.

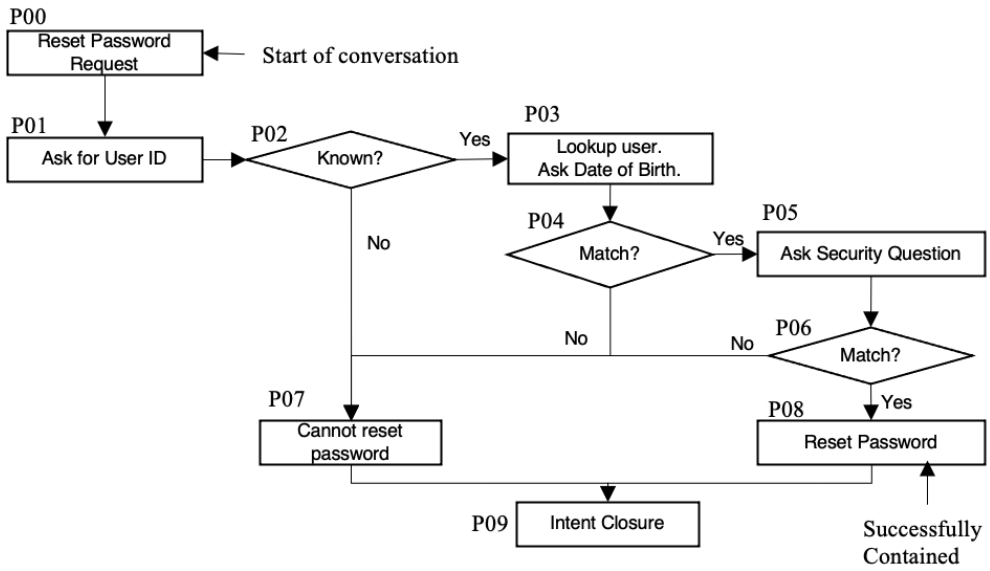


Figure 2 FICTITIOUS INC's reset password conversational flow. Any conversation that visits P00 is counted as a password reset conversation. Only conversations that include P08 are successfully contained.

A password reset conversation always starts with dialog P00 and P01. After that, the password reset flow has only one path to success. The successful path includes dialog nodes P00, P01, P03, P05, and P08. These nodes form a conversion funnel which is shown in Figure 3. Every conversation that includes P03 must necessarily include P01, but some conversations that include P01 will not include P03. A conversation that includes P01 but not P03 will have "drop-off" at P01. By measuring the drop-off between P01, P03, P05, and P08, FICTITIOUS INC can narrow in on why password reset conversations fail to complete.

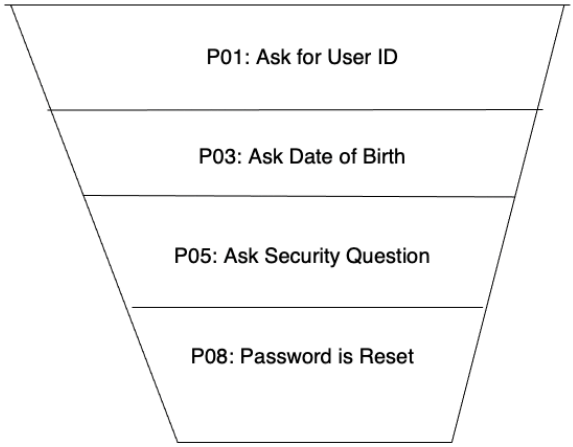


Figure 3 A successful password reset flow, visualized as a funnel. A conversation that completes each step in this funnel is successfully contained.

FICTITIOUS INC can analyze their password reset process flow via a conversion funnel by analyzing their virtual assistant logs and counting how many times each dialog node is invoked. Then, they can compute the drop-off in each step of the funnel. This high-level analysis will illuminate what parts of the process flow require further analysis. The parts causing the most drop-off should be improved first.

How can you run log analysis in your specific virtual assistant platform?

There are multiple ways to perform the analysis done in this section. The specific steps vary by virtual assistant platform. For instance, your virtual assistant platform may make it easy to find conversations that include one dialog node but not another.

The techniques in this chapter are purposely generic, even if somewhat inefficient. If your virtual assistant provider does not include analytic capabilities, you will likely want to build the analyses described in this section.

FICTITIOUS INC's password reset conversion funnel metrics can be found in Table 3.

Table 3 Conversion funnel for FICTITIOUS INC's password reset dialog flow. This analysis shows a steep drop-off after asking for the user ID and the security question.

Dialog Node	Conversations including this dialog node	Conversations that don't include this dialog node	Drop-off from previous node %
P01: What's your user ID?	200	0	0%
P03: What's your date	135	65	33%

of birth?			
P05: What's your security question?	130	5	4%
P08: Password reset complete	80	50	38%

The conversion funnel tells FICTITIOUS INC that one-third of password reset flow conversations included the question "What's your user ID?" but do not include the question "What's your date of birth?". The "What's your user ID?" question has a 33% drop-off rate. It's also the largest source of drop-offs, causing containment failure on 65 total conversations. The entire conversion funnel can be visualized as in Figure 4.

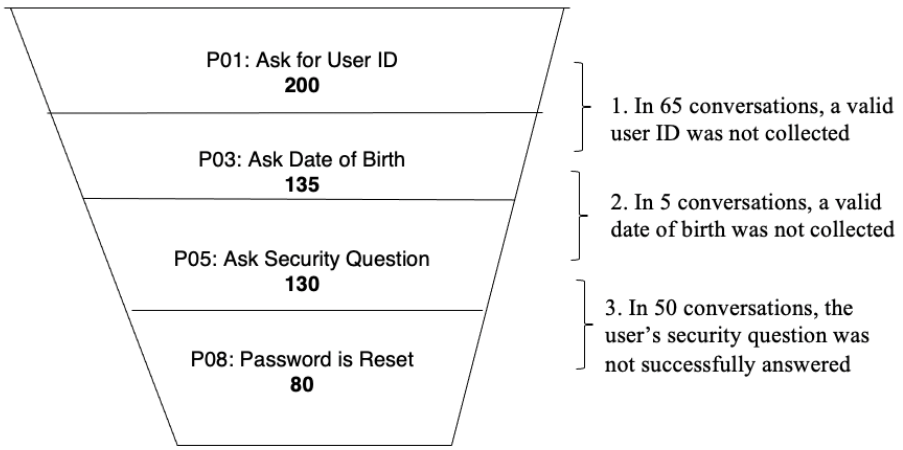


Figure 4 FICTITIOUS INC's password reset flow conversion funnel annotated with the number of conversations containing each step of the dialog. P01 and P05 cause most of the total drop-off.

The severe drop-offs between P01 and P03, as well as P05 and P08, are both detrimental to FICTITIOUS INC's successful containment metric. The P05 to P08 drop-off is more several from a relativistic perspective (38% vs 33%), but the P01 to P03 drop-off affects more conversations in total. FICTITIOUS INC should first focus on the P01 to P03 drop-off.

ANALYZING THE FIRST SOURCE OF DROP-OFF IN THE FIRST INTENT

The first detailed analysis for the P01 to P03 drop-off is to find out what users are saying to the assistant between P01 and P03. Depending on their virtual assistant platform, FICTITIOUS INC can query for:

- What users say immediately **after** P01
- What users say immediately **before** P03

This query will tell FICTITIOUS INC what users are saying in response to the "What is your User ID?" question. FICTITIOUS INC can inspect a small sample of these responses, perhaps 10 or 20, in their initial investigation. The query results are shown in Table 4. All valid FICTITIOUS INC user IDs follow the same format: four to twelve alphabetic characters followed by one to three numeric characters. Any other user ID string is invalid. Before reading ahead, see if you can classify the response patterns.

- afreed1
- don't know, that's why i called
- ebrown5
- fjones8
- hgarcia3
- I don't know it
- I'm not sure
- jdoe3
- mhill14
- nmiller
- no idea
- pjohnson4
- pdavis18
- tsmith
- vwilliams4

The analysis of these responses is shown in Figure 5. The analysis surfaces several patterns in the response utterances. The expected response to P01 is a valid user ID consisting of four-to-twelve letters followed by one to three numbers. But that is not what users always provide!

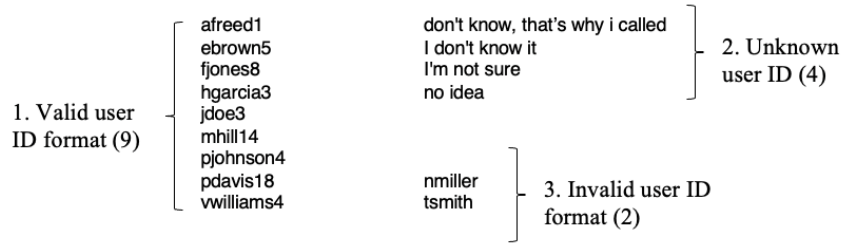


Figure 5 Patterns in user utterances given in response to FICTITIOUS INC's question P01: "What is your User ID?"

FICTITIOUS INC can transform these patterns into actionable insights that will improve successful containment.

- **Insight #1**: **Many users don't know their user ID**. FICTITIOUS INC could build an intent for `#i_dont_know`. When a user is asked for their ID and responds with

`#i_dont_know`, the assistant could provide the user with instructions on how to find their user ID. Or the assistant could be programmed to validate the user another way.

- **Insight #2**: **Many users provide their user ID incorrectly**. This may be because they don't actually know their user ID, or they may have entered it incorrectly[1]. These users could be given another chance to enter their ID or guidance on what a valid user ID looks like.

Couldn't FICTITIOUS INC have found these insights during their design phase?

The design phase will never be perfect. Improving an assistant based on real-world feedback can be better than building an over-engineered assistant with features that nobody actually needs.

There may be additional problems occurring at this point in the dialog, but with this quick analysis, FICTITIOUS INC has already found two unique error patterns causing password reset conversations to fail. These two patterns will address most of the 65 drop-offs caused between P01 and P03. FICTITIOUS INC can assign resolution of these patterns to their design and development teams and move onto further analysis. Let's continue FICTITIOUS INC's password reset analysis at the other major source of drop-offs.

ANALYZING THE SECOND SOURCE OF DROP-OFF IN THE FIRST INTENT

Fifty of FICTITIOUS INC's 120 drop-offs in password reset occurred immediately after P05: asking the user's security question. The user has gotten tantalizingly close to P08 which actually resets the password– they are only one step away! FICTITIOUS INC can modify their trusty query depending on their virtual assistant platform to find:

- What users say immediately **after** P05
- What users say immediately **before** P08

As with the prior analysis, a sample of query results should suffice to identify patterns. This analysis is somewhat more difficult if the user's specific security question is not known, however, in a quick analysis the questions are not necessarily needed. The alphabetized query results are shown below. Just like the prior exercise, try to find the patterns before reading ahead.

- 02/04/2018
- Anderson
- FLUFFY
- Jones
- Lopez
- new york
- null
- null
- null

[1] In the case of voice assistants, the assistant may not have heard the user correctly. One possible resolution is to improve the assistant's voice training on user IDs.

- orange
- PEnnsylvania
- purple
- skiing
- trains
- Volleyball

An analysis of these utterances is shown in Figure 6. Similar to the previous analysis, three patterns are present in the data.

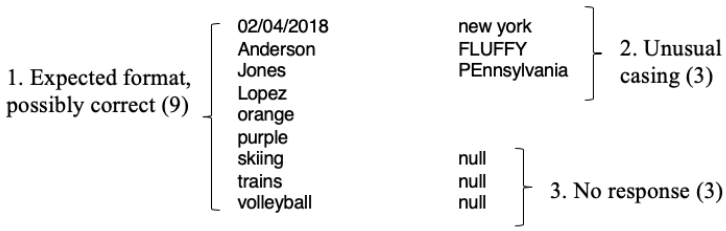


Figure 6 Patterns in user utterances given in response to FICTITIOUS INC's question P05: Answer your security question.

Two insights jump out from this data:

- **Insight #3**: **Several responses are** `null`[2]. This indicates that the user ended the conversation without answering the question – closing the chat window or hanging up the phone. This could happen because the user didn't know the answer or because the user was frustrated with a process that took longer than expected. FICTITIOUS INC could alter the dialog for this question to indicate that it's the final step. Or FICTITIOUS INC could investigate an alternative verification option for these users.
- **Insight #4**: **Responses include unusual casing**. The responses "FLUFFY", "new york", and "PEnnsylvania" seem like possible correct answers, but with questionable upper- and lower-casing. FICTITIOUS INC should investigate if the response to this question needs to be case-sensitive.

ANALYZING THE THIRD SOURCE OF DROP-OFF IN THE FIRST INTENT

FICTITIOUS INC's password reset flow has a third place where drop-offs occur. 5 out of the total 120 drop-offs happen between P03 and P05, when the user is asked for their date of birth. What analysis should FICTITIOUS INC do about these drop-offs in their initial assessment of their assistant?

FICTITIOUS INC probably does not need to do anything, for now. There are several good reasons for this:

- **Data-driven approach**: The five drop-offs are a small fraction of the 120 total in the password reset flow. FICTITIOUS INC already has four solid leads to solve the other 115 drop-offs. (Existing leads: unknown user IDs, invalid user IDs, no security

[2]The specific value will vary by conversational platform. You may also see "conversationEnd", "hangup" or even the empty string "".

question response, and unusual casing of security question response.)

- **Intuitive sense**: There are three questions in the password reset flow, and it makes intuitive sense that the "date of birth question" would have the least number of problems. We expect less people to forget their date of birth than their user ID or the answer to their security question. The low number of drop-offs at date of birth confirms this.
- **Opportunity cost**: The time and energy FICTITIOUS INC focuses on this small problem is time and energy they can't spend on another problem.
- **Prioritize bigger problems first**: FICTITIOUS INC's biggest containment problem came from 120 uncontained password reset conversations. Their second biggest containment problem was appointments with 35 uncontained conversations. The appointments flow should be analyzed before the date of birth issue.
- **Alignment with success metrics and business value**: There is more business value from focusing on the appointments flow first. FICTITIOUS INC doesn't get any credit for analyzing 100% of a process flow. Their success comes from improving successful containment.
- **The Golden Rule of prioritization:** "Not now" doesn't mean "not ever".

Always keep the business success metric in mind when analyzing and improving on your virtual assistant. Depending on the size of FICTITIOUS INC's development team, the analysis done so far may already be enough to keep them busy improving the assistant. But if they have the capacity, they can continue analysis. FICTITIOUS INC's success metric of successful containment dictates that they should analyze the appointments containment issue next.

Keep your business success metric in mind when analyzing and improving on your virtual assistant.

10.1.2 Inspecting other process flows for containment problems

FICTITIOUS INC's second major source of uncontained conversations was in the `#appointments` flow. Users wanting an appointment with FICTITIOUS INC will need to provide a store location, a date, and a time. They can provide this information all at once or the assistant can prompt them for the information one piece at a time. The `#appointments` flow is depicted in Figure 5.

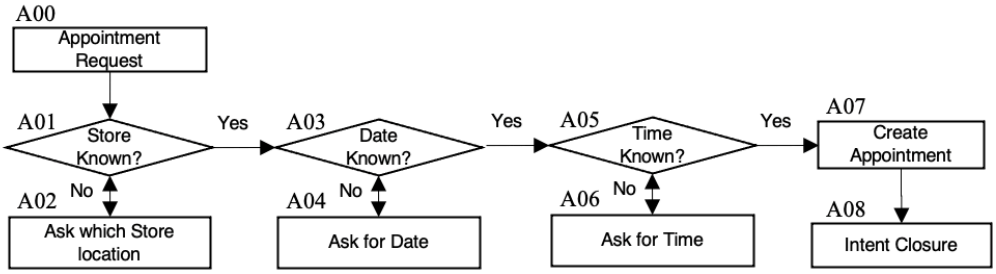


Figure 7 FICTITIOUS INC's appointment process flow. An appointment requires a store location, date, and time. The user may provide these all at once or one at a time.

This process flow presents an interesting analysis challenge due to the inclusion of the optional nodes. Every successful `#appointments` conversation must include A00 and A07, but it's unclear how many should include the optional nodes A02, A04, and A06. Thus, counting conversations including the optional nodes does not have immediate value. FICTITIOUS INC should start with the simple conversion funnel in Figure 6.

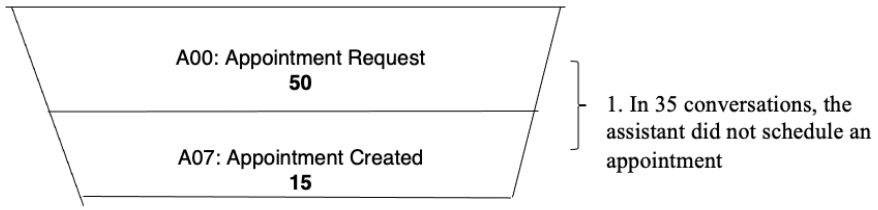


Figure 8 Conversion funnel for FICTITIOUS INC's appointment process flow. Because A02, A04, and A06 are all optional steps, they are not included.

With only two guaranteed steps in the process, the analysis options are limited. The optional questions throw a wrinkle into a purely funnel-based analysis. Based on our familiar analysis method FICTITIOUS INC can look for either:

- What users say immediately **after** A00
- What users say immediately **before** A07

FICTITIOUS INC will start by analyzing utterances immediately after A00. A sampling of those utterances is shown below. Before reading past the list, what conclusions can you make for FICTITIOUS INC? What is causing the containment problem for `#appointments`?

- at Maple
- elm
- Elm please
- no i don't want an appointment
- not what I want

- today
- tuesday at 9pm
- what time are you open?
- what??
- why do i need an appointment for this?

These utterances are analyzed in Figure 9, which shows two clear patterns in the data.

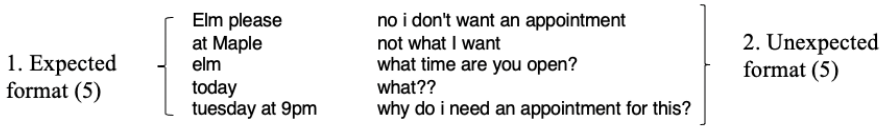


Figure 9 Sample utterances received by FICTITIOUS INC after dialog A00, where the user is asked to provide details on the location, date, and/or time for an appointment.

The utterances fall into two patterns:

- **Happy path**: Five of the utterances are clearly responding to A02 (where do you need the appointment) or A04 (what day would you like the appointment). There is no immediately obvious reason why these conversations would fail to complete successfully.
- **Unhappy path**: Four of the utterances are clearly not enthusiastic about setting up an appointment.
- (The final utterance "what time are you open" could fit into either pattern. Without additional context, it's too hard to tell.)

The key insight from this analysis is

- **Insight #5**: **Users are surprised when the assistant creates an appointment.** Users appear to enter the `#appointments` flow when that is not their actual intent.

FICTITIOUS INC's primary problem with password resets was that users didn't know how to answer the questions in the process flow. But FICTITIOUS INC's primary problem with appointments looks like users getting a process flow they didn't want. FICTITIOUS INC can confirm this hypothesis by exploring the user utterances received immediately before A00. These utterances are shown below.

- Can I make an appointment at Maple?
- Can you ship to my house?
- Consults available at 9pm today?
- Do I need an appointment for login problem?
- Help me set up a consultation
- I need to talk to someone at Elm on Friday.
- Schedule appointment
- What's the schedule for the sales promotion?
- When can I make a return at Maple?
- When can I shop in-person at your store?

The utterances are analyzed in Figure 10.

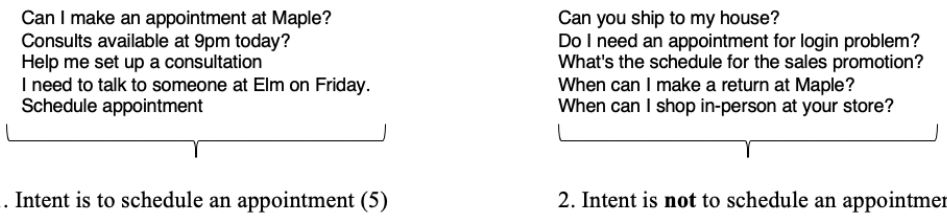


Figure 10 Sample utterances received by FICTITIOUS INC before dialog A00. The virtual assistant is currently initiating the #appointments flow after each one of these utterances.

The hypothesis holds. Several of these utterances are not asking to set up an appointment. FICTITIOUS INC's `#appointments` intent is capturing too many utterances that are not related to appointments. This analysis makes some intuitive sense – we would not expect users to have chronic problems with the location, date, or time questions. While you cannot yet rule out that some conversations fail at these questions, the over-selection of the `#appointments` intent is the primary problem in the appointments flow. FICTITIOUS INC's assistant is inaccurate when it comes to identifying the `#appointments` intent.

FICTITIOUS INC's five analysis insights and suggested next steps are summarized in Table 4.

Table 4 Summarized analysis and next steps for FICTITIOUS INC. Each of these insights directly affects the "successful containment" success metric.

Insight	Next steps
Many users don't know their user ID.	Design a response that helps users find their user ID.
Many users provide their user ID incorrectly.	Design a response that gives users a second chance, possibly with additional advice.
Several users don't attempt to answer the security question.	Investigate an alternate method of validating the user.
Security question responses include unusual casing	Consider a case-insensitive evaluation of the answer.
Users are surprised when the assistant creates an appointment.	Improve the assistant's training around the `#appointments` intent.

FICTITIOUS INC has three more intents we have not analyzed for containment problems. Each of `#employment_inquiry`, `#store_hours`, and `#store_location` had high containment in the very first analysis. These intents each use a single question-and-answer response, so it is difficult for a conversation that starts one of those intents not to finish it. (If there is just a single response, the user would have to close the assistant while the assistant was

generating the response for the conversation to not be contained.) Users are reaching the end of the conversations for these intents, which is enough to declare the conversations contained. But is the assistant recognizing these intents correctly?

10.2 Analyzing the classifier to predict future containment problems

If FICTITIOUS INC's assistant identifies the wrong intent, it may still "contain" the conversation yet not satisfy the user. (This is why it is difficult to evaluate an assistant using only one success metric.) Users who get responses based on the wrong intent may be "contained" today but are unlikely to return to the assistant in the future. Therefore an inaccurate assistant can reduce the total volume of contained conversations. This harms FICTITIOUS INC's bottom line – the user may select a more expensive resolution option (calling a human agent) or may stop being a FICTITIOUS INC customer altogether!

FICTITIOUS INC should analyze the assistant's classifications to see how often it predicts the wrong intent. There are two different analyses FICTITIOUS INC can run:

- **Representative baseline**: Take a random sampling of utterances and see how they are classified. The variation in this sample will ensure an accurate evaluation of the assistant.
- **Find gaps in the training data**: Any utterance for which the assistant predicts an incorrect intent is a potential candidate for improving the assistant, especially if the prediction was made with high confidence. Similarly, any utterance correctly predicted but with low confidence sits in a similar gap. FICTITIOUS INC should collect some of both types of utterances for use in improving the system.

10.2.1 Representative baseline

Before making any changes to the classifier, FICTITIOUS INC should understand how well it already works. Now that their assistant is in production, they have the perfect data set to test their assistant's accuracy. Before they were in production, they may have run a blind test on data they expected to see in production, but now they have the real thing. FICTITIOUS INC needs to gather some production data for a blind test. This test will give a baseline for how accurate the assistant was before changes were made.

The baseline needs to include a representative variety of data. The easiest way to accomplish this is to take a random sampling of all utterances that were used to start a conversation[3]. FICTITIOUS INC can take this data and build a blind test set. A sampling of conversation-starting utterances for FICTITIOUS INC's assistant is shown in Table 8.

The predictions in Table 8 look mostly correct, but FICTITIOUS INC shouldn't eye-ball this. They should create a **blind test set**. A blind test set is a spreadsheet with two columns: user utterances and the *actual* intent for those utterances. Table 8 includes the utterances with their *predicted* intent. We've seen multiple times that the assistant makes some mistakes – some predicted intents do not match the actual intent.

[3] A random sampling of first user utterances is "good enough". This sampling will not include any retries, or any intents attempted after the first. The analytic query to get "first user utterances" is easy to write and jump-starts analysis. Don't let the perfect be the enemy of the good!

Table 5 Sample conversation-opening utterances received by FICTITIOUS INC's assistant. FICTITIOUS INC needs to include the utterances and their actual (not predicted) intent in a blind test set.

User utterance	Assistant's prediction	Assistant's confidence
Can you ship to my house?	#appointments	70%
I want to apply for a job	#employment_inquiry	81%
I forgot my password	#reset_password	100%
Reset my password	#reset_password	100%
Need my password reset	#reset_password	93%
I can't login	#reset_password	82%
What are your hours on Wednesday?	#store_hours	99%
How late are you open tonight?	#store_hours	98%
Where is the nearest store to me?	#store_location	99%
I need driving directions to Maple	#store_location	72%

FICTITIOUS INC can take multiple approaches to convert Table 8 into a blind set: fully unassisted, partly assisted, or fully assisted. Let's explore these approaches.

FULLY UNASSISTED

In this approach, FICTITIOUS INC takes the utterances from Table 8 and copies them into a new spreadsheet with one column. They have a subject matter expert fill in a second column, containing the actual intent for each utterance. Ideally the SME should have the list of FICTITIOUS INC's intents and the definition of those intents, but no other guidance. This approach is demonstrated in Figure 7.

User utterance	Intent
Can you ship to my house?	
I want to apply for a job	
I forgot my password	
Reset my password	
Need my password reset	
I can't login	
What are your hours on Wednesday?	
How late are you open tonight?	
Where is the nearest store to me?	
I need driving directions to Maple	

The person filling in the intent column needs to do all the work.

Figure 11 Fully unassisted labeling of production user utterances for a blind test set. This table does not include any prediction data from the assistant. The person providing the intent has a completely unbiased view, but they have a lot of work to do.

The biggest argument in favor of the fully assisted approach is that the person providing the intents is not biased by the assistant's predictions. This should ensure high-quality intent labeling, but it comes at a cost. This is the slowest of the three approaches. If FICTITIOUS INC uses this approach, they are likely to build a smaller blind test set due to time considerations alone.

Let's examine the partly assisted option.

PARTLY ASSISTED

In this approach, FICTITIOUS INC transforms Table 8. First, all predictions below a certain confidence threshold are removed. Then, the confidence column is removed. A subject matter expert only needs to fill in the remaining blanks in the intent column. This approach is demonstrated in Figure 8.

User utterance	Intent
Can you ship to my house?	
I want to apply for a job	
I forgot my password	#reset_password
Reset my password	#reset_password
Need my password reset	
I can't login	
What are your hours on Wednesday?	#store_hours
How late are you open tonight?	
Where is the nearest store to me?	
I need driving directions to Maple	

Predictions above the confidence threshold are assumed correct.

The subject matter expert fills in all remaining blanks.

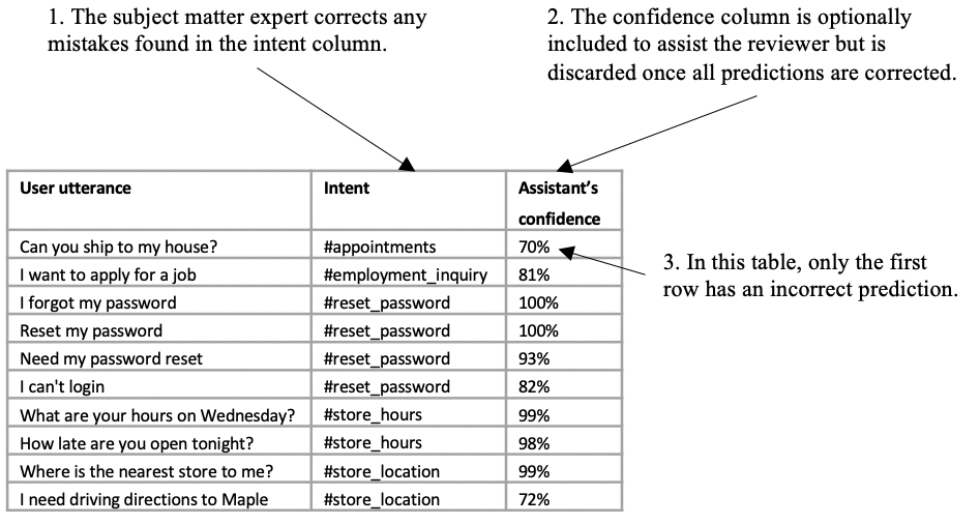
Figure 12 Partially assisted option for building FICTITIOUS INC's blind test data set. In this example, the original predictions from the assistant with 99% or higher confidence are assumed to be correct.

The best part of this approach is that it is faster. There's no need to have utterances reviewed if the assistant handled them with 100% confidence. A variation on this approach is to trust the assistant's predictions if they are above some non-100% confidence threshold. The farther this threshold is lowered, the less work for a subject matter expert, but there's a higher chance of mistakes slipping by. Since FICTITIOUS INC is doing this exercise for the first time, I'd recommend using at least 99% as the threshold. Once they are more in tune with their virtual assistant's performance, they can consider a lower threshold.

A partly assisted approach is my preferred method of adding production utterances for a blind test set. There's one more approach FICTITIOUS INC can consider, and it's even faster.

FULLY ASSISTED

In the fully assisted approach, the subject matter expert is given Table 8 as-is and told to correct any mistakes in the prediction column. If the assistant's prediction is correct, there is no work for the expert. This is demonstrated in Figure 9.



User utterance	Intent	Assistant's confidence
Can you ship to my house?	#appointments	70%
I want to apply for a job	#employment_inquiry	81%
I forgot my password	#reset_password	100%
Reset my password	#reset_password	100%
Need my password reset	#reset_password	93%
I can't login	#reset_password	82%
What are your hours on Wednesday?	#store_hours	99%
How late are you open tonight?	#store_hours	98%
Where is the nearest store to me?	#store_location	99%
I need driving directions to Maple	#store_location	72%

Figure 13 Fully assisted approach. The subject matter expert is shown all of the assistant's predictions and only updates the incorrect predictions. This is the fastest approach but biases the expert with the assistant's opinion.

This is the fastest way to produce a blind test set from production logs. For the subject matter expert, correcting mistakes is easier than producing new answers. However, this approach has the highest risk of introducing bias into the test data. For a given utterance, the assistant may make a "good" prediction even though a "better" prediction is possible. By showing the expert the "good" prediction, they may be biased to accept it. If the expert had to generate their own prediction, they choose the better prediction.

FICTITIOUS INC can use any of these approaches to generate their blind test data set: a fully unassisted approach, partly assisted, or fully assisted. The trade-offs are shown in Figure 10. For FICTITIOUS INC's first improvement cycle I recommend the partly assisted approach.

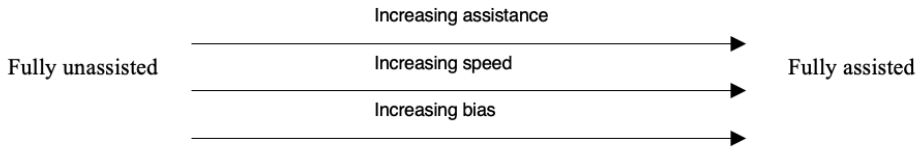


Figure 14 The trade-offs in the approaches used to generate test data from production logs.

No matter which approach FICTITIOUS INC uses to create a blind test set, they are ready to start looking at where the assistant can be more accurate at identifying intents. One great source they can use is low-confidence predictions.

10.2.2 Finding Gaps in the training data

FICTITIOUS INC has now gathered a blind test set which they can use to understand how accurate their assistant is. They need to know how accurate the assistant is before they start making changes to improve the assistant. They know the assistant misclassifies some utterances. They need to gather some misclassified utterances to help improve their assistant. They first need to find these utterances and then can work on testing and improving the classification of these utterances.

There are three ways FICTITIOUS INC can find training gaps:

- **Find utterances with low confidence intent predictions**. The assistant assigns a confidence value to each prediction it makes. We can define low confidence as a confidence value below a threshold, for example below 0.5. Predictions with low confidence are generally of lower quality than those with higher confidence[4]. This set of utterances is likely to identify "problem spots" in the assistant's training.
- **Work backward from containment problems**. When FICTITIOUS INC analyzed the `#appointments` process flow, they found several conversations that were not contained because the wrong intent was identified.
- **Find incorrect predictions that had high confidence**. Where the assistant is both confident and wrong, there is a significant training gap. These gaps are the rarest and hardest to find.

FIND LOW CONFIDENCE INTENT PREDICTIONS

Low confidence predictions are a great place to find gaps in training. A low confidence prediction means the assistant was not very sure which intent to select. Because many low-

[4]The assistant can still be wrong with high confidence. Since a human is required to tell if the assistant is right or wrong, we use a "low confidence filter" to reduce the number of utterances a human has to review.

confidence predictions will be wrong, assistants are often configured not to select an intent with if the confidence is low and will instead ask the user to restate their question. An assistant that does not take action on intents with low confidence will prevent false positives (responding with a wrong intent) but will also suffer from false negatives (not responding, when it could have).

A sampling of low-confidence predictions from FICTITIOUS INC's assistant is listed in Table 9. Some predictions are correct: "when and where can I schedule time with an expert" is indeed an `#appointments` intent. Some predictions are incorrect: "I need help finding you from my house" is `#store_location`, not `#appointments`.

Table 6 Low-confidence predictions made by FICTITIOUS INC's assistant. For each of these utterances, the user was prompted to restate their question. The system does not use any intent predicted with less than 0.5 confidence.

Utterance	Intent	Confidence
where and when can i schedule time with an expert	`#appointments`	0.43
i need help finding you from my house	`#appointments`	0.37
time to talk in person	`#appointments`	0.33
what roles do you have available?	`#employment_inquiry`	0.27
log in problem	`#reset_password`	0.44
Open on Valentine's day?	`#store_hours`	0.42
Valentine's day hours	`#store_hours`	0.31
gps directions to you	`#store_location`	0.3
where do you deliver?	`#store_location`	0.3
do you ship to store or my house	`#store_location`	0.23

Low-confidence predictions are also a great place to find new intents. Table 9 includes two utterances where the user is asking about delivery options. Currently, FICTITIOUS INC's `#store_location` intent is matching to these utterances. FICTITIOUS INC can consider adding a `#delivery_options` intent if it turns out to be a common user need.

Figure 11 annotates the sample of FICTITIOUS INC low-confidence predictions.

User utterance	Intent	Assistant's confidence
I need help finding you from my house	#appointments	37%
where do you deliver?	#store_location	30%
do you ship to store or my house	#store_location	23%
where and when can i schedule time with an expert	#appointments	43%
time to talk in person	#appointments	31%
what roles do you have available?	#employment_inquiry	27%
log in problem	#reset_password	44%
Open on Valentine's day?	#store_hours	42%
what time can i pickup my package	#store_hours	38%
gps directions to you	#store_location	30%

1. Confusion between #appointments and #store_location
2. These suggest a new intent: #delivery_options
3. Correct intent, low confidence. Assistant is not strong here.

Figure 15 Sample of FICTITIOUS INC low-confidence predictions.

Based on these findings, FICTITIOUS INC should start by improving the `#store_location` training data. They have unambiguous false-negative utterances like "gps directions to you" that could be added to the training for `#store_location`. They also have unambiguous false-positive utterances like "where do you deliver" that should go to a different intent (creating `#delivery_options`) or should be "negative examples"[5] for `#store_location`. The assistant also is slightly weak at identifying `#store_hours` requests specific to holidays. This could be fixed by updating the `#store_hours` training examples.

Not every low confidence prediction indicates a gap in training. The utterance "tell me a joke" does not match any intent in FICTITIOUS INC's assistant. For this utterance, the system responds with a message like "I'm sorry, I did not understand." That is a perfectly fine response! FICTITIOUS INC is trying to improve their containment, not to make their assistant sentient. A set of low confidence predictions will contain some useful patterns as well as some noise.

Evaluating low confidence predictions is a good analysis because it is fast. These predictions can be found with a generic query and are likely to illuminate training gaps. FICTITIOUS INC can also search for possible training gaps by looking at uncontained calls.

WORK BACKWARD FROM CONTAINMENT PROBLEMS

Process flows with poor containment can also be mined for training data gaps. In FICTITIOUS INC's analysis of the `#appointments` flow, they found that many conversations started the appointments process flow when that's not what the user wanted! An example is shown in Figure 12.

[5] Most training data is a list of positive examples: "Directions to your store" means #store_location. Some virtual assistant platforms let you create negative examples, or counterexamples, like "Where do you deliver" does *not* mean #store_location.

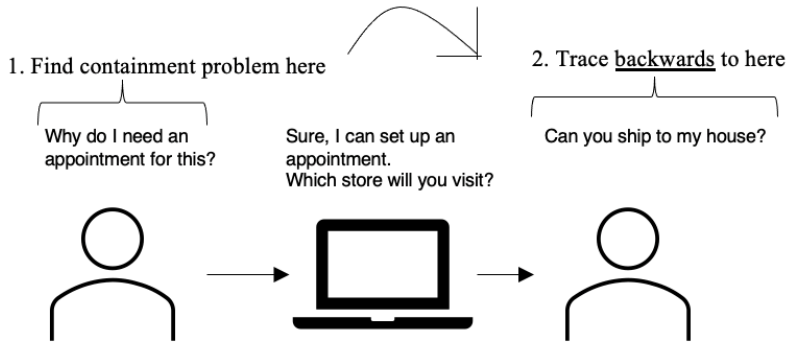


Figure 16 FICTITIOUS INC can find conversations that were not contained, then walk backward to find utterances with poor classification.

in Figure 12 FICTITIOUS INC found an uncontained appointments conversation that ended with "Why do I need an appointment for this?" They could walk backward in that conversation to find what the user initially said: "Can you ship to my house?" That utterance was misclassified as `#appointments`.

FICTITIOUS INC has previously analyzed the `#appointments` process flow and found the uncontained conversations. They can walk backward in each of these conversations to find utterances that were misclassified as `#appointments`, and these utterances will also help identify error patterns and gaps in the FICTITIOUS INC assistant's training.

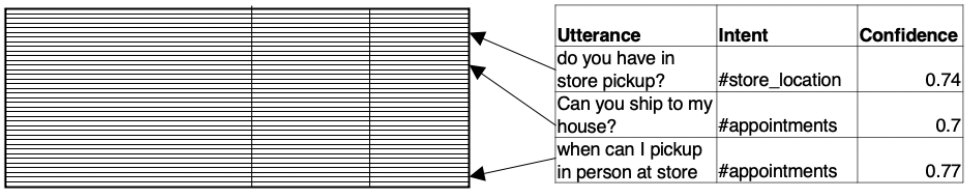
Uncontained conversations are a good candidate source for finding problems in training. Not every uncontained conversation will start with a classification error, but many will. Since the definition of successful containment may vary by intent, it can be more work to query and walk back through uncontained conversations instead of querying for low confidence predictions. Still, looking at uncontained conversations is a nice complement to the data found in low confidence predictions. FICTITIOUS INC can save time in this first analysis by first focusing on process flows with poor containment. They can review the other process flows in future analysis cycles if needed.

There's one more source of potential training gaps: high-confidence, but incorrect, predictions.

FIND HIGH-CONFIDENCE INCORRECT PREDICTIONS

FICTITIOUS INC would like to find all of the times their assistant made the wrong prediction, but with high confidence. These errors are particularly hard for the assistant to deal with. It makes sense to ask the user to restate an utterance if the assistant isn't sure what the intent is. That mechanism acts like self-correction and keeps the assistant from going down the wrong path too often. But if the assistant identifies the wrong intent with high confidence, it has no mechanism to address that at runtime. When the wrong intent is predicted with high confidence, there may be a serious gap in the training.

Figure 13 shows FICTITIOUS INC's analysis of the assistant's high confidence but incorrect predictions.



Utterance	Intent	Confidence
do you have in store pickup?	#store_location	0.74
Can you ship to my house?	#appointments	0.7
when can I pickup in person at store	#appointments	0.77

Figure 17 Finding high-confidence but incorrect predictions in FICTITIOUS INC's assistant can be like finding a needle in a haystack. Only three utterances out of five hundred matched these criteria!

Unfortunately, there is no simple way to find these utterances. FICTITIOUS INC's assistant handled hundreds of conversations. Most of these conversations started with a high-confidence prediction, and most of those predictions were correct. Only a painstaking review of the conversation-opening utterances could surface these problematic utterances. High-confidence predictions are easy to find; narrowing down those predictions to the incorrect ones takes time.

Fortunately, these utterances and the patterns they represent can be found in the other analytic methods. Figure 13 surfaces two error patterns:

- There's a need for a `#delivery_options` intent.
- The `#appointments` intent is prone to false positives.

FICTITIOUS INC was able to find both of these insights in other analyses. This shows that there are multiple ways to find the same insights. This also demonstrates a diminishing return to analysis. The first analyses were relatively quick to perform. This last analysis took much longer to execute but did not turn up anything that wasn't found in a simpler analysis.

FICTITIOUS INC's analysis process started with them building a baseline of their accuracy by building a blind test set and testing their assistant against it. Next, they identified specific areas for improvement. They can now use this new data to do additional training and testing of their classifier using the same process they learned earlier in this book.

> **"In God we trust, all others must bring data."**
>
> **W. Edwards Deming**

10.3 When and why to improve your assistant

This chapter followed FICTITIOUS INC's first analysis of their assistant after it went to production. Through this analysis, they found several surprising shortcomings of their assistant and formed an action plan to improve the assistant. The analysis was done quickly with the intent of identifying the most severe problems first. Implicit in this analysis was that there would be multiple cycles of analysis and improvement.

FICTITIOUS INC can fairly ask: how many improvement cycles will be needed? How long will those improvement cycles take? There's not a set answer, the important thing is that virtual assistants are not one-and-done – they need frequent improvement. Consider giving

your assistant a thoughtful analysis every three months. A more frequent analysis is good early in an assistant's life, and stable assistants may need less analysis and revision. There are several good reasons why continual analysis and improvement of your assistant is a good idea:

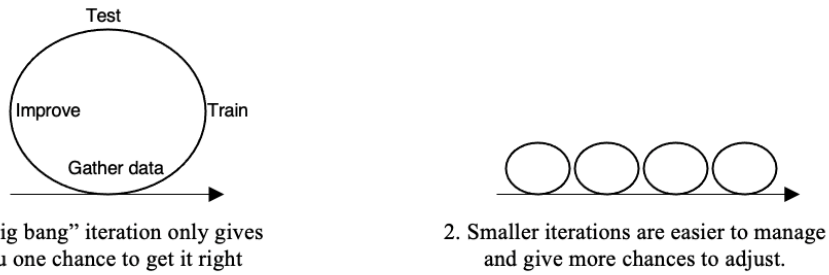
- You can't fix everything at once
- Users will not always react the way you expect them to
- User needs will change

10.3.1 You can't fix everything at once

FICTITIOUS INC's first analysis of their assistant surfaced several insights that could improve the assistant. Their multi-faceted analysis found the most severe problems in their assistant but there's no reason to believe it found all of the problems. Though FICTITIOUS INC could continue their analysis and find even more problems, the analysis will have diminishing returns.

FICTITIOUS INC's goal is to increase successful containment and their analysis has found the biggest sources of uncontained conversations. Fixing the most severe containment problems directly aligns with FICTITIOUS INC's success metrics. Further, time and energy are limited. Every hour spent on analysis is an hour not spent on improvement.

Additionally, some of the containment problems have interdependencies. When FICTITIOUS INC updates their intent training to fix some intents, it may adversely affect other intents. FICTITIOUS INC is best served making small, incremental changes and verifying that they help drive successful containment. Smaller changes are easier to make, to observe, and to roll back if they don't work. Figure 14 demonstrates that shorter improvement cycles accelerate your ability to learn and improve the assistant.



1. "Big bang" iteration only gives you one chance to get it right

2. Smaller iterations are easier to manage and give more chances to adjust.

Figure 18 The smaller the set of improvements, the more improvement cycles you can run. Shorter cycles are less risky and offer more chances for learning.

10.3.2 You can't always predict how users will react

FICTITIOUS INC carefully designed their assistant with the needs of their users in mind. However, as their analysis showed, users did not react as expected. In the password reset flow, users frequently did not know their user ID and often answered their security question

with insensitivity to case. FICTITIOUS INC will need to adjust that process flow based on this user behavior if they want to increase the successful containment of password resets.

> The best-laid plans of mice and men oft go awry.
>
> **Robert Burns**

The more an assistant's design process is centered on user needs, the better the assistant will serve users. But no matter how good the design process is, there will be some unexpected user behavior that's not accounted for in the design. These differences are the "unknown unknowns" – it's impossible to predict what they are, but it's a certainty that they exist.

There's no shame in an imperfect Version 1 design that misses some user needs. Additional versions will inevitably be needed. There's only shame in never improving the design.

10.3.3 User needs will change

FICTITIOUS INC discovered a new user need in their first analysis: users requesting different delivery options. Other user needs will likely emerge over time.

> In theory, there's no difference between theory and practice. In practice, there is.
>
> **Benjamin Brewster**

No matter how well you designed your assistant, users will eventually make new requests. In 2020, COVID-19 rocked the world with a bevy of new request types:

- Healthcare companies saw new requests for COVID symptom checking and processes on testing
- Employers were bombarded with questions on new work-at-home policies
- Food-service companies saw the new phrase "contactless delivery"
- Governments fielded questions about new COVID-19-related policies

Even outside of a global pandemic, user needs will change. New intents will arise, and some existing intents may wane. You may be able to predict the rise of new intents, but as alluded to earlier your ability to predict rising trends is probably not as good as you think it is. A periodic review of your virtual assistant logs will help you discover new intents requested by users and also the way they phrase them. This periodic review can also quantify how frequently the new intents are used.

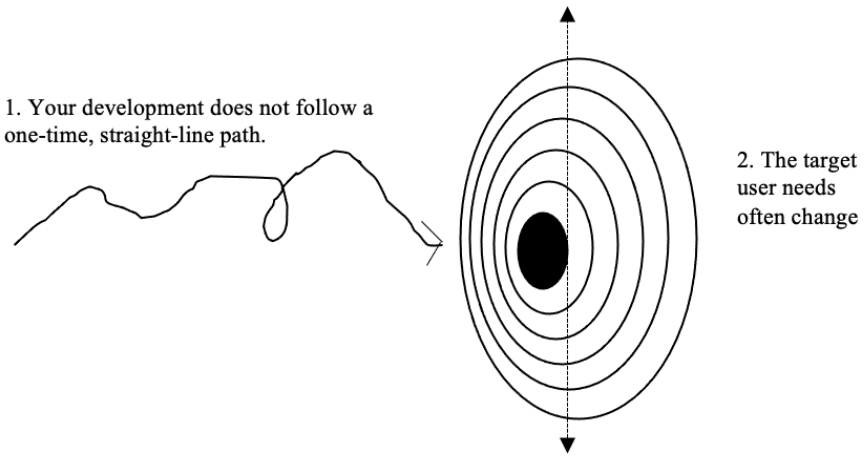


Figure 19 Improving your assistant must be done iteratively because user needs will change! You can't use a straight line to hit a moving target, you must be able to course-correct.

Periodic analysis and improvement of your assistant keeps your virtual assistant up to date with how users interact with it. The improvement process should be done "as often as needed", but a reasonable cadence is to do it every three months.

A reasonable cadence is to analyze and improve your assistant every three months.

10.3.4 Not every problem is technical!

The preceding sections outlined a number of technical techniques for analysis and improvement of a virtual assistant. The technical solutions in this chapter assumed that user behavior cannot be changed. I enjoy a good technical solution, but that assumption about user behavior is not always valid. Sometimes user behavior can be influenced or changed in a way that makes both the users and the virtual assistant more successful.

Users who are not expecting to interact with a virtual assistant may try to immediately "opt out" of the virtual assistant. This involves trying to find a way to escape the virtual assistant and get to a human. This desire is understandable – these users may have had bad experiences with other automated solutions, or they may think a human will get them what they need faster. Or these users may have tried your virtual assistant once before, not known what to do, and decided to never give it a chance again.

FICTITIOUS INC had problems with successfully containing calls about password resets and appointment scheduling. FICTITIOUS INC could proactively reach out to users and give them tips on how to achieve their needs quickly with the virtual assistant. This communication could take many different forms: email, instructions on a website, or a

physical form. Figure 16 is a business card-sized cheat sheet FICTITIOUS INC can send to their users which addresses two of the key challenges in the assistant.

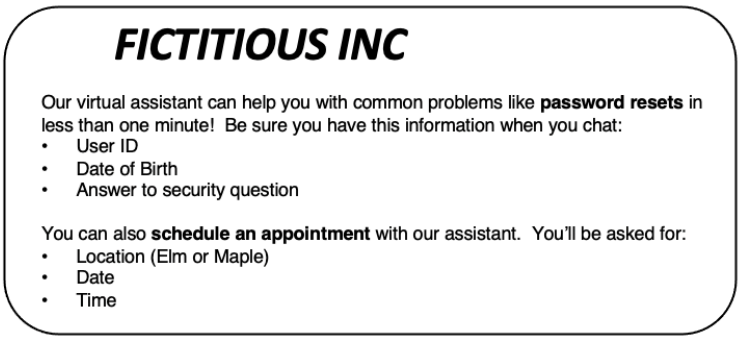


Figure 20 FICTITIOUS INC can create a business card-sized cheat sheet for their virtual assistant users telling them what they can do with the assistant and what information will be expected from them. This will make users more successful in using the assistant.

Figure 16's postcard has several benefits:

- **Speaks directly to user value**: You can get your password reset in *one minute* using the assistant. (It might take 15 minutes to get connected to a human agent to do the same)
- **Sets expectations**: Users know exactly what the assistant needs. Rather than being surprised during the conversation, the user can prepare ahead of time what they need to finish their transaction.
- **It's small and concise**: The business card form factor easily fits inside a wallet or can be taped onto a user's landline or computer monitor or can be placed somewhere else that is easily accessible.
- **It biases user language towards what your assistant already understands**: The users are primed to use the utterances "password reset" or "schedule an appointment". Your assistant will have no problem identifying the intent in these utterances.

Behavior is easier to change in some user groups than others. FICTITIOUS INC's easiest possible user population would be other FICTITIOUS INC employees – the company would have many ways to reach and influence those users. There's usually a way to reach your users, even if it requires some creativity. Influencing user behavior can be cheaper and easier than making purely technical changes.

10.4 Summary

- Analysis and improvement should be tied to the business success metrics
- Analyze the most serious process flow failures to find the most significant areas for improvement

- Assistants need frequent improvement because user needs change in unpredictable ways

11

How to build your own classifier

This chapter covers:

- Constructing a simple text classifier from the ground up
- Appraising and critiquing a classifier based on what it learns from a training cycle
- How each training example affects the classifier's learning
- Why you'd build your own classifier

In previous chapters, we learned about how to design and construct a virtual assistant, and how to make sure it responds appropriately to the user. Virtual assistants use classifiers[1] to extract meaning (as an intent) from a user's natural language input, as shown in Figure 1. The intent detected by the classifier is used by the virtual assistant to determine the best response. Thus, getting the right intent from user input is a key part of a well-functioning virtual assistant.

[1]Some virtual assistant platforms refer to the classifier as the Natural Language Understanding or NLU component.

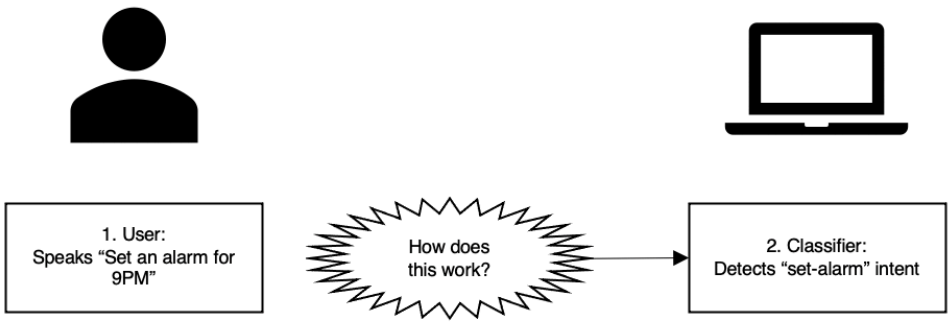


Figure 1 Orientation for this chapter. How does the classifier actually work? How can you make it better?

We have briefly discussed that virtual assistants are trained, but we have not discussed how that training actually works. In this chapter, you will learn how to train your own classifier. We will build a simple text classifier for FICTITIOUS INC's virtual assistant.

If you are not interested in building your own classifier, you can safely skip this chapter. However, if you understand how a classifier is built you will be much more successful training and improving your virtual assistant. Building a custom classifier is most important for classification and routing virtual assistants. Most conversational assistant developers use the classifier provided by their virtual assistant platform.

In the next chapter, I will talk about how the training data for a classifier requires volume, variety, and veracity. The mathematical examples in this chapter will drive home why each of those is required. Specifically, you will see how volume and variety drive the classifier's quality of learning.

Having a good working knowledge of how a classifier actually works will help you make better decisions while training, testing, and improving your classifier. You do not need to work out the calculations by hand nor implement these patterns in code. Having a fundamental understanding of the mathematical underpinnings in the calculation move you from "beginner mode" to "expert mode".

In this chapter, I will demonstrate the simplest possible text classifier. I will start with a classifier that just detects one intent (`#reset_password`). It will take an utterance and decide if it is a "reset password" intent or not a "reset password" intent.

After testing a single intent, I will expand to multiple intents. There are two ways to detect multiple intents: with an array of binary classifiers (each detecting one intent only) or a single all-in-one classifier that can detect all intents. I will compare and contrast those two methods with a set of three intents. For every classifier example, I will show the internal calculations used by the classifier to determine the intent behind a user utterance.

By the end of this chapter, you will be able to assess a set of training data as having enough volume and variety to make good predictions. You will be able to differentiate between the binary classifier and all-in-one approaches and determine which one your

chosen virtual assistant platform uses. You will be ready to train a classifier and have high confidence in its predictive capability.

11.1 Why build your own classifier?

There is a classic build vs buy decision when it comes to classifiers. The classifier is one of the most important pieces of the system, but it can also be the most cryptic. Many virtual assistant providers expose their classifier only through low-code or no-code interfaces to shield end-users from the internal complexity. This makes it easier and faster to start building virtual assistants.

Virtual assistant providers are very good at text classification. They usually have teams of data scientists and machine learning experts dedicated to providing accurate text classifiers that are easy to use. They constantly tune algorithms to work for a variety of use cases and try to minimize the volume of training data required. Because of these algorithms, you can focus on providing good training data, instead of having to provide both training data and an algorithm.

Further, for most conversational assistants, you are unlikely to build a better classification algorithm than what is provided by virtual assistant platforms. I'm reminded of the adage "never build your own database" – even though it does not quite apply here, caution is warranted. But there are still good reasons to build your own classifier.

11.1.1 Classification is a differentiator

The first and perhaps most compelling reason to build your own classifier is if having a better classifier is a differentiating feature of your solution. This is primarily the case if you are not building a conversational assistant. Most virtual assistant providers optimize for the conversational assistant use case and provide very good general-purpose classifiers, and consumers can use intents and entities to drive conversations as shown in Figure 2.

Figure 2 Conversational assistant classification is fairly simple. An utterance generally has one intent and zero or more entities.

If you are building a classification and routing virtual assistant, the user's input is probably not just a single "utterance" but contains several parts. Let's look at a couple of typical use cases:

- **Email**: An email has several fields useful in classification: a sender, a recipient list, a subject line, and a message body, and each may be treated differently. An email may be less interesting if the sender's address is from a certain domain. An email can be more interesting if certain words appear (or do not appear) in the body.
- **Support requests**: A support request usually has a title and a body. Within the body, you may specifically look for a product or operating system names, to route the request to the appropriate party.

- **Documents**: Documents can be classified according to their file name, file type, file size, and overall contents.

When you have data that does not easily fit into a general text classifier, you have three options. First, you could choose a single field to classify against, and use a general text classifier on that field. Second, you could concatenate several fields into a single field, and use a general text classifier on that field. Lastly, you could build a custom classifier. Figure 3 demonstrates the options.

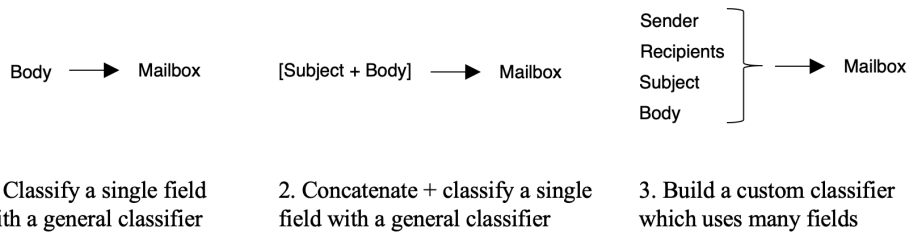


Figure 3 Classification options for email. The first two options can use a general-purpose classifier, the third option requires custom work.

Simple classification options can provide quick value, but you may need a custom classifier depending on how accurate you need to be and how much time you are willing to spend.

11.1.2 Classification is a core competency

If you have a team full of data scientists, you can build a strong classifier that favorably compares to other classifiers on the market. Your data scientists will appreciate the challenge!

Conversely, if data science is not a strong skill for you or your organization, you may not want to build your own classifier. If you are new to data science you will need to invest time and energy into learning at least the basics.

11.1.3 Traceability

If you need full traceability that a closed-source or "black box" algorithm can't provide, you may choose to build your own classifier. There are plenty of open-source options for text classification including Scikit-learn, Natural Language Toolkit (NLTK), SpaCy, TensorFlow, PyTorch, or Keras.

When you have access to the source code of your classifier you know what it is doing every step of the way and you can see all code changes. Closed-source or software-as-a-service solutions can change any code detail at any time, and you will not be aware of what changed. Without access to the source code, your only ability to control changes is to select a specific version of the classification software.

Source code level traceability is especially important for people working in highly regulated environments and industries.

11.1.4 To Learn

Building a classifier for the sheer sake of learning how to do it is a fine motivation. I salute any hobbyists or lifelong learners who want to attempt to build a classifier. It's a rewarding experience and helps you appreciate what machine learning can and can't do.

Additionally, while building your own classifier you will get a very strong sense of how the training process affects the classifier. This will help you as you train and test other classifiers. You will gain an intuitive sense for what will work well, and not work at all, in other classifiers you encounter.

11.1.5 Build or buy?

There are good reasons for either building your own classifier or using an existing classifier. These reasons are summarized in Table 1.

Table 1 Decision matrix for whether or not to build your own classifier

If you are...	Then you should favor....
Building a conversational assistant	Existing classifier
Building a classification and routing assistant	Building a classifier
Not skilled at data science	Existing classifier
Skilled at data science	Building a classifier
Requiring source code control of all components	Building a classifier
Low on free time	Existing classifier
Learning for the sake of learning	Building a classifier

For the rest of the chapter, we assume you are interested in building a classifier.

11.2 Build a simple classifier from first principles

FICTITIOUS INC's Chief Technical Officer is uncomfortable with using a closed-source classifier and wants to build a new classifier for their retail conversational assistant. In the following sections, we will explore how to build a classifier suitable for use in the conversational assistant.

Recall that FICTITIOUS INC's assistant handled over one dozen intents. Their classifier must therefore be able to classify utterances to over one dozen classifications. We will start by first learning how to classify into two classifications: `#reset_password` and "not `#reset_password`". After that we will expand into three classifications: `#reset_password`, `#store_hours`, and `#store_location`. We will then discuss how to expand to as many additional classifications as FICTITIOUS INC needs.

Let's look inside our classifier. Let's take the utterance "Help me reset my password" which has the intent `#reset_password`. How does the classifier know that's a `#reset_password` utterance rather than any other type of utterance?

The first secret to reveal is that machine learning does not operate directly on text. Under the covers, all machine learning algorithms ultimately work on numbers, not text.

Machine learning algorithms work on numbers, not text!

We don't just mean this in a pedantic view (that everything reduces down to zeros and ones), but that the mathematical functions and algorithms have a numerical basis. Figure 4 shows how one might think about a classifier.

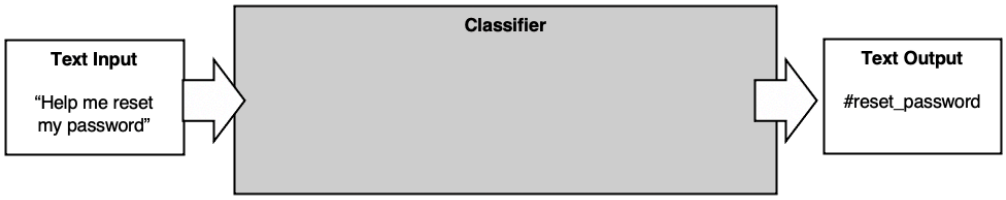


Figure 4 Simplistic mental model of how a text classifier works

We could consider the classifier as a function *f*. It would then be appropriate to think of classification as the following equation:

$$f(input) = prediction$$

A potential numerical view of function f is shown in Figure 5.

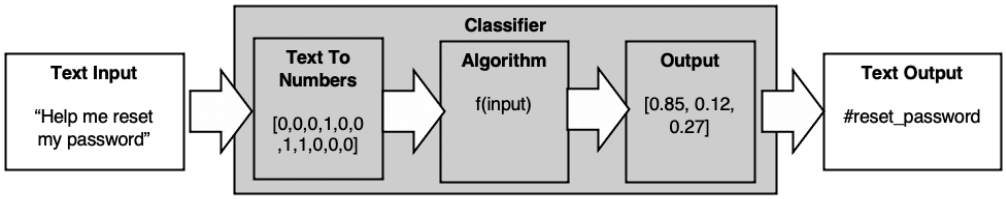


Figure 5 A text classifier uses numbers under the covers. This chapter discusses how those numbers are used.

The conversion from text to numbers is not immediately intuitive. There are multiple methods for converting text to numbers. Let's walk through an example based on a common classification technique. We will build a very simple text classification algorithm and demonstrate how it converts text to numbers, and numbers to text.

11.2.1 The simplest text classifier

The simplest way to convert text into numbers is through a technique called "bag of words". As the name implies, any text string fed into a bag of words classifier is treated as a mere collection of words. There is no meaning attached to any of the words – the order of the words does not matter - the words are simply counted. A simple bag of words algorithm may lower-case input, and remove common words (like "a", "the", etc), but that's about it.

Let's train a bag of words classifier with just two text statements: "Reset my password" and "When does the store open". This is the simplest possible classifier we can train – just two input examples. In fact, we will only train it to recognize one intent (`#reset_password`), by including positive and negative examples.

Table 2 Example conversion of text to a bag of words for a classifier that will detect #reset_password. The order of the words doesn't matter – the classifier may alphabetize the bag of words for convenience. For each bag, we count the number of words in the bag.

Text statement	"Bag" of words	Class label
Reset my password	my:1, password:1, reset:1	Positive
When does the store open	does:1, open:1, store:1, when:1	Negative

Table 2 shows how a bag of words algorithm breaks down some sample text statements. The resulting bag of words now has some numeric representation of the text statements. The bag still has some "words" in it – our text statements are not fully converted to numbers. Table 3 shows how the classifier references each "bag" of words by an array index.

Table 3 "Indexing" of words to numbers. Our bag of words algorithm indexes the words alphabetically.

Word	does	my	open	password	reset	store	when
Index	0	1	2	3	4	5	6

With this numeric reference, we can now get rid of the words entirely. The statement "Reset my password" contains three words: "reset" (index: 4), "my" (index: 1), and "password" (index: 3). Thus, we can numerically represent "reset my password" as an array, setting 1 in the index positions for the words found in the statement and 0 for the other index positions. Figure 6 shows the journey "reset my password" takes in becoming the numeric array [0, 1, 0, 1, 1, 0, 0].

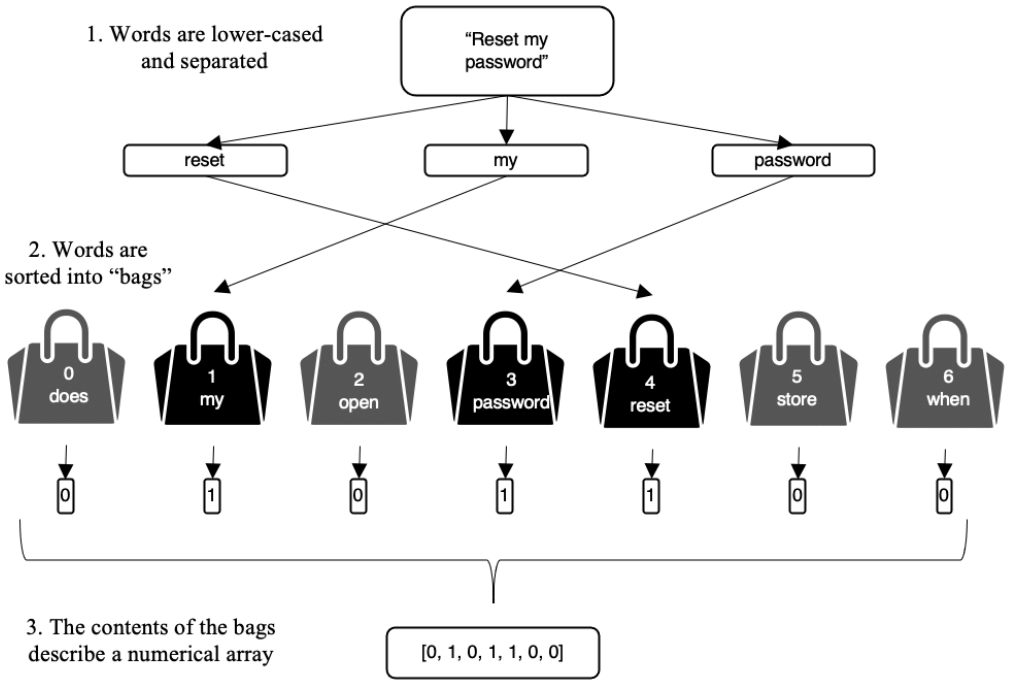


Figure 6 The journey from text statement to numeric array in a bag of words classifier.

Now we know how the bag of words classifier converts any text statement into a numerical array. This array covers the "input" to the classifier – we still need to show the output from the classifier. Our simple classifier has two output states: Positive (the input is `#reset_password`), and Negative (the input is not `#reset_password`).

Table 4 shows how several example utterances are mapped to numerical arrays by our simple bag of words classifier, and what output is assigned to each input.

Table 4 Numeric representation of some text statements

Textual input	"Bag" of words as a numeric array	Classification output
Reset my password	[0, 1, 0, 1, 1, 0, 0]	Positive
Password reset	[0, 0, 0, 1, 1, 0, 0]	Positive
When does the store open	[1, 0, 1, 0, 0, 1, 1]	Negative
When does my password reset	[1, 1, 0, 1, 1, 0, 1]	Negative

11.2.2 The mathematics behind a simple classifier

In the previous section, we saw how an input text phrase is converted to a numeric expression. This is only half the battle – we do not have a fully functional equation yet. We have described the input but not the output.

Let's continue with our very small example. By representing positive examples as 1, and negative examples as 0, we can build a fully numeric representation of our textual training data.

Table 4 Numeric representation of input and output

Text statement	As Input	As Output
Reset my password	[0, 1, 0, 1, 1, 0, 0]	1
When does the store open	[1, 0, 1, 0, 0, 1, 1]	0

We can describe this training data via two equations:

$$f([0,1,0,1,1,0,0]) = 1$$

$$f([1,0,1,0,0,1,1]) = 0$$

All we have to do now is discover the function f!

The function f will be discovered through a regression technique. Regression is a way of looking backward at a set of data to discover a function that best "fits" that data. Regression techniques iterate over a range of candidate functions, testing each one against the training data, and selecting the function that makes the smallest total errors.

The function selected by the regression process is only an estimate. (A perfect function would never have any errors!) We want the function to learn patterns from the training data and extrapolate those patterns to data it has not seen.

For classification, we will use logistic regression[2] algorithm. Logistic regression is a technique similar to linear regression, except that the target is a discrete classification. In our example, that classification is binary (0 or 1). The central equation in logistic regression is a linear function where every parameter is multiplied by a weight and then summed. There is also a constant added, called the "bias" (we will refer to it as weight_0). For a logistic regression with two parameters, the function would be as follows:

$$f = weight\_0 + weight\_1 * parameter\_1 + weight\_2 * parameter\_2$$

The amazing part behind machine learning is that given enough examples it will infer (learn) the parameter weights. The detailed math behind this learning is beyond the scope of this book, however we will describe the math at a high level.

Recall that our training data only has two output values: 1 and 0, based on whether the input is a positive (1) or negative (0) example of the class we are trying to detect. The values between 0 and 1 should represent the probability that an input belongs to the positive

[2] https://en.wikipedia.org/wiki/Logistic_regression

class – thus the machine learning algorithm should be constrained to returning values in the range [0,1]. The logistic function is shown below:

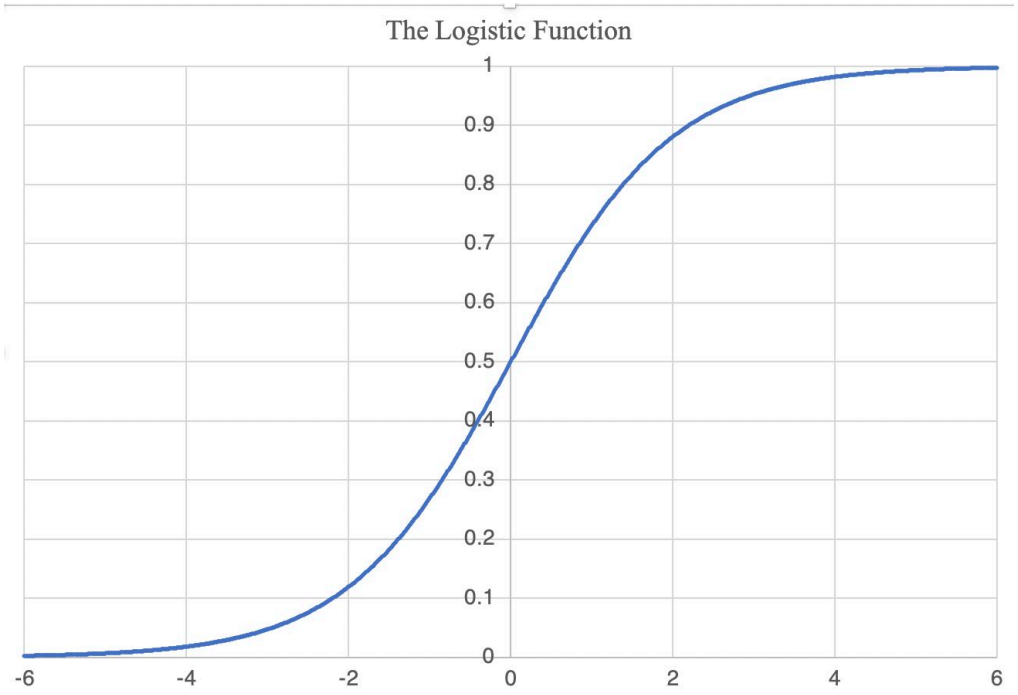


Figure 7 The logistic function.

The horizontal axis represents all the possible values of the function f (shown earlier).

The vertical axis is the probability that the value of f represents a positive example. This axis is bounded between the values of 0 and 1.

$$Probability\ of\ positive\ example = e^f / (1 + e^f)$$

Let's find out what our regression function really looks like. First, because our bag of words contains seven words, we know our regression function should have seven parameters (one for each word). We will also expect a constant value in the function (this is sometimes called the "bias"), as is typical in regression algorithms. We can find the exact parameter values learned by the classifier by training it on our two example statements ("reset my password" and "when does the store open").

Code Listing: Train a simple classifier

```
training_data = read_data_from_csv_file("training_2class_small.csv")
vec, clf, pipe = train_binary_classifier_pipeline("reset_password", training_data)
display(eli5.show_weights(clf, vec=vec, top=20, target_names=pipe.classes_))
```

I trained a logistic regression classifier on this example (code available via https://github.com/andrewrfreed/CreatingVirtualAssistants/tree/main/code/classifier_demo_ notebook) and inspected the weights learned by the classifier. These weights are shown in Figure 8.

y=reset_password top features

Weight[?]	Feature
+0.378	reset
+0.378	password
+0.378	my
+0.189	<BIAS>
-0.378	does
-0.378	open
-0.378	store
-0.378	when

Figure 8 Learned weights in the simple bag of words classifier.

The weights in Figure 8 are may be surprising. There is almost no variation in the weights. This is because we have only provided two examples ("reset my password" and "when does the store open"), and the classifier does not have much to learn from. As the number of training examples increases, the weights assigned to each feature will change. The fact that "password" and "my" have the same weight is a strong signal that we have undertrained (and that our classifier is not very clever).

Now that we have learned the feature weights, we can use the logistic equation to classify any example we like. Let's first test "reset my password". Each of the words "reset", "my", "password" contributes a 1 (feature value) times 0.378 (feature weight). And we must always include the bias parameter (0.189) when using the classifier. When the classifier sees "reset my password" it classifies it as `#reset_password` with 0.79 confidence:

$$f = 0.189 + 0.378*1 + 0.378*1 + 0.378*1 = 1.323$$

$$e^f = e^{1.323} = 3.755$$

$$probability = e^f / (1 + e^f) = (3.755/4.755) = 0.790$$

The system has fairly high confidence that this utterance should be classified as `#reset_password`. That's good – it's is `#reset_password`! The closer to 1 the confidence is, the more likely the utterance belongs to that intent.

Similarly, we can build the equation for "when does the store open". The equation starts with the bias parameter. The classifier removes common words, so "the" is removed – we'll show it in the equation with a 0. The other four words have a negative weight as shown in

Figure 8, so our equation includes four sets of the feature value 1 times the feature weight -0.378.

$$f = 0.189 + 0 - 0.378*1 - 0.378*1 - 0.378*1 - 0.378*1 = -1.323$$

$$e^f = e^{-1.323} = 0.266$$

$$probability = e^f / (1 + e^f) = (0.266/1.266) = 0.210$$

The system has fairly low confidence (0.21) that this utterance is a `#reset_password` example. That's also good – it's not `#reset_password`! The closer to 0 the confidence is, the more likely the utterance does not belong to that intent.

The classifier only knows how to score words that are in the training set. Any words it has not trained on will be ignored. Consider the phrase "where are you located". None of the words "where", "are", "you", or "located" exist in the training set, so the equation only includes the bias parameter. "Where are you located" is classified as `#reset_password` with 0.547 confidence.

$$f = 0.189$$

$$e^f = e^{0.189} = 1.201$$

$$probability = e^f / (1 + e^f) = (1.201/2.201) = 0.547$$

Wait a minute - "Where are you located" is more likely to be `#reset_password` than not? That's what the 54.7% probability means. This may be a surprising result – but recall that this classifier is severely undertrained. With only two training examples it should be no wonder that this classifier makes some questionable predictions. A classifier must be trained with a larger volume and variety of training data if it is going to make good predictions.

Try it yourself!

Try to classify some utterances with this classifier. Note that if your utterances do not contain any of the words "reset", "my", "password", "where", "are", "you", or "located", it will be classified as #reset_password with 0.547 confidence!

Let's visualize the previous three classifications. In Figure 9 each example is plotted on the logistic regression curve. Try a few examples of your own – and predict how the classifier will treat them. What happens if you repeat words ("password help please I need my password reset")? What happens if you have a very short statement ("password reset")? What happens if you "mix" words from the two training examples ("when does the password reset")? Explore via the Jupyter notebook in this chapter's source code repository!

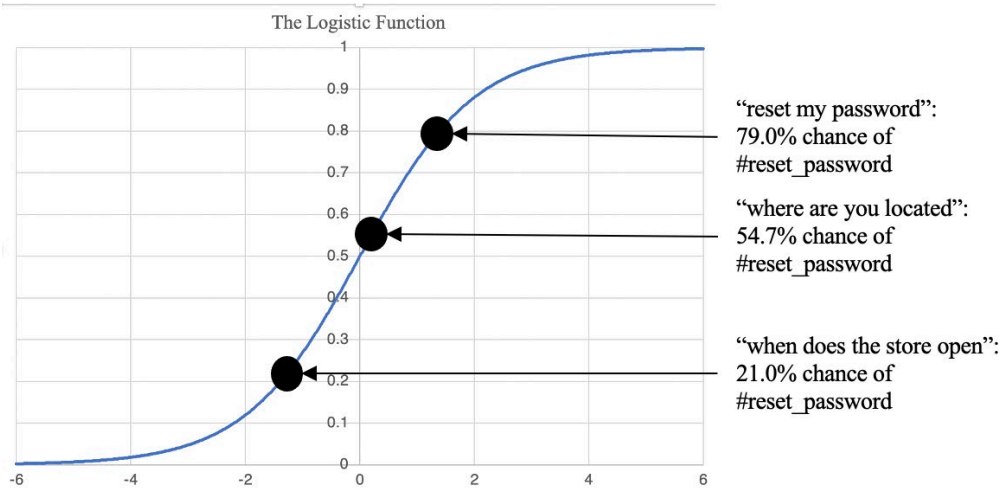


Figure 9 Logistic regression results for a classifier trained on two examples

Wait a minute – why don't we see 100% confidence anywhere?

You may be surprised that the classifier does not make predictions with 100% confidence (or in fact 0% confidence). The examples we tested the classifier on came straight from the training data, so we might reasonably expect the classifier to remember that.

Different classifiers have different training methodologies. Some classifiers do in fact remember their training data and will return 100% confidence when you test them with an example seen in training. Other classifiers make inferences during training and then "forget" the rest of the training data. Our simplistic bag of words classifier is in this latter group.

A classifier with such light training is hardly trustworthy and is only used for illustrative purposes. When more training data is added, with both volume and variety, the results from the classifier will be more accurate.

11.3 Expanding the simple classifier

In the previous section, our classifier could only predict one class (one intent). Either an utterance was `#reset_password` or it was not. You may call this a prediction of two classes (`#reset_password` and everything else), but this is still far short of useful. A typical virtual assistant will support dozens of classes. FICTITIOUS INC needs to support at least a dozen intents.

11.3.1 Predicting more than one class

Can we extend this simplistic model to identify more than one class? Yes, we can! Let's make a very small extension to our previous classifier. Instead of predicting one class, let's

predict three classes. We will provide training data as in Table 5. This table has three intents, each with one example utterance.

Table 5 The smallest possible training set for three output classes

Input (Utterance)	Output (Intent)
Reset my password	#reset_password
When does the store open	#store_hours
Where is my nearest store	#store_location

In the one-class example, we had seven "bags" of words: "reset", "my", "password", "when", "does", "store", "open". The new third utterance in Table 5 adds three new bags for the words "where", "is", and "nearest". (We already have bags for "my" and "store" from the previous utterances). We will now have ten bags of words, and we should expect that the weights will be different in the new model.

Before we train the model we have to figure out how to assign a numeric value to the output. You may consider giving each class an arbitrary number: say 0 for `#reset_password`, 1 for `#store_hours`, and 2 for `#store_location`. This presents a numeric problem – how can we design a function whose minimum value is `#reset_password`, and what does it mean for `#store_hours` to be in the "middle"? (And what would happen if we "changed the order" of the intents?)

The answer is to use multiple classifiers. Each classifier can focus on detecting a single class and we can use the aggregated results to determine which class is the most likely. The classifier that returns the highest positive confidence will give us the output class. Table 6 shows how each classifier uses a sort of tunnel vision so that it only worries about one class at a time.

Table 6 Each classifier has a binary view of the training data

Input (Utterance)	Classifier 1	Classifier 2	Classifier 3
Reset my password	#reset_password	#not_store_hours	#not_store_location
When does the store open	#not_reset_password	#store_hours	#not_store_location
Where is my nearest store	#not_reset_password	#not_store_hours	#store_location

I used a code snippet like the following to train three classifiers. This code is available in the book's GitHub repository. The training data from Table 5 was stored in the file `training_3class_small.csv`.

Training three classifiers

```
training_data = read_data_from_csv_file("training_3class_small.csv")
```

```
#Train the three classifiers
reset_password_vec, reset_password_clf, reset_password_pipe = \
    train_binary_classifier_pipeline("reset_password", training_data)
store_hours_vec, store_hours_clf, store_hours_pipe = \
    train_binary_classifier_pipeline("store_hours", training_data)
store_location_vec, store_location_clf, store_location_pipe = \
    train_binary_classifier_pipeline("store_location", training_data)

#Inspect the weights for each classifier
display(eli5.show_weights(reset_password_clf, vec=reset_password_vec, top=20,
       target_names=reset_password_pipe.classes_))
display(eli5.show_weights(store_hours_clf, vec=store_hours_vec, top=20,
       target_names=store_hours_pipe.classes_))
display(eli5.show_weights(store_location_clf, vec=store_location_vec, top=20,
       target_names=store_location_pipe.classes_))
```

When we train the three classifiers, we now get three sets of weights, as shown in Figure 10.

y=reset_password top features

Weight[?]	Feature
+0.531	reset
+0.531	password
+0.234	my
-0.234	does
-0.234	open
-0.234	the
-0.234	when
-0.297	is
-0.297	nearest
-0.297	where
-0.430	<BIAS>
-0.531	store

y=store_hours top features

Weight[?]	Feature
+0.460	when
+0.460	the
+0.460	open
+0.460	does
+0.230	store
-0.230	is
-0.230	nearest
-0.230	where
-0.230	password
-0.230	reset
-0.460	my
-1.000	<BIAS>

y=store_location top features

Weight[?]	Feature
+0.531	where
+0.531	nearest
+0.531	is
+0.297	store
+0.234	my
-0.234	does
-0.234	open
-0.234	the
-0.234	when
-0.297	password
-0.297	reset
-1.258	<BIAS>

Figure 10 Three binary classifiers each with their own independent weights. Our only training data was "reset my password" for reset_password, "when does the store open" for store_hours, and "where is my nearest store" for store_location.

We can make a couple of observations from the weights:

- For the `#reset_password` classifier, the word "my" has a lower weight than "reset" and "password". This is because "my" also shows up in the `#store_location` example. The classifier only associates "reset" and "password" with one intent, while it associates "my" with *two* intents. (For the same reason, the word "store" has reduced weight in `#store_hours`.)
- The `#reset_password` classifier has a strong negative weight on the word "store", which has appeared in every single utterance that is not `#reset_password`.
- Each classifier has a negative bias. This is because each classifier has seen more negative than positive examples. This is an implicit signal that each class is somewhat rare.
- Several of the weights are larger within each classifier as compared to Figure 8, which had a maximum weight of 0.378 (and -0.378). Having more examples per classifier

(three examples vs two) is allowing stronger inferences to be made.

To test a single utterance, we now need to run through a series of three equations. Let's evaluate "reset my password":

$$f\_reset\_password = -0.430 + 0.531 + 0.234 + 0.531 = 0.866$$

$$e^{f\_reset\_password} = e^{0.866} = 2.377$$

$$probability = e^{f\_reset\_password} / (1 + e^{f\_reset\_password}) = (2.377/3.377) = 0.704$$

$$f\_store\_hours = -1 - 0.230 - 0.460 - 0.230 = -1.920$$

$$e^{f\_store\_hours} = {}^{-1.920} = 0.147$$

$$probability = e^{f\_store\_hours} / (1 + e^{f\_store\_hours}) = (0.147/1.147) = 0.128$$

$$f\_store\_location = -1.258 - 0.297 + 0.234 - 0.297 = -1.618$$

$$e^{f\_store\_location} = {}^{-1.618} = 0.198$$

$$probability = e^{f\_store\_location} / (1 + e^{f\_store\_location}) = (0.198/1.198) = 0.165$$

The predictions can be summed up in the probability table shown in Table 7. In this table, we also add a "negative prediction probability", which is simply one minus the positive prediction probability. This negative prediction probability is how likely an utterance does *not* belong to the intent.

Table 7 Probabilities from each of the three binary classifiers on the input text "reset my password". #reset_password has the highest confidence so it is chosen as the output.

Classifier	Positive Prediction Probability	Negative Prediction Probability
`#reset_password`	**0.704**	0.296
`#store_hours`	0.128	0.872
`#store_location`	0.165	0.834

Table 7 has the promising result that utterances which are completely unrelated to an intent, are not predicted to belong to that intent. The system has 0.872 confidence that "reset my password" is *not* a `#store_hours` utterance. The more we train the classifier, the more confident it can be.

Why don't the probabilities add up to one across classifiers?

Within a classifier the probabilities do add up to one: every input is either a positive or negative example of that class. The probability for positive and the probability for negative must sum up to one.

The three classifiers are completely independent. There is no reason their positive probabilities need to be related.

If your virtual assistant provider returns probabilities that add up to one, there are two likely possibilities:

- They are scaling the probabilities so that they add to one
- They are using a different algorithm, like an all-in-one classifier, which we will explore later in this section

We can also see a promising trend when we test a brand new, off-the-wall utterance that does not have any overlap in our training data. The utterance "tell me a joke" contains no words that overlap with the training data. The predictions for "tell me a joke" are summed up in the probability table shown in Table 8.

Table 8 Probabilities from each of the three binary classifiers on the input text "tell me a joke". The negative predictions have higher confidence since this example has no overlap with any intent class.

Classifier	Positive Prediction Probability	Negative Prediction Probability
`#reset_password`	0.394	0.601
`#store_hours`	0.269	0.731
`#store_location`	0.221	0.779

None of the classifiers return a probability greater than 50% on this utterance. This means that "tell me a joke" is not predicted to belong to any of our three intents. This is very good news! We can infer that any utterance that has no overlap with our training data will not belong to any of the three intents. Our three classifiers can identify "out of scope" utterances by default!

What classification should the system give to "tell me a joke"? `#reset_password` is the highest scoring intent, but the confidence of 0.394 is very low. Consider that if you picked a random intent, you would have 0.333, or 1 in 3, confidence. Your virtual assistant platform may return `#reset_password`, or "no intent found" (try it out!). Virtual assistants generally apply thresholds to intent confidence, so that low confidence intents are not chosen.

The more training examples we add to the training data, the better predictions we will get. In a reasonably balanced training set, the bias parameter will naturally trend downward. This will help the classifier recognize for any given intent, most utterances do not belong to that intent. Let's pretend you have 10 intents each with 5 examples – each intent will be trained with 5 positive examples and 45 negative examples. Since each intent has nine times as many negative examples as positive, each classifier will naturally learn that "most utterances aren't positive examples".

11.3.2 An all-in-one classifier

Your virtual assistant platform may use a single classifier rather than an array of binary classifiers. A single classifier has a similar mathematical structure but slightly different

equations. I will demonstrate the classifier here and provide code in my GitHub repository[3] for you to explore.

Figure 11 shows a set of weights for a single classifier trained on our three input classes. Compared to Figure 10, the weights are overall smaller in this single classifier and the bias parameter is not always negative. This is because every input is assumed to belong to this classifier – there is no concept of an "out of scope" utterance in this classifier. (If we wanted to detect "out of scope", we'd have to add a new intent specifically for it.) Thus, it is harder for the weights to get large – all the probabilities have to add up to one.

y=reset_password top features		**y=store_hours** top features		**y=store_location** top features	
Weight?	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**
+0.122	reset	+0.119	when	+0.122	where
+0.122	password	+0.119	open	+0.122	nearest
+0.086	<BIAS>	+0.119	does	+0.122	is
+0.059	my	+0.059	store	+0.063	store
-0.059	does	+0.014	<BIAS>	+0.059	my
-0.059	open	-0.059	password	-0.059	does
-0.059	when	-0.059	reset	-0.059	open
-0.063	is	-0.059	is	-0.059	when
-0.063	nearest	-0.059	nearest	-0.063	password
-0.063	where	-0.059	where	-0.063	reset
-0.122	store	-0.119	my	-0.100	<BIAS>

Figure 11 Feature weights in the all-in-one classifier

Figure 12 shows predictions for the same two utterances we tested on the binary classifiers: "reset my password" and "tell me a joke". The classifier will pick the classification with the highest probability. We can see that the classifier "got it right" on "reset my password", while giving that prediction a 47.3% probability of being correct. "tell me a joke" is also classified as `#reset_password` with a lower probability of 36.8%.

[3] https://github.com/andrewrfreed/CreatingVirtualAssistants/tree/main/code/classifier_demo_notebook

```
In [111]: eli5.show_prediction(clf, "reset my password", vec=vec, show_feature_values=True)
```

Out[111]:

y=reset_password (probability 0.473, score 0.390) top features

Contribution?	Feature	Value
+0.122	reset	1.000
+0.122	password	1.000
+0.086	<BIAS>	1.000
+0.059	my	1.000

y=store_hours (probability 0.256, score -0.224) top features

Contribution?	Feature	Value
+0.014	<BIAS>	1.000
-0.059	password	1.000
-0.059	reset	1.000
-0.119	my	1.000

y=store_location (probability 0.271, score -0.166) top features

Contribution?	Feature	Value
+0.059	my	1.000
-0.063	reset	1.000
-0.063	password	1.000
-0.100	<BIAS>	1.000

```
In [112]: eli5.show_prediction(clf, "tell me a joke", vec=vec, show_feature_values=True)
```

Out[112]:

y=reset_password (probability 0.362, score 0.086) top features

Contribution?	Feature	Value
+0.086	<BIAS>	1.000

y=store_hours (probability 0.337, score 0.014) top features

Contribution?	Feature	Value
+0.014	<BIAS>	1.000

y=store_location (probability 0.301, score -0.100) top features

Contribution?	Feature	Value
-0.100	<BIAS>	1.000

Figure 12 Output from two predictions with the all-in-one classifier. In the first example, each classifier has learned how the words in "reset my password" contribute to an intent, and the utterance is easily classified as #reset_password. In the second example, the prediction is ambiguous since the classifier has not seen any of the words in "tell me a joke".

We can once again compare these predictions to randomly picking an intent. The 47.3% confidence for "reset my password" is significantly higher than random selection (33.3%). The 36.8% confidence for "tell me a joke" is almost indistinguishable from random selection.

11.3.3 Comparing binary classifiers to all-in-one classifiers

An all-in-one classifier may consume fewer system resources than an array of several classifiers, but I find the "confidence math" confusing. Is 47.3% a good probability? Is 36.8% bad? When training volume is small, it's tough to say what a good probability level is. Fortunately, when the classifier is trained on higher volumes of data the classifier will make stronger predictions.

Our test above was not quite fair to the all-in-one classifier. It must be trained with an additional class that you could call "other" or "everything else". This class would represent ALL of the statements that do not belong to any of the intents you are specifically training. In practice, training this "other" class can be difficult and time-consuming due to the sheer number of possibilities that could be loaded into that class.

Regardless of which classifier type you use, every positive example for one intent is necessarily a negative example for every other class. The binary classifiers thus automatically get lots of negative examples and start to implicitly handle an "other" class through their bias parameter. The all-in-one classifier must be explicitly trained with "other" examples.

Binary classifiers can be trained with an "other" class if you choose. Virtual assistant platforms expose this in a variety of ways. Some let you create "counterexamples" or "negative examples" – these are inputs that should be treated as negative examples for every intent. In other virtual assistant platforms you must define "other" as a positive class – some developers call it `#out_of_scope`.

Do commercial vendors use the bag of words algorithm?

No. The bag of words technique is a quick way to train a text classifier, but it has numerous shortcomings. Your commercial provider uses an algorithm much smarter than the simplistic bag of words counter used in this example.

Here is a summary of failings in bag of words that stronger algorithms solve for:

- Lemmatization: "open" and "opening" are different "words" with the same lemma, which means they should be treated the same. Lemmatization helps make that happen. Lemmatization also ensures that plurals are treated the same as singulars ("store" and "stores")
- Mis-spellings: Bag of words does not know that "sotre" means "store".
- Order *does* matter: Word sequences have more meaning than individual words.
- Synonyms and similar words are treated independently. "Help" and "assist" should have the same meaning.
- All words are not created equal: Rare words generally are more important than common words and should be given more weight. The term frequency-inverse document frequency (TF-IDF) algorithm solves for this.

11.4 Extending even further

We started the chapter predicting just a single intent using a tiny amount of data. In the previous section, we expanded by predicting multiple intents but still with a very small amount of data. You should start seeing the patterns of how we expand the assistant with more training data and more intents.

11.4.1 What happens when you add more training data?

As you expand your training data with increased volume and variety you will improve the performance of the classifiers. In my original test, I only used three total utterances – one for each intent. Let's see what happens when we increase the training volume with 500% more examples. We also enhance the variety by using different words and phrases. Table 9 shows the new training data.

Table 9 New training data for our classifier. This has 500% more examples than the first training data set.

#reset_password	#store_hours	#store_location
Reset my password	Are you open on Thanksgiving	Where are you located
I can't log in	How late are you open today	Are you downtown
I forgot my password	Give me your hours please	Are you near the mall
I have a pw issue	When does the store open	Where is my nearest store
Stuck at the home screen	What time do you close	How do I find you
Can't remember my login info	What are your hours	I need a map to your location

The increased training data has a significant effect on the classifier. In Figure 13 I show the weights learned by this classifier after training on the new data. You can see the stronger inferences the classifier is learning through the wider variation between the weights. The first classifier in this chapter assigned equal weights (0.378) to every word, suggesting that

each of the words was equally important. The updated classifier set, with its different weights, knows that some words are more important than others for identifying intent.

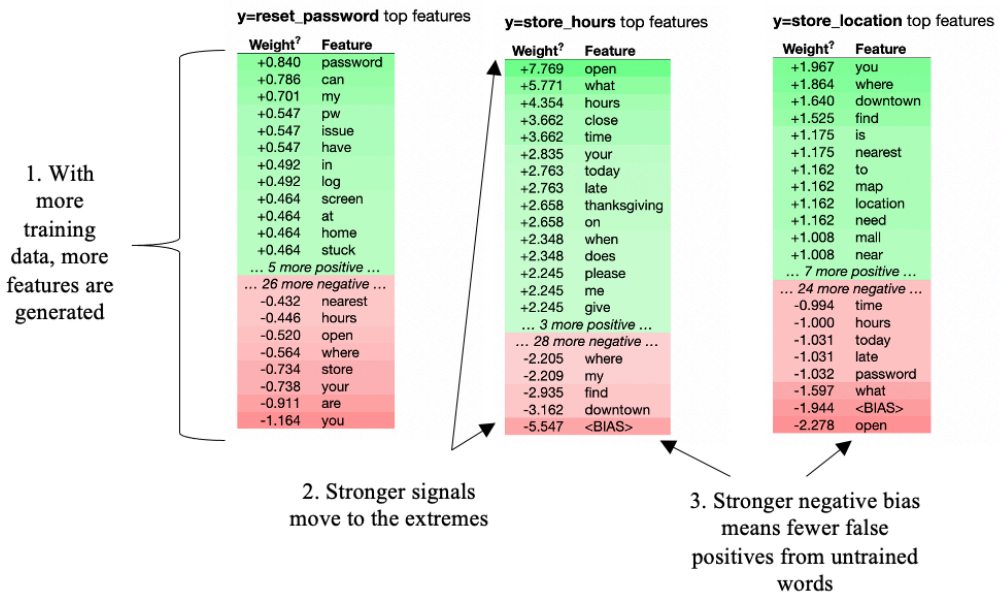


Figure 13 This classifier was trained with six times as much data variation as the original classifier and produces more features and stronger signals. We will show that this also leads to predictions with higher confidence.

This variation of weights is important – it starts to capture the subtlety in natural language. The original classifier felt that the words "reset", "my", and "password" were equally important. The improved classifier group has learned variation: "password" is the most important word of that set for the `#reset_password` intent. Similarly, the word "open" is much more important for `#store_hours` than "store".

The feature weights still seem questionable – what gives?

In Figure 13, words like "you", "your", and "in" are providing more signal than makes sense intuitively.

The classifiers in this chapter are all still very naïve – they treat all words the same way. You can treat words differently with features like stopword removal, term frequency weighting, stemming and lemmatization, or other features you can imagine. More sophisticated features like these can make your classifier smarter.

Most commercial vendors do not expose the algorithms that they use, but they do use more sophisticated algorithms than bag of words. This helps them produce stronger results with less training data.

The classifiers in Figure 13 also make better predictions. The phrase "reset my password" is now predicted as `#reset_password` with 0.849 probability (it was 0.704 before we added more training data); the phrase "tell me a joke" is predicted as NOT `#reset_password` with 0.623 probability (previously 0.601). As we can see in Table 10, the performance of our classifiers improved as we added a greater volume and variety of training data.

Table 10 Summary of classifiers and performance in this chapter. Prediction accuracy increases when we add better training data.

Reference	Number of intents	Number of training examples	"reset my password" prediction	"tell me a joke prediction"
Figure 8	1	2	`#reset_password: 0.790`	`#reset_password: 0.547`
Figure 10	3	3	`#reset_password: 0.704`	`Not #reset_password: 0.601`
Figure 13	3	18	`#reset_password: 0.849`	`Not #reset_password: 0.623`

11.4.2 Exercise: Experiment on your own!

The code used to generate all of the figures and tables in this section is available in the book's Git repository at https://github.com/andrewrfreed/CreatingVirtualAssistants/tree/main/code/classifier_demo_notebook. The code consists of a Jupyter Notebook[4] and some sample data. Download the notebook and explore!

Experiment by adding more training data or more intent classes and see how the predictions change. Change one thing at a time so that you can assess its impact – for instance, add a single new intent before adding a second new intent, this way you can understand the impact from each intent you've added.

If you are feeling adventurous try changing the classification algorithm itself. Can you come up with better features than just counting the words? Stronger features will help you train a classifier with less input data. Even though commercial vendors use strong classification algorithms you may find it instructive to experiment with your own algorithms.

11.5 Summary

- Classifiers convert text to numeric values to build predictive regression models.
- The predictions of a classifier are strongly influenced by the volume and variety of the training data used.
- Virtual assistants may use a single classifier or an array of binary classifiers. This implementation choice affects how you interpret the prediction and the associated probability behind that prediction.

[4] https://jupyter.org/

12

Training for voice

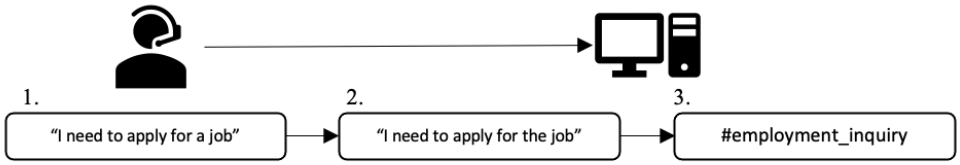
This chapter covers:

- Collecting data to test and train a speech-to-text model.
- Evaluating the impact of a speech-to-text model on the virtual assistant's success metric.
- Identifying the most appropriate speech training option for a virtual assistant.
- Training custom speech recognition models for open-ended and constrained inputs.

FICTITIOUS INC created its first conversational assistant through a text chat channel. That assistant has been so successful that FICTITIOUS INC is now planning to expand to a telephone-based assistant as well. This assistant will cover the same customer care intents as the original assistant but will take voice input rather than text input.

Speech-to-text models *transcribe* audio into text. In a voice virtual assistant, the user speaks, and the transcription of that audio is passed to the virtual assistant for intent detection and dialog routing. In previous chapters we have seen how virtual assistants use text: data must be collected, and the virtual assistant needs to be trained and tested.

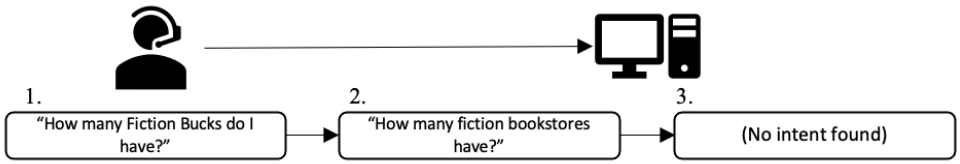
FICTITIOUS INC will add speech recognition technology to their existing assistant to transcribe audio input into text. They have experimented with speech recognition technology in the past with mixed results. FICTITIOUS INC's transcriptions have been accurate on generic language like "reset my password" or "what time are you open" or "I need to apply for a job". Figure 1 shows how an untrained speech-to-text model integrates with a virtual assistant. The model may make small mistakes (confusing "a" and "the") but captures the essence of the user's statement.



1. User speaks into a microphone
2. Speech to text engine transcribes audio into text
3. The virtual assistant infers an intent from the textual transcription (not from the audio)

Figure 1 Speech-to-text integration into a virtual assistant. An "out of the box" speech-to-text model works well on common language.

However, the speech-to-text technology has struggled on some with FICTITIOUS INC's jargon. (For example, FICTITIOUS INC's loyalty program is called "Fiction Bucks".) An untrained speech-to-text model is unlikely to have come across that language before, as seen in Figure 2. FICTITIOUS INC needs a speech-to-text solution which can handle both generic language and their jargon.



1. User asks how many loyalty points they have
2. Speech to text engine fails to transcribe the key phrase "Fiction Bucks"
3. The virtual assistant cannot find an intent in the transcribed text

Figure 2 An untrained speech-to-text model will struggle on jargon. The model has never seen "Fiction Bucks" before and does not understand it well.

In this chapter, you will learn how to use speech recognition technology. You will learn how to collect audio data for testing and training speech recognition technology. Then you will learn how to train a custom speech-to-text model and use it in your virtual assistant.

12.1 Collecting data to test a speech-to-text model

FICTITIOUS INC needs data to test its speech-to-text model. With this data, they can find out how accurate a speech-to-text model will be in the assistant and where they will have to train the model to get more accurate transcriptions. They can use many of the same data sources they used when training their text-based virtual assistant. As with the textual data, the speech data they collect can be used for both training and testing the virtual assistant.

Speech data is a pairing of an audio file and the textual transcription of that audio file. Table 1 shows some sample speech data. When we play the audio file Password1.wav we will hear someone saying "reset my password". Training and testing a speech-to-text model requires this paired data format: audio and text.

Table 1 Sample speech data. Every audio file is associated with the text spoken in that file.

Audio file	Text
`Password.wav`	"reset my password"
`EmploymentInquiry.wav`	"I need to apply for a job"
`LoyaltyPoints.wav`	"how many Fiction Bucks do I have?"

This format is required because we want the speech-to-text model to know what words sound like. As seen before, the words "Fiction Bucks" sound a lot like the words "Fiction books". Since FICTITIOUS INC wants their speech-to-text model to recognize the phrase "Fiction Bucks", they need to have examples of what that phrase sounds like.

Transcription works using phonetics behind the scenes

Words are a sequence of phonetic sequences. Each segment of the sequence is called a phoneme. Speech-to-text engines extract potential phonemes from an audio signal and attempt to combine them into legitimate words or word phrases.

The International Phonetic Alphabet (IPA) pronunciation for "Fiction Bucks" and "Fiction Books" is similar:

`.ˈfɪk.ʃɨn .ˈbʌks` = Fiction Bucks

`.ˈfɪk.ʃɨn .ˈbʊks` = Fiction Books

The only difference in these two phrases is in the third phoneme (`ˈbʌks` vs `ˈbʊks`)

The three V's of data apply to speech data too! Speech data must have variety, volume, and veracity to evaluate and train a speech-to-text model.

- **Variety**: Representative variety in data includes both what people will say and how they will say it. There should be audio data for each of the intents (what people will say) as well as demographic diversity in the speakers (how they will say it).
- **Volume**: Depending on the use case, it can take between 5 and 25 hours of audio data to train a speech-to-text model. Less data is required if the model is only going to be tested.
- **Veracity**: The audio files must be accurately transcribed for both evaluating and training a speech-to-text model. "Garbage in, garbage out."

Bias alert!

A common pitfall in artificial intelligence projects is when data does not come from a diverse set of users. Beware the temptation to gather audio data only from your development team. It is highly unlikely that the demographics in your team match the demographic diversity of your user population. Without diverse data, you may be biasing your solution against some groups of users.

An expert takes six to ten hours to transcribe one hour of audio into text. The most common transcription practice is to feed audio into a speech-to-text engine to get an "initial transcription". Then the human transcriber listens to the audio and corrects any mistakes in the "initial transcription". Transcription takes longer than you might expect!

The three V's of data are useful for understanding what FICTITIOUS INC's data needs are. Let's now look at potential sources for this data. The two primary sources of data are call transcripts and "synthetic" data. Let's start with call transcripts.

12.1.1 Call recordings as speech training data

FICTITIOUS INC is like many companies with human-staffed customer service call centers. They record customer service calls "for quality and training purposes". These recorded calls help train their virtual assistant. FICTITIOUS INC has already reviewed these calls to extract textual utterances; now they can extract audio segments as well.

Call center recordings need some work done on them before they can be used for speech training and evaluation. As shown in Figure 3, a call center recording includes more audio than is needed, because it contains two speakers. The audio segments where the call center agent is speaking are not useful for the virtual assistant. Only the segment where the end-user is speaking is useful, and this part needs to be extracted from the recording.

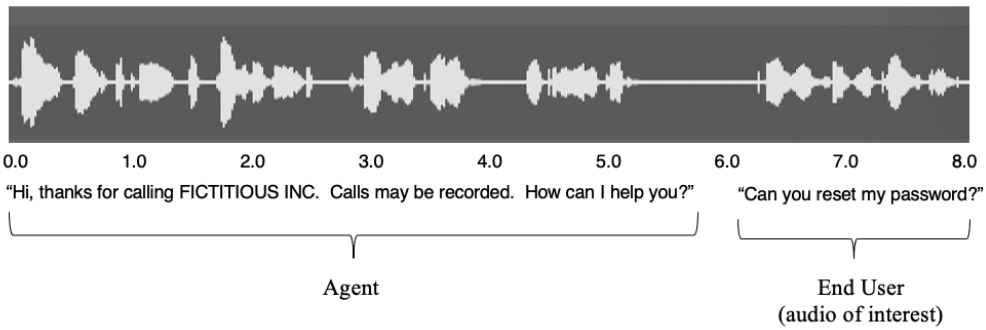


Figure 3 Sample call recording including agent greeting and user intent statement. Only the end-user's statement is useful for training and testing the virtual assistant.

Figure 3 shows the first eight seconds of audio in a call recording. In this call, the agent speaks for the first five seconds, there's a one-second break, then the end-user speaks for the last two. FICTITIOUS INC needs to cut these two seconds out of the recording and save

them into a new file, such as `can_you_reset_my_password.wav`, and also make sure the transcription is stored with the file. (Putting the transcription in the file name is a potential shortcut.)

Extracting the audio of interest from an audio file

If the audio file contains separate channels for each speaker, the channel from the caller can be automatically extracted for use in speech training. If the audio file does not separate the channels, the caller's audio needs to be extracted manually.

Figure 3's phone call was direct and to the point. Figure 4 depicts a contrasting call. In this call, the user is verbose. The user seems grateful to connect with another human being and engages indirectly. This is not how users interact with virtual assistants. This recording should not just be segmented on the end-user speech. Instead, only the audio segment including the intent statements should be extracted. The recording arguably includes three intent statements, each of which could be extracted to an isolated file.

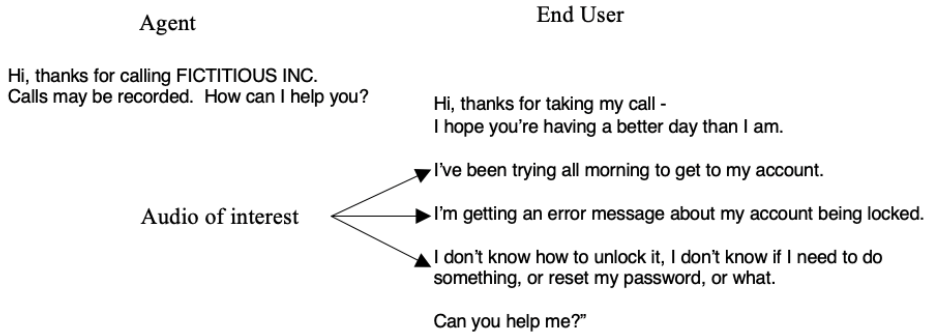


Figure 4 End-users can be verbose when talking with agents. They are unlikely to greet a virtual assistant the same way. Only part of the end-user's statement is relevant to the virtual assistant.

After FICTITIOUS INC has processed these two call recordings, they will have four pieces of audio data as shown in Table 2.

Table 2 FICTITIOUS INC audio data collected after reviewing two call recordings.

File	Transcription
File 1	Can you reset my password?
File 2	I've been trying all morning to get to my account.
File 3	I'm getting an error message about my account being locked.
File 4	I don't know how to unlock it, I don't know if I need to do something, or reset my password, or what.

Call recordings are a wonderful source of speech data because they are real data. FICTITIOUS INC's call recordings will include a variety of callers with demographic diversity. These recordings include the way users pronounce FICTITIOUS INC jargon in a variety of accents. FICTITIOUS INC employees may enunciate "Fiction Bucks", but their callers may slur the two words together. This distinction is important for the speech engine. Lastly, FICTITIOUS INC has already been reviewing call recordings to gather intent training data. Using the recordings for speech training is a second benefit from the same data source.

Call recordings also have downsides. Processing call recordings is a lengthy process because the audio of interest must first be isolated and then transcribed. Call recordings contain more data than is useful for training an assistant: everything the agent says can be omitted, as well as any greetings or off-topic banter from the caller. Since humans talk differently to each other than they do to automated solutions, there may be large segments of each recording that are unusable.

Call recordings can take a lot of disk space. Many call centers re-encode and compress their recordings to save disk space. This process can reduce disk space needs by an order of magnitude, but also lowers the quality of the call recordings. (A WAV file may be ten times larger than an MP3 version of the same call.) If the call recording quality is too low it may be unusable for speech training or it may skew speech training results. Speech engines are more accurate when using lossless formats (like WAV and FLAC) rather than lossy (like MP3, MULAW, and OGG).

The benefits and downsides of using call recordings is summarized in Table 3.

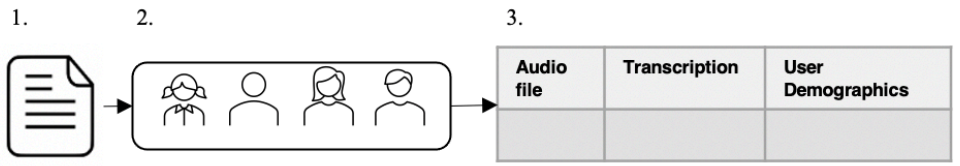
Table 3 Summary of the benefits and downsides of using call recordings as speech data.

Benefits of call recordings	Downsides of call recordings
• Recordings contain a representative variety of caller demographics, accents, and pronunciations of domain-specific words and phrases. • Recordings contain a representative variety of topics. • Recordings can be used for identifying intents and training speech-to-text models – multiple benefits from the same source!	• Extra time is required to process a recording to extract the segments of interest. • Recordings require a lengthy transcription process. • Humans talk differently to humans than virtual assistants. • Recordings are sometimes saved in a lossy format. The audio quality of the recording can be worse than the actual call.

Call recordings can be a good source of speech data, but they also have challenges. Let's look at another source of speech data.

12.1.2 Generating "synthetic" speech data.

Instead of extracting "real" speech data from call recordings, FICTITIOUS INC can generate their own speech data. Since this data will not come from real users, we will call this "synthetic" data. The data collection process is outlined in Figure 5.



1. Build a script of utterances. Each scripted line is an intent or entity training example.
2. A diverse set of users read the script. An audio file is created for each line in the script.
3. Audio data is stored with the textual transcription and demographic summary.

Figure 5 Synthetic data collection process. Multiple people record multiple audio segments to create speech data.

FICTITIOUS INC can take some or all of the textual utterances they were going to train their intent classifier on and add them into a script for users to read. Then they will get a variety of users to record these utterances. The recorded utterances will be stored in individual audio files, accompanied by their textual transcription and summary demographic information. A sampling of FICTITIOUS INC's synthetic speech data is shown in Table 4.

Table 4 Sample speech data from synthetic data collection. Demographics of the speaker are collected for assuring a diversity of voices.

Audio file	Transcription	Demographics
`reset_my_password_1.wav`	"reset my password"	Male, US Southern English accent
`reset_my_password_2.wav`	"reset my password"	Female, Hispanic accent
`how_many_fiction_bucks_do_i_have_1.wav`	"how many Fiction Bucks do I have"	Female, US Northeastern English accent

Synthetic data collection is a popular option because of control and speed. FICTITIOUS INC can dictate exactly what data they want to request. They can specify an exact distribution of data collected for each intent or domain-specific phrase. The uniformity in the data collection process helps them be aware of any skew in speaker demographics, and they can apply targeted efforts to close those gaps.

FICTITIOUS INC can request synthetic data in a crowdsourced manner, with individuals contributing as little as five minutes each to record audio. The transcription process for synthetic data takes less time than for call recordings because the transcriber knows what the speakers are expected to say.

Why is transcription needed in synthetic data collection?

Speakers are expected to follow a script and speak a handful of known phrases. But speakers are human and make mistakes! In synthetic data collection, the role of the transcriber is to verify that each audio file contains the expected text. Speakers make enough mistakes that this extra effort is worth it.

Synthetic data collection has drawbacks as well. The usefulness of synthetic data is directly related to the quality of the script read by the speakers. If the script is filled with phrases that real users don't say, the audio data will not be very useful. Additionally, synthetic data is often collected from people who are pretending to be real users, not the real users themselves. They make speak and intonate different from real users. (Do they say "can you reset my password" with a sense of urgency, or nonchalance?)

Lastly, the demographic distribution of users in the synthetic data collection may be completely different than the distribution of the real user population. If the difference between these distributions is too large, the data will be biased and will likely bias the virtual assistant. User demographic groups that are underrepresented may find that the assistant does not understand them. The benefits and downsides of using call recordings are summarized in Table 5.

Table 5 Summary of the benefits and downsides of synthetic recordings as speech data.

Benefits of synthetic recordings	Downsides of synthetic recordings
• Collect exactly what phrases you want. • Data uniformity helps identify gaps in speaker demographics. • Transcription takes less time. • Recordings can be crowdsourced.	• Data collected is only valuable if the script is representative of what callers will say. • Does not come from real users but from simulated users. • It can be difficult to match the demographic distribution of the real user population.

Once FICTITIOUS INC has gathered some audio data, they can test a speech-to-text model.

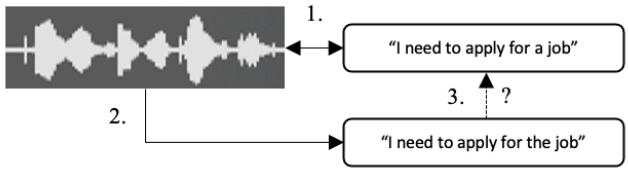
12.2 Testing the speech-to-text model

Most virtual assistant platforms integrate with one or more speech-to-text engines. These engines often come with default, pre-trained models. These default models are generally trained on generic text. Most of FICTITIOUS INC's terminology is generic – lots of call centers deal with password resets, appointments, and employment inquiries. A default model may suffice for FICTITIOUS INC. Even if a generic model is not good enough, FICTITIOUS INC should test against a generic model before training their own custom model – the generic model is a baseline to compare any custom model against.

Test data volume

In theory, you may want to test your model against the largest possible data set. In practice, it's good to use 20% of your audio data for testing your model and use the other 80% for training a model (if training is needed).

FICTITIOUS INC will build a test set from their audio data. Each test data is a pair of an audio file and the correct transcription of that audio file. Each test data will be tested with the process depicted in Figure 6.



1. The audio file and correct transcription are collected.
2. The speech model transcribes the audio file.
3. The model's transcription is tested by comparing it to the correct transcription.

Figure 6 Testing a speech-to-text model on a single audio file.

For each piece of test data, the speech-to-text model will transcribe the audio. Each test data pair becomes a triple, with the addition of the model's transcription. An exemplary output from a speech-to-text model test is shown in Table 6.

Table 6 A speech-to-text model is evaluated by comparing the model's transcription to the correct transcription of the audio file.

Audio file	Expected (Correct) Transcription	Actual Model Transcription
`Password.wav`	"reset my password"	"reset my password"
`EmploymentInquiry.wav`	"I need to apply for a job"	"I need to apply for the job"
`LoyaltyPoints.wav`	"how many Fiction Bucks do I have?"	"how many fiction bookstores have?

Table 6 is just raw data. The output can be manually inspected, but it's not clear what the data tells us. Does this speech model work well or not? How much will the speech model contribute to (or detract from) FICTITIOUS INC's success metrics? This is difficult to determine when manually inspecting transcriptions for accuracy. An automated analysis of speech accuracy is both more valuable and easier to tie to success metrics.

There are three metrics commonly used to evaluate speech models:

- **Word Error Rate (WER)**: evaluates number of words with a transcription error
- **Intent Error Rate (IER)**: evaluates the number of intent detection errors caused by a transcription error
- **Sentence Error Rate (SER)**: evaluates number of sentences containing a transcription error

Let's start with Word Error Rate.

12.2.1 Word Error Rate

The simplest way to measure speech transcription accuracy is by evaluating the *Word Error Rate*. The Word Error Rate is a ratio of the number of incorrect words in the model's

transcript divided by the number of words in the correct transcript. FICTITIOUS INC's speech data is evaluated for Word Error Rate in Table 7.

Table 7 Word errors in FICTITIOUS INC's test data. Each word error is bolded.

Expected (Correct) Transcription	Actual Model Transcription	Word Error Rate (WER)
"reset my password"	"reset my password"	0% (0 of 3)
"I need to apply for **a** job"	"I need to apply for **the** job"	14.3% (1 of 7)
"how many Fiction **Bucks do I** have?"	"how many fiction **bookstores** have?	42.9% (3 of 7)
		Total: 23.5% (4 of 17)

There are three types of word errors:

- Substitution: The model's transcription replaces a correct word with a different word.
- Insertion: The model's transcription adds a word that was not in the correct transcript.
- Deletion: The model's transcription removes a word that was in the correct transcript.

The phrase "I need to apply for a job" had one substitution error ("the" for "a"). The phrase "how many Fiction Bucks do I have" had one substitution error ("bookstore" for "Bucks") and two deletion errors ("do" and "I").

Speech model providers often make it easy to compute Word Error Rate because it can be computed generically – just by counting errors. For FICTITIOUS INC, the Word Error Rate is missing important context. There's no clear way to tie the 23.5% Word Error Rate to containment. Some of the errors are trivial ("the" vs "a"), some of the errors seem important ("bookstore" vs "Bucks do I"). FICTITIOUS INC should tie all analysis back to success metrics. How does the Word Error Rate affect containment?

(Containment is the percentage of conversations that are not escalated out of the virtual assistant. When a virtual assistant handles a conversation from beginning to end, that conversation is "contained" within the virtual assistant.)

FICTITIOUS INC cannot draw a straight line from Word Error Rate to containment, but they can infer one by inspecting the word errors. Rather than counting errors generically, they can count errors by word. From this, they can infer if the speech-to-text model is making mistakes on words likely to affect the use case. This new view on word errors is shown in Table 8.

Table 8 FICTITIOUS INC word errors by word.

Word	Errors	Total Occurrences	Error Rate
Bucks	1	1	100%
do	1	1	100%
a	1	1	100%
I	1	2	50%

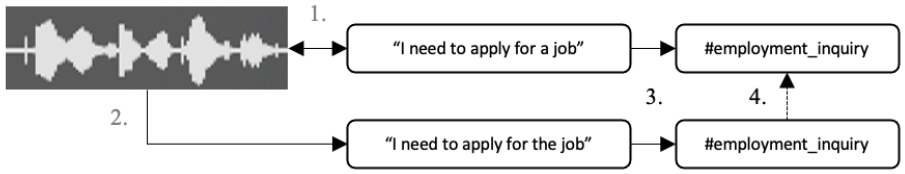
(All other words)	0	12	0%

Table 8 has a very small data set but suggests that the speech-to-text model will not work well when FICTITIOUS INC callers ask about their Fiction Bucks balance. The error rates on "do", "a", and "I" are less likely to impact the virtual assistant. The classifier in the assistant should be resilient to minor errors with these common words.

FICTITIOUS INC does not need to guess about the impact of word errors on their virtual assistant's classifier. They can measure the impact directly by evaluating an Intent Error Rate.

12.2.2 Intent Error Rate

FICTITIOUS INC's success metric is to contain calls and complete them successfully. An important part of successfully containing a conversation is identifying the correct user intent. Speech transcription errors are not a problem if the virtual assistant identifies the correct intent for the user. FICTITIOUS INC can compute an Intent Error Rate through the process diagrammed in Figure 7.



1. The audio file and correct transcription are collected.
2. The speech model transcribes the audio file.
3. Both transcriptions are classified by the virtual assistant into intents.
4. The intents are compared.

Figure 7 Computing a speech-to-text model's intent error rate for a single piece of speech data.

FICTITIOUS INC's Intent Error Rate is computed in Table 9.

Table 9 Intent errors in FICTITIOUS INC's test data. Each word error is bolded. The expected and actual transcriptions are each classified, then the expected and predicted intents are compared.

Expected (Correct) Transcription	Actual Model Transcription	Expected Intent	Predicted Intent	Intent Error Rate (IER)
"reset my password"	"reset my password"	`#reset_password`	`#reset_password`	0%
"I need to apply for **a** job"	"I need to apply for **the** job"	`#employment_inquiry`	`#employment_inquiry`	0%

"how many Fiction **Bucks do I** have?"	"how many fiction **bookstores** have?	`#loyalty_points`	`Unknown`	100%
				Total: 33.3%

The Intent Error Rate puts the performance of the speech-to-text model into context. FICTITIOUS INC had to infer what a 23.5% Word Error Rate meant for their users. The Intent Error Rate does not need any inference. A 33.3% Intent Error Rate means that the virtual assistant will fail to predict the user's intent correctly 33.3% of the time.

Representative data alert!

The Intent Error Rate is directly usable as shown above if the speech data is representative of production usage. If production data has a different distribution of intents or user demographics, the Intent Error Rate will be skewed.

Intent Error Rate is a great way to evaluate how a speech-to-text model impacts the virtual assistant's intent identification. When the wrong intent is predicted, several success metrics go down including user satisfaction and call containment. A low Intent Error Rate is important.

Not every message in a conversation includes an intent. The Intent Error Rate alone is not sufficient to evaluate the impact of a speech-to-text model on a virtual assistant. Let's look at one more metric: Sentence Error Rate.

12.2.3 Sentence Error Rate

For audio segments that are not expected to contain intents, FICTITIOUS INC can evaluate a speech-to-text model using Sentence Error Rate. *Sentence Error Rate* is the ratio of sentences with an error compared to the total number of sentences. Sentence Error Rate is a good metric to use when an entire string must be transcribed correctly for the system to succeed. We have seen that intent statements do not need to be transcribed 100% accurately for the system to succeed – the virtual assistant can still find the right intent if some words are inaccurately transcribed.

We can stretch the definition of a sentence to include any standalone statement a user will make. For instance, in FICTITIOUS INC's password reset flow, they ask for the user's date of birth. The date of birth can be treated as a full sentence. If the speech-to-text model makes a single mistake in transcribing the date of birth, the entire date will be transcribed wrong. Table 10 shows a computation of Sentence Error Rate for FICTITIOUS INC audio files containing dates.

Table 10 Sentence errors in FICTITIOUS INC's test data for dates. Each word error is bolded. Each transcription only contains a single sentence.

Expected (Correct) Transcription	Actual Model Transcription	Sentence Error Rate (SER)
"January first two thousand five"	"January first two thousand five"	0% (No error)
"**One** eight nineteen **sixty-three**"	"June eight nineteen sixteen"	100% (Error)

"**seven** four twenty oh one"	"eleven four twenty oh one"	100% (Error)
		Total: 66.7%

Sentence Error Rate is a good metric for evaluating any conversational input that must be transcribed exactly correctly. FICTITIOUS INC's password reset flow collected and validated a User ID, date of birth, and an answer to a security question. They should not execute a password reset process if they are not perfectly sure they are resetting the right user's password, so transcribing these responses accurately is important.

In FICTITIOUS INC's appointments process flow, they collect a date and time for the appointment. Again, these data points must be captured exactly correctly, or the user may be given a completely different appointment than they expected! Virtual assistants commonly collect data that must be captured correctly. The virtual assistant can always ask the user to confirm any data point before the virtual assistant proceeds, but any "sentence errors" will prolong the conversation as the user is prompted to repeat themselves.

Sentence Error Rate computation

For some data inputs, you may run a post-processing step on the speech-to-text model transcription before comparing it to the correct transcription. For instance, in numeric inputs "for" and "four" should both be treated equivalently as the numeral "4". Similarly, "to" and "too" are equivalent to "two".

Your speech-to-text model may enable this automatically. This functionality is sometimes called "smart formatting" or "recognition hints".

Once FICTITIOUS INC has evaluated the performance of their speech model, they can decide if the performance is sufficient, or if they need to train a custom model to get even better results. They now have a baseline to compare their custom model against. Let's explore how they can train their own custom speech to text model.

12.3 Training a speech-to-text model

Most virtual assistant platforms integrate with speech engines, and most of these speech engines support customized training. The major speech engine providers offer differing levels of customization. FICTITIOUS INC should target the minimum level of customization that meets their needs. Speech training has diminishing returns, and some levels of customization take hours or days to train.

Before FICTITIOUS INC does any training, they should select an appropriate "base" model for their use case. Base models are available for use without any custom training at all and come in multiple flavors. FICTITIOUS INC is starting with English, but they will likely have a choice between US English, UK English, or Australian English. Within a language and dialect, FICTITIOUS INC may have choices between models optimized for audio coming from telephone, mobile, or video. The choices are summarized in Figure 8.

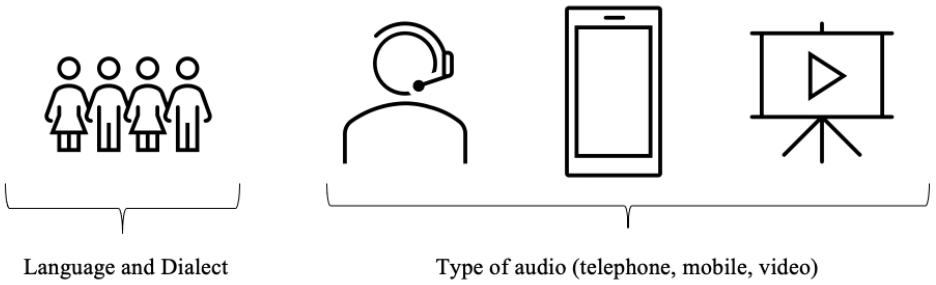


Figure 8 Speech platforms offer several types of base models, optimized for different languages/dialects as well as different audio channels. Choose the base model most suitable for your application.

Why so many choices in base models?

Audio data is encoded differently in different technology applications. Narrowband technologies (like a telephone network) compress the audio signal to a "narrow" range of frequencies. Broadband technology uses a wider range of frequencies, producing a higher quality audio. Like any model, speech-to-text models need to be trained on representative data. Be sure to match your use case to the right base model.

The base models are trained by the speech platform provider with data that they own. Each base model is generally trained with language and audio data from a variety of different users, both native and non-native speakers. Speech platform providers are making strides towards producing models with less bias, but FICTITIOUS INC should definitely test their model against a representative user set to verify this.

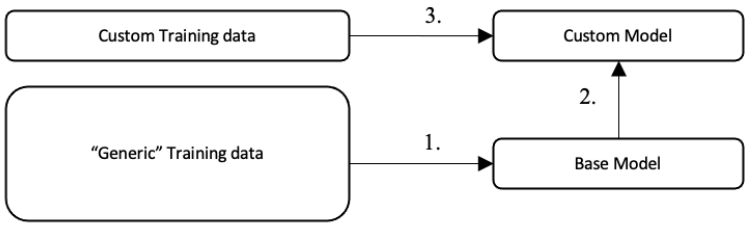
Terminology alert!

Speech-to-text providers use differing terminology. Technically, custom speech models are adaptations of base models. In this view, base models are trained, and custom models are "adapted" or "customized" rather than trained.

Speech adaptation is the process of building a new custom model that extends a base model with custom data.

For the sake of simplifying our mental model of virtual assistants, this chapter will use the simpler term "training", which is close enough for our purposes.

After FICTITIOUS INC has selected a base model, they can begin training a custom model. The custom model is an adaptation or an extension of a base model. The overview of a custom model training process is shown in Figure 9.



1. Select a base model, trained by the speech platform.
2. Create a custom model that extends a base model.
3. Train the custom model with custom data.

Figure 9 Summary of the custom speech-to-text model training process.

FICTITIOUS INC will generally not have access to the "generic" training data – this data is usually owned by the speech platform. The platform will expose a base model that FICTITIOUS INC can extend into a custom model. FICTITIOUS INC will train that custom model with their specific data.

Training data size note

In this section, we will use very small training data sets to illustrate the key concepts. In practice, FICTITIOUS INC would use several hours' worth of data to train their custom models.

Depending on their speech platform provider, FICTITIOUS INC will have three customization options available to them: language models, acoustic models, and grammars.

- A *`language model`* is a collection of text utterances that the speech engine is likely to hear.
- An *`acoustic model`* is a speech-to-text model that is trained with both audio and text.
- A *`grammar`* is a set of rules or patterns that a speech-to-text model uses to transcribe an audio signal into text.

FICTITIOUS INC can use different training options for different parts of their virtual assistant. Let's start with the simplest option: language models.

12.3.1 Custom Training with a Language model

Training a custom language model is the simplest form of speech training that FICTITIOUS INC can do for their virtual assistant. A *language model* is a collection of text utterances that the speech engine is likely to hear. FICTITIOUS INC can build a text file from one or more of these sources: transcriptions of call recordings, or intent and entity training examples in their virtual assistant. This text file is the training data for the language model. FICTITIOUS INC's language model training data file is depicted in Table 11.

Table 11 FICTITIOUS INC's language model training data.

language_model_training_data.txt
reset my password
I need to apply for a job
how many Fiction Bucks do I have?

Table 11 is the only table in this book that shows a one-column training data table. This is not a mistake. Language models use unsupervised learning. This means they are only given inputs – no outputs are specified anywhere in the training data. A language model is trained by teaching it how to read the specific language in the domain it is trained on.

Children are often taught how to read by learning letters and the phonetic sounds they make. For instance, the English language contains 42 distinct phonemes. Once you master the 42 phonemes, you can sound out almost any English word (even if it takes a while). Mastery of the phonemes lets you pronounce words you have never encountered before.

A language model reads a training data file in a very similar way. An example is shown in Figure 10. The base model in the speech platform is trained how to phonetically read. In the figure, we see how the language model breaks the phrase "How many Fiction Bucks do I have" into phonemes.

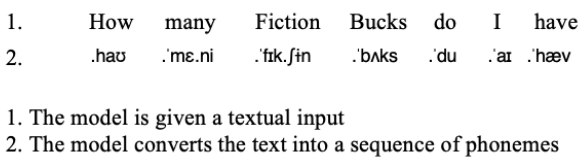


Figure 10 A language model learns phonetic sequences from textual input.

A speech-to-text model transcribes an audio signal into a sequence of phonemes. Most speech engines generate several transcription hypotheses: a primary hypothesis and one or more alternate hypotheses. The **base** model from a speech platform may transcribe the audio signal in "How many Fiction Bucks do I have" as shown in Figure 11.

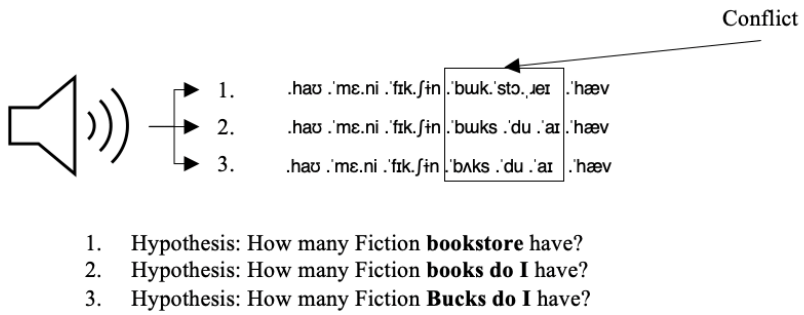
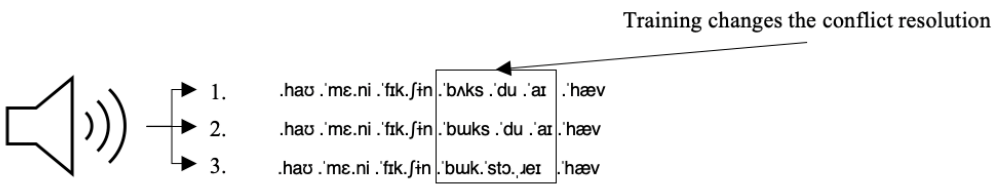


Figure 11 A speech-to-text model generates several alternative hypotheses for the audio rendition of "How many Fiction Bucks do I have?". This base model does not immediately recognize the FICTITIOUS INC term

"Fiction Bucks".

"Books" and "bucks" are phonetically similar. A base model is likely trained on data that includes the word "books" (`ˈbʊks`) more than the word "bucks" (`ˈbʌks`). This is especially true when we consider the sound in context. There are many fewer instances of the phrase "Fiction Bucks" than the phrase "fiction books" – the former is only used at FICTITIOUS INC, but the latter is used all over the world. Given a difficult choice between two phonetically similar phrases, the speech-to-text model chose the phrase it had seen the most before.

When FICTITIOUS INC adds language model training to their speech-to-text model, the speech-to-text model gives higher precedence to FICTITIOUS INC's specific language. FICTITIOUS INC's language model training data includes their domain-specific term "Fiction Bucks". This encourages the model to transcribe "bucks" (`ˈbʌks`) instead of "books" (`ˈbʊks`), especially when adjacent to the word "Fiction". Figure 12 shows how the **custom** speech-to-text model works after FICTITIOUS INC trains it with a language model.



1. Hypothesis: How many Fiction **Bucks do I** have?
2. Hypothesis: How many Fiction **books do I** have?
3. Hypothesis: How many Fiction **bookstore** have?

Figure 12 A speech-to-text model generates several alternative hypotheses for the audio rendition of "How many Fiction Bucks do I have?". Language model training helps the speech-to-text model select the best alternative from similar-sounding phrases.

Language model training is perhaps the quickest way to improve speech transcription. The training is fast because it only uses textual data. Even better, this text data should already be available from the utterances used for intent and entity training! Audio data is still required to test the model after it is trained. Still, language models are not a panacea.

Check your speech platform to see if they offer language models

Several of the speech-to-text providers surveyed for this book included language model functionality, but some do not. The most common terminology is "language model" but some providers refer to it as "related text". Check your platform's documentation.

Some speech-to-text platforms offer a "light" version of language models called "hints" or "keywords". Rather than training a full custom language model, a speech transcription request can be accompanied by a list of words that may be expected in the audio being transcribed.

Some speech platforms do not offer language model training at all. These platforms can only be trained with pairs of audio files and their transcripts. Language models also rely on the speech platform's built-in phonetic reading of words. In a language model, you cannot show the speech platform exactly how a word sounds with an audio example; you can only train the model on what words are likely to occur in relation to other words. Because the model is not trained with audio, it may fail to understand words pronounced with various accents, especially for domain-specific and uncommon words.

The benefits and disadvantages of language models are summarized in Table 12.

Table 12 Benefits and disadvantages of language models

Benefits	Disadvantages
• Training is relatively fast. • Does not require any audio to *train* a model (audio is still required to test). • Can reuse intent and entity training utterances as language model training data.	• Not all speech platforms offer language models. • Relies on the platform's built-in phonetic reading of words – does not explicitly train the model how these words sound. • May be insufficient if the user base has a wide variety of accents.

Language models are a great option if your platform provides them, but not every platform does. Let's look at the most common speech-to-text model training option: the acoustic model.

12.3.2 Custom Training with an Acoustic model

FICTITIOUS INC has collected audio files and their associated text transcripts. This is exactly the fuel required to train an acoustic model. An *acoustic model* is a speech-to-text model that is trained with both audio and text. An acoustic model generally uses supervised learning with the audio as an input and the text as the "answer". When this model is trained, it learns to associate the acoustic sounds in the audio with the words and phrases in the text transcripts.

Terminology alert!

Many speech platforms do not give a special name to acoustic models. They may just call them "custom models". An acoustic model is trained with both audio and text. This is the default training mode in many speech platforms and the only training possibility in some speech platforms.

An acoustic model is an especially good option for training a speech-to-text model to recognize unusual words across a variety of accents. By pairing an audio and a text transcript together, the speech-to-text model directly learns what phonetic sequences sound like and how they should be transcribed.

The phrase "Fiction Bucks" is specific to FICTITIOUS INC's jargon and a base speech-to-text model probably won't have ever encountered that phrase. FICTITIOUS INC likely needs

to train the speech-to-text model to recognize it. Figure 13 shows a variety of ways the phrase "How many Fiction Bucks do I have" may sound across a variety of accents.

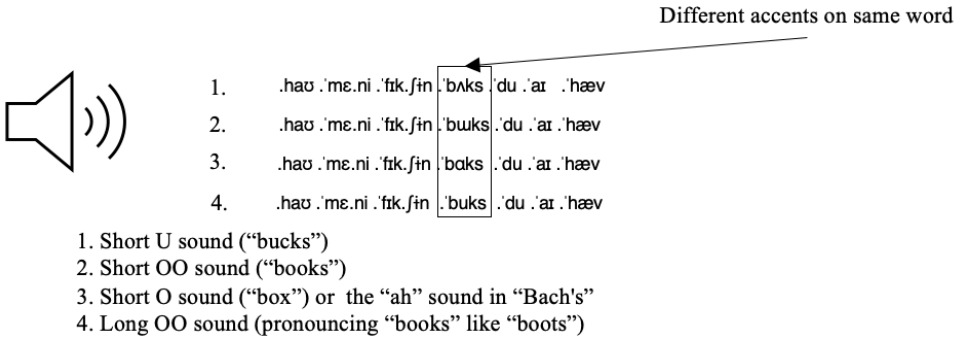


Figure 13 Phonetic sequences for "How many Fiction Bucks do I have" in several different accents. An acoustic model can be trained to transcribe each of these the same way.

When FICTITIOUS INC samples audio from a variety of user demographics, they may get some or all of those pronunciations of "Fiction Bucks". They can train an acoustic model with each of these pronunciations and the model will learn to transcribe each of them to the same phrase, even though they sound different phonetically.

Acoustic models generally use a supervised learning approach. This means the speech-to-text model is directly trained to associate audio signals to text transcripts. The model learns to identify variations of phonetic sequences that should get identical transcriptions. Acoustic model training is also useful for training a model to recognize uncommon words or words with unusual spelling. Acoustic training is also useful for "loan words" – words borrowed from another language. If a speech engine does not know how to pronounce a word, it is likely to need acoustic training to transcribe that word.

Save some data for testing the model!

Acoustic models need to be trained with a lot of data – five hours minimum, ten to twenty hours is better. But don't use all of your data for training the model. Optimally, use 80% of the data for training and save 20% for testing. You want to be able to test your speech-to-text model against data it was not explicitly trained on. That 20% of the data becomes the "blind test set" for your speech-to-text model.

Acoustic models are widely available on speech platforms and are sometimes the only option provided in a platform. The primary benefit of acoustic models is that they are given a direct translation guide from audio to phonemes to text via their audio and text transcript training data. Acoustic models are the best option for training a speech-to-text model to recognize different accents.

Acoustic models have some downsides too. Acoustic models often require a large amount of training data to be effective – a minimum of five hours is generally required, with ten or

twenty hours being preferable. This volume of data is computationally expensive to process. Acoustic model training generally takes hours to complete, and sometimes takes days.

The benefits and disadvantages of acoustic models are summarized in Table 13.

Table 13 Benefits and disadvantages of acoustic models

Benefits	Disadvantages
• Default training option for most platforms. • Good when the user base has demographic diversity and varying accents. • Trains the model on the exact sounds made by words and phrases.	• Training time can be lengthy – up to a day or two in some cases. • Can require a large volume of data (five hours minimum, ten to twenty preferred)

Language models and acoustic models are both good options for training a speech-to-text model to recognize open-ended language. "How can I help you" is an open-ended question and will have a wide variety of responses. This is why we usually train a language or acoustic model on intent-related utterances.

Sometimes a virtual assistant asks a constrained question, where the structure of the expected input is known. When the virtual assistant asks, "What's your date of birth", the response is very likely to be a date. Some speech platforms offer a training option that constrains the transcription options. Let's explore this option.

12.3.3 Custom Training with Grammars

A *grammar* is a set of rules or patterns that a speech-to-text model uses to transcribe an audio signal into text. FICTITIOUS INC can use a grammar anywhere in their virtual assistant where the expected user utterances fit a finite set of patterns. FICTITIOUS INC's first possibility for grammars is in collecting user IDs. A FICTITIOUS INC user ID is a series of four to twelve letters followed by one to three numbers. This can be coded in the regular expression `[A-Za-z]{4,12}[0-9]{1,3}` as shown in Figure 14.

Terminology alert!

Every speech platform I investigate that offered this functionality called it "grammar". Hooray for standardization!

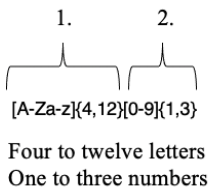


Figure 14 Regular expression for FICTITIOUS INC's user ID pattern. Their user IDs are always four to twelve letters followed by one to three numbers.

The expected range of FICTITIOUS INC user IDs is small enough that it fits in a 24-character regular expression. FICTITIOUS INC can use a grammar to recognize user IDs. The speech-to-text model learns several rules from the grammar including the following:

- The phoneme `ˈtu` means "two", not "to" or "too"
- The phoneme `ˈfɔɹ` means "four", not "for" or "fore"
- "M" is possible as `ˈɛm` – never use the similar-sounding phonemes for "him" `ˈhɪm` or "hem" `ˈhɛm`. The model will avoid other words starting with soft "h" sounds. ("A" vs "hay", "O" vs "ho", to name just a few.)
- The phoneme for "deal" `ˈdil` is better interpreted as "D L" `ˈdi ˈɛl`.

The grammar for FICTITIOUS INC's user ID pattern packs a lot of training in a small space! FICTITIOUS INC also collects dates several times in their virtual assistant: date of birth (in the password reset flow) and date of appointment (in the create appointment flow). There are a large number of possible dates, but ultimately dates can be captured in a finite set of rules. Figure 15 shows 36 possible month formats and 62 possible day formats. (There's actually a few more, callers can use "oh", "zero", and "aught" interchangeably.)

Month is one of these

January	One	Oh One
February	Two	Oh Two
March	Three	Oh Three
April	Four	Oh Four
May	Five	Oh Five
June	Six	Oh Six
July	Seven	Oh Seven
August	Eight	Oh Eight
September	Nine	Oh Nine
October	Ten	One Oh
November	Eleven	One One
December	Twelve	One Two

Day is one of these

One	Thirteen	Twenty-five	Oh One	One One	Two One
Two	Fourteen	Twenty-six	Oh Two	One Two	Two Two
Three	Fifteen	Twenty-seven	Oh Three	One Three	Two Three
Four	Sixteen	Twenty-eight	Oh Four	One Four	Two Four
Five	Seventeen	Twenty-nine	Oh Five	One Five	Two Five
Six	Eighteen	Thirty	Oh Six	One Six	Two Six
Seven	Nineteen	Thirty-one	Oh Seven	One Seven	Two Seven
Eight	Twenty		Oh Eight	One Eight	Two Eight
Nine	Twenty-one		Oh Nine	One Nine	Two Nine
Ten	Twenty-two			One Oh	Two Oh
Eleven	Twenty-three		Three Oh		
Twelve	Twenty-four		Three One		

Figure 15 There are a finite number of ways to give a date in "month-day" format. (There are a finite number of ways to give a year too.) In fact, some users prefer to say "zero" or "aught" instead of "oh" in their dates.

EXERCISE 1

Can you build a list of possible year formats?

Date formats

There are multiple date formats including month-day-year, day-month-year, and year-month-day. It's impossible to write a grammar which covers all three at once (how would you unambiguously transcribe "oh one oh two twenty oh one"?) Pick one.

Month-day-year is the primary pattern in the United States but not in most of the world. If your user base is broad enough that they might use multiple formats, be sure to give the user a hint. FICTITIOUS INC could ask "What is your month, day, and year of birth?" instead of "What is your date of birth?"

The number of ways to express a date within a given date format is large but finite. Still, it is a frightening proposition to write a single regular expression to handle all of the possibilities within a date format. Most speech platforms that have grammars offer some sort of rules engine which can break the work into a manageable number of pieces. Code Listing 1 shows how to code a set of rules to match date phrases.

Code Listing 1: Pseudocode for month-day rules engine

```
$date_phrase = $month and $day Annotation1: Complex rules can be broken down into simpler
       rules. A date is a month and a year
$month = $month_name or $month_number Annotation2: There are two major patterns to months,
       either the name of the month or the number
$month_name = January or February or March or April or May or June or July or August or
       September or October or November or December
$month_number = $one or $two or $three or $four or $five or $six or $seven or $eight or
       $nine or $ten or $eleven or $twelve Annotation3: We first capture the twelve numeric
       variations
$zero = zero or oh or aught Annotation4: The "zero" logic is captured once and reused
$one = one or $zero one
$two = two or $zero two
$three = three or $zero three
$four = four or $zero four
$five = one or $zero five
$six = one or $zero six
$seven = one or $zero seven
$eight = one or $zero eight
$nine = one or $zero nine
$ten = ten or one $zero
$eleven = eleven or one one
$twelve = twelve or one two

# $day is left as an exercise for the reader
```

EXERCISE 2

Can you build a grammar for $day?

Most speech platforms with grammars use strict interpretation when applying a grammar. This means they try very hard to "force fit" an audio signal into the specified pattern – and if the audio doesn't fit the pattern, they may not return a transcription at all. This is in stark contrast with language models and acoustic models, which will happily attempt to transcribe any input. FICTITIOUS INC's user ID regular expression `[A-Za-z]{4,12}[0-9]{1,3}` is strict. If the user says, "I don't know" or "agent", the speech engine can't force fit that into the grammar and may not return a transcription at all. Code Listing 2 shows one way to make the user ID grammar more forgiving and lenient so that it can transcribe more phrases than just valid user IDs.

Code Listing 2: Pseudocode for a more forgiving and lenient user ID grammar

```
$user_id = $valid_user_id or $do_not_know or $opt_out
$valid_user_id = regular_expression("[A-Za-z]{4,12}[0-9]{1,3}")
$do_not_know = "I don't know" or "I don't have it" or "No idea"
$opt_out = "agent" or "representative" or "customer service" or "get me out of here"
```

Grammars come with significant tradeoffs. The strict nature of a grammar allows the speech-to-text model to more accurately transcribe inputs that match an expected pattern. The strictness has a cost when it inputs do not match the expected pattern. This strictness can be alleviated by encoding additional patterns into the grammar, but this makes the grammar more complex and difficult to manage.

Grammars are relatively quick to train. No audio input is required for training; only an encoding of the grammar rules is needed. Most speech platforms with grammars allow you to build complex rule patterns into the grammar itself. Some platforms let you combine grammars with language and/or acoustic models.

Grammars are not suitable when the range of expected inputs does is unconstrained. FICTITIOUS INC cannot use a grammar to collect responses to their open-ended question "How may I help you?"

The benefits and disadvantages of grammars are summarized in Table 14.

Table 14 Benefits and disadvantages of grammars

Benefits	Disadvantages
• Most accurate method for capturing constrained inputs. • Useful when the input follows a set of rules. • No audio required for training – only rules.	• Only suitable when the expected input is constrained. • May fail to transcribe unexpected responses or digressions.

Create your own grammar, if your speech platform does not provide one

If you can write code that is executed between speech-to-text transcription and before the virtual assistant acts, you can write your own "grammar" in code. Your "grammar" will actually be a post-processor. For instance, when the speech-to-text model returns "January to for twenty ten" your post-processor can replace the "to"/"for" with numbers, giving "January two four twenty ten".

A general comparison of language models, acoustic models, and grammars is found in Table 15.

Table 15 Comparison of language model, acoustic model, and grammar

Feature	Language Model	Acoustic Model	Grammar
Availability in speech platforms	Sometimes	Always	Sometimes

Trained with	Text only	Audio and text pairs	Rules only
Tested with	Audio and text	Audio and text	Audio and text
Training time	Minutes	Hours	Seconds
Open-ended questions	Works well	Works well	Does not work
Constrained questions	Works well	Works well	Works best
Training method	Unsupervised learning	Supervised learning	Rules
If the user says something unexpected	Works	Works	Does not work well
Ability to handle varying accents	Good	Best	Mostly good

FICTITIOUS INC can train multiple speech-to-text models for different parts of their assistant. The model can default to using a language model and/or an acoustic model and use a grammar in specific parts of the conversation (asking for user IDs and dates). Just as FICTITIOUS INC plans to iteratively train and improve their intent classifier, they should expect to iteratively train and improve their speech-to-text model(s).

12.4 Summary

- Speech-to-text models transcribe from audio to text.
- Virtual assistants can be resilient to some transcription mistakes. Don't evaluate a speech-to-text model based on pure accuracy; evaluate it based on how transcription mistakes affect the virtual assistant. Intent Error Rate always affects the assistant, Word Error Rate does not.
- Testing a speech-to-text model always requires audio with matching transcripts. Language models are trained only with text. Acoustic models are trained with audio and text. Grammar models are programmed with rules.
- Language models and acoustic models are best for transcribing open-ended inputs and can also transcribe constrained inputs. Grammars are best at transcribing constrained inputs and are completely unsuitable for open-ended inputs.