Doc.stxt

Get Started

Welcome to Bland AI

Bland is a platform for AI phone calling. Using our API, you can easily send or receive phone calls with a programmable voice agent.

We really care about making our phone calls…

Fast: sub-second latency from person speaking to AI responding.

Reliable: it’s our responsibility, day in and day out, to make sure your phone calls work. No exceptions.

Ultra flexible: configure all aspects of your agent’s behavior, by settings it’s voice, creating transfer scenarios, configuring the initial greeting, etc.

​

What you can do with Bland

Send phone calls

Dispatch AI phone calls to call customers, leads, and to streamline operations.

Set up inbound numbers

Create inbound phone numbers for customer support, etc.

Do live function calling

Connect external APIs and take live actions during phone calls.

Extract structured data from calls

Extract JSON data to answer questions about your calls.

Create call campaigns

Simultaneously send thousands of calls at once.

Fine-tune a custom LLM

Fine-tune a custom LLM using your enterprise’ call recordings and transcripts.

​

Getting started

API Reference

Read the API reference.

Starter guide

Learn to send your first phone call and test your agents.

Get your keys

Sign up on the developer portal.

Enterprise

Learn about enterprise features & meet with a member of the Bland AI team.

Get Started

Starter guide

Learn how to use Bland’s API in under five minutes

​

Creating a developer account

To get started, sign up on the developer portal.

Enter your phone number and verification code. Finally, once your developer portal loads, go to the Send phone call page.

​

Sending your first phone call

Although Bland is an API-first platform, the send phone call page provides a simple interface for quickly testing calls. On the left side you can adjust the call options and on the right hand side you can see how the code updates.

Once you’re satisfied with a call, copy the code on the right side (in Javascript, Python, or cURL) and add it to your application.

1

Enter your phone number

In the Phone Number field, enter your own phone number.

2

Write your prompt

For the task box either select one of the example prompts or write your own. For more instrucionts about prompting your AI phone agent, read this blog post.

3

Send the call

Scroll to the bottom of the page, and press the Send call button. Note, calls are charged at $0.12/minute, billed to the exact second.

To send a phone call programatically, read the API reference.

​

Testing your phone agent

Once you’ve sent your first phone call, the next step is to test and improve the outputs from your phone agent.

One way to test your agent is to send yourself test calls. A faster way, however, is to use the Bland AI testing suite.

1

Set up the test suite

Select the model and language and insert your current prompt into the task box.

2

Write a message

Start messaging your phone agent. Act like you’re the person receiving the call, and purposefully ask edge-case questions to throw the phone agent off.

3

Update the prompt

Based on the responses you receive, update the instructions in the prompt.

​

Next steps

You now know how to send and test phone calls, but you’ve only scratched the surface of Bland’s capabilties.

Areas for further exploration:

Bland Enterprise

Custom Twilio Integration

Connect Bland to your own Twilio account

​

Overview

Enterprise customers can connect their own Twilio account to Bland. Easily bring over your existing phone numbers, integrations, and more.

Pre-requisites:

Your own Twilio account

​

Step 1: Creating an Encrypted Key with your Twilio Credentials

Go to your Twilio Console and get your Account SID and Auth Token.

Create an encrypted\_key by sending an API request to Bland.

This is the only time that your encrypted\_key will be returned to you. Make sure to store it securely, and new keys will need to be generated if lost.

​

Step 2: Using the Encrypted Key in Outbound Calls

Include encrypted\_key in the headers (in addition to the Authorization header) of your API requests, and we’ll use that account’s credentials to make the call.

For example:

{

"Authorization": "BLAND\_API\_KEY",

"encrypted\_key": "YOUR\_ENCRYPTED\_KEY"

}

Note:

You can set your from number in the API request - this will need to be a number owned by that Twilio account (and not one purchased through Bland).

By default, we’ll send calls from a randomly selected number in the specified Twilio account if a from is not specified.

​

Step 3: Uploading Inbound numbers

Go to your Twilio Console and get your Twilio phone number(s).

Upload your numbers through the API.

We’ll validate that these numbers are owned by that account and add them to your Bland account.

​

Step 4: Configuring Inbound Numbers/Webhooks

Note: When updating inbound numbers, the headers need to include the encrypted\_key in addition to the Authorization header. Doing so makes sure the updates are applied to the correct Twilio account.

Once you update an inbound number through the Dev Portal or API, that number will be automatically configured to run on Bland’s infrastructure. No additional steps are required!

Bland Enterprise

Fine-tuning & Custom LLMs

​

Summary

Bland will fine-tune a custom model for your enterprise using transcripts from succesful prior calls. Then Bland will host that LLM and provided dedicated infrastrucure to enable phone conversations with sub-second latency.

Bland will also enable you to connect to a custom LLM & will host that LLM to drive latency down further.

Get in touch here.

​

Background on fine-tuning

Traditionally, most AI phone agents use private models from companies like OpenAI and Anthropic. Those LLMs are large, and perform best at following instructions and delivering high quality outputs. The downside, however, is they are very slow. Additionally, because they’re general models, their personality, tone, and overall capabilities are limited.

Conversely, open source models generally perform worse at a broad range of tasks. However, by fine-tuning an open-source model with examples of a given task, you can significantly improve it’s performance at that task, even surpassing the capabilties of top-of-the-line models like GPT-4.

​

How do I fine-tune with Bland?

To inquire about fine-tuning connect with the Bland team here.

During the initial conversation you will discuss:

The format of data we’ll require

How long fine-tuning will take (typically one week)

Pricing (typically under five figures)

​

How do I bring my own LLM?

The Bland team will advise on connection method, requirements for the connection, etc.

Typically takes under 24 hours to set up after kickoff.

Inquire here.

Bland Enterprise

Custom languages & voices

Enterprise customers can bring their own TTS service or work with Bland’s engineering team to configure higher quality voice clones.

Enterprise customers can also request foreign languages like German, Spanish, Italian, and Portoguese, and Bland’s engineering team will set up different transcription and TTS to accomodate the request.

Reach out for enterprise access, here.

Bland Enterprise

Bring your own Twilio

Enterprise customers can create and connect their own Twilio account to Bland.

Features include:

Full ownership of telephony infrastructure

Ability to connect to existing telephony infrastructure

Closer control of spam risk and any discounted rates already included on the account

Reach out for enterprise access, here.

Bland Enterprise

Enterprise rate limits

By default, Bland customers can dispatch 1000 calls/day before hitting rate limits.

Enterprise customers start at 20,000 calls per hour, and 100,000 calls per day.

To raise your rate limits or discuss limits larger than what’s offered on enterprise, reach out here.

Featured guides

Bland & Botpress

​

Overview

The integration process involves several key steps, starting with setting up your Botpress server and configuring your Bland AI account to ensure both platforms can communicate effectively. You’ll learn how to authenticate your Botpress instance with Bland AI, send dynamic data (like phone numbers and intents) from chatbot conversations to Bland AI, and initiate voice calls based on user inputs or specific triggers within your chatbot workflows. This guide is also a great way for no-coders to deploy a bland AI project easily, with little to no coding experience.

This guide will cover:

Setting Up Botpress: Instructions on preparing your Botpress environment for integration, including installation and basic configuration.

Configuring Bland AI: How to set up your Bland AI account, obtain necessary API keys, and understand the API’s capabilities related to voice interactions.

Integration Workflow: Step-by-step guidance on creating workflows in Botpress that interact with Bland AI’s API, focusing on initiating voice calls to user-provided numbers.

Testing and Troubleshooting: Tips for testing your integrated solution to ensure it works as expected and troubleshooting common issues that may arise during the integration process.

Advanced Use Cases: Ideas and examples for leveraging this integration to create innovative chatbot applications that blend chat and voice interactions in unique and valuable ways.

​

Getting started

Read the entire guide.

​

About Nort Labs

Nort Labs is an agency that leverages AI technology to create bespoke marketing, design, and automation solutions for their customers. You can learn more about Nort’s services by visitng their website.

Conversational Pathways

Gain greater control over your AI agent and the conversational flow. Create a Pathway Now!

Template Pathway Video Tutorial

​

Introduction

Conversational pathways are our new way of prompting Bland that has led to major breakthroughs in realism.

Greater dialogue control

Give your agent instructions on how it should respond at specific points of the conversation. Choose between prompting or fixed sentences.

Webhooks

Execute webhooks at any point during the conversation, and send speech during/after the webhook.

End call

Control when your agent ends the call

Knowledge Base

Connect your agent to a knowledge base, to answer any questions the user has.

​

Terminologies

To understand how pathways work, let’s first understand the terminologies.

​

Nodes

These blocks you see here are called Nodes.

​

Pathways

Each of these dotted lines is called a Pathway. Their start end end points are the Purple Circles on the top and bottom of the nodes.

In order to create a pathway from a node, you would click on the purple circle at the bottom of the node and drag your mouse to connect to the top purple circle another node.

Upon doing so, you will now have a new dotted line connecting the two nodes, with a ‘New Pathway’ button in the middle of the line.

In order to instruct the agent when to take this pathway, you would click on the Edit icon on the ‘New Pathway’ button, and input the conditions for when the agent should take this pathway. In the above example, the agent would take this pathway if the user is not available to talk, so I labelled the pathway as ‘not a good time to talk’.

And there you have it! You have now created a pathway from one node to another, and instructed the agent when to take this pathway. You can connect as many nodes as you want in this manner, and create as many pathways as you want. In order to create a new Node, press the ‘Add new Node’ button at the top-left of the screen.

​

How the Pathways Agent Works

The agent starts at the first node, and then moves to the next node based on the pathway that the agent decides to take. The agent will then execute the instructions in the node as dialogue, and then move on to the next node based on the pathway that the agent decides to take. This process will continue until the agent decides to end the call.

The agent will make decisions based on the labels you put in the pathways, when connecting one node to another, and the dialogue generated will be based on the instructions you set in the nodes.

For Example,

In this example, at the node named ‘Ask for reservation info’, the node asks for the user’s reservation information. Based on the user’s response, the agent will then move on to the next node based on the labels you put in the pathways. For the current node, it will check if the user has provided reservation information where the number of guests is either less than 8 or more than 8. If the user has provided reservation information where the number of guests is less than 8, the agent will move on to the node named ‘Reservation booking’. If the user has provided reservation information where the number of guests is more than 8, the agent will move on to the node named ‘Transfer Call’. The agent will then execute the instructions in the node as dialogue, and then move on to the next node based on the pathway that the agent decides to take. And the process repeats!

​

Conditions

Conditions are a way to provide the agent with a condition that must be met in order for the agent to move on to the next node. If the condition is not met, the agent will stay on the same node and ensure the condition is met until the condition is fulfilled.

Using the same example above, I set the condition for the ‘Ask for reservation info’ node as follows - “You must get the date, time, and number of guests for this reservation”. This means that the agent will stay on the ‘Ask for reservation info’ node until the user provides the date, time, and number of guests for the reservation. If the user says something else or deviates from the conversation, the agent will stay on the ‘Ask for reservation info’ node and prompt the user to provide the date, time, and number of guests for the reservation.

This helps you to ensure that the user provides the necessary information before moving on to the next node, and helps you to control the flow of the conversation.

​

Global Nodes

Global Nodes take precendence over the condition decisions made by the agent. You can treat a global node as a node, that every other node in the pathway has a pathway to, with the label as the ‘Global Pathway Label’.

Using the Reservation Booking Example, if the user were to ask a question like ‘What are the opening hours of the restaurant’ when the agent is at the ‘Ask for reservation info’ node, the condition decision would not be met as the user did not provide the date, time, and number of guests for the reservation. However, the pathway label would be ‘user has a question about the restaurant’s hours or location’, which links to a Global Node. As Global Nodes take precedence over the condition decision, the agent would then move to the ‘Global Node’ named ‘Restaurant Questions’ and provide the user with the opening hours of the restaurant. After providing the user with the opening hours of the restaurant, the agent would then automatically return to the ‘Ask for reservation info’ node, and continue with the flow of the conversation.

This helps you to handle edge cases where the user might ask a question that is not related to the current conversation, and allows you to provide the user with the information they need, before returning to the conversation.

Tip: The variables {{lastUserMessage}} and {{prevNodePrompt}} can be used in the Global Node to provide the agent with context on what the user said, and steering the conversation back to its own original goal.

Example Global Node Prompt:

​

Node Types

There are currently 6 different types of Nodes

Default

Webhook

Knowledge Base

End Call

Transfer Call

Wait for Response

You can select the type of node you want by clicking on the dropdown of Node Type.

​

Base/Default Node (Important!)

The default node provides the ability to generate a response to the user. This functionality is exposed to all other nodes as well.

You can either:

Use the Prompt field to give instructions on what the agent should do at this point in the conversation. This is the recommended way to generate responses as it makes the conversation more human and natural.

Enable the ‘Static Text’ toggle to provide a fixed response, and the agent will always say the same thing at this point in the conversation.

​

Optional Decision Guide

The optional decision guide is only to be used if your phone agent currently is not going down the correct pathway. We do not expect this to happen often, but still want to provide you the tools to handle these issues if they arise.

It is a way to provide the phone agent with example scenarios of what the user might say, and what pathway the agent should take in response.

You would put in examples of what the user might say in the User Input field, and then in the Pathway field, you would provide the pathway the agent should take in response.

​

Condition

The condition is a way to provide the agent with a condition that must be met in order for the agent to move on to the next node. If the condition is not met, the agent will stay on the same node and ensure the condition is met until the condition is fulfilled.

​

Global Nodes

Each node can be configured to be a Global Node. Global Nodes are nodes that are accessible by every other node in the Conversational Pathway. This means that it has an implicit pathway to every other node, and the label would be the ‘Global Label’.

After entering a Global Node, the agent will execute the instructions inside the Global Node, and then automatically return to the node it was at before entering the Global Node, so it can continue with the flow of the conversation.

In a Global Node, you can also forward the agent to another node, by toggling the ‘Enable Forwarding’, which will allow you to select the node you want to move the agent to. This is useful if you want to move the agent to a existing node for a certain scenario, which could happen at any point in the conversation.

​

Transfer Call Node

The transfer call node is used to transfer the call to another number when the node is reached, and the dialogue at this node is complete.

As such, you may have the agent say any final words before the call is transferred.

​

End Call Node

The end call node will end the call when the node is reached, and the dialogue at this node is complete.

As such, you may have the agent say any final words before the call is ended.

​

Knowledge Base Node

The knowledge base node is used to connect your agent to a knowledge base, to answer any questions the user has.

Paste in any text in the ‘Knowledge Base’ field, and the agent will search through the knowledge base to answer the user.

Coming soon - PDF Upload/Vector Database Integrations…

​

Wait for Response Node

The Wait for Response Node works the same way as the Default Node, except it is also equipped with the ability to wait if the user requires time to respond or needs to hold for a moment.

​

Webhook Node

The webhook node is used to execute webhooks at any point during the conversation, and send speech during/after the webhook.

Webhook Information is all you need in order to execute a webhook, and works the same way as Dynamic Data. Refer to the Dynamic Data section for more information…

Similar to how the dialogue is handled in all other nodes, you can control the dialogue sent before, and after the webhook is executed.

Variables received from the webhook can be used in the dialogue as well, as shown in the example below.

​

Global Prompt for all Nodes

The ‘Add Global Prompt to All Nodes’ feature is to assist in providing context/instructions to the agent for all nodes, without having to manually input the same prompt for each node. One Example of a Global Prompt could be to provide the agent with instructions on how to handle the call, the tone of voice to use, or answering any questions the user might have.

​

Live Call Logs

The live call logs are a way to provide you with a live feed of the conversation between the agent and the user. This is useful for debugging purposes, and to see how the agent is responding to the user in real-time and the decisions the agent is making. On top of the transcript, we expose the updated node the agent took, the pathway that the agent took, as well as whether the condition was met or not.

​

Testing the Pathway Agent via Chat

You can test the responses from your pathway agent by clicking on the ‘Chat with Pathway’ button at the top right of the screen. This will open a chat window where you can test the agent by sending messages to the agent, and see how the agent responds. The live call logs will also be displayed on the right side of the screen, so you can see the decisions the agent is making in real-time.

​

Variables Reference/ Extraction

​

Variables

You can reference variables in pathways using double curly braces like {{first\_name}}. You can pass variables into your call by passing in the key-value pairs in the request\_data field when sending a call.

Some examples of variables that you can access at each node:

{{lastUserMessage}} - The last response from the user

{{prevNodePrompt}} - The prompt from the previous node

{{now\_utc}} - The current time in UTC

{{from}} - The phone number the call is from

{{to}} - The phone number the call is to

{{call\_id}} - The unique identifier for the call

​

Extracting Variables from Call/User Response

At each node, you can also extract variables from the user’s response using the Extract Variables from Call Info field. You would put in the name of the variable you want to extract, the type of the variable (integer, string, boolean), and the description for what information you want the variable to store. You can also provide specific formats you expect and examples in the description. The more descriptive it is, the better the agent will be at extracting the variable more accurately.

Do note that when enable variable extraction, and wanting to reference the variable in the subsequent nodes, it would introduce slight latency as the agent would have to extract the variable from the user’s response before generating the dialogue for the next node.

Within the webhook node, the variable extraction happens before the webhook is executed, so that you can reference the variables extracted from the user’s response in the webhook’s request data. The variables extracted from the webhook in response\_data can also be referenced in the dialogue generated after the webhook is executed, in the same manner.

For Example, the image below shows how the Webhook Node is set up to extract the date, time, and number of guests for the reservation, and how the extracted variables are referenced in the webhook’s request data.

​

Fine Tuning the Pathway agent

While Conversational Pathways gives you a lot greater control compared to the regular call agent, hallucinations can still occur, or the agent might make wrong decisions in the pathway. However, we have provided you with the tools to handle these issues if they arise. Each node can be fine-tuned on the decision it makes, as well as the expected dialogue it generates, allowing you to handle even the most extreme edge cases that might arise easily.

​

Steps to Fine-Tuning the agent

Upon triggering and testing a call with your pathway using the ‘Chat with Pathway’ button, or sending a call, you will be able to see the live call logs.

If you see a decision that the agent made that you do not agree with, or if the agent is hallucinating, you can fine-tune the decision the agent makes by clicking on the Edit button on the PATHWAY DECISION INFO block.

Upon doing so, you will be able to see the decisions the agent made for the condition, the pathway it took, and the dialogue it generated. You can then fine-tune the agent by changing the dialogue, the condition, or the pathway the agent took.

Upon saving your changes, you will be able to see the fine-tuning data in the node where the decision was made. This training data is used to train the agent to make better decisions in the future, and to ensure that the agent does not make the same mistake again.

Do note that the decisions for the condition and pathway chosen is made by the node the agent is currently at, and the dialogue is generated by the node which the agent decides to take the pathway to.

For Example, If the agent is at Node 1, and the agent decides that the condition is achieved, and decides to take the pathway to Node 2, the dialogue generated will be from Node 2, and the decisions for the condition and pathway chosen will be from Node 1. As such, the training data for the condition and pathway chosen will be stored in Node 1, and amendments to the dialogue generated will be stored in Node 2.

​

Quick Start with Tutorial / Templates

To immediately start playing around with Conversational Pathways, you can use one of our templates. Visit the Conversational Pathways page on our Dev Portal, and duplicate the ‘Restaurant Reservation’ Template, and run through the agent to see how it works! We have also created a video walkthrough of that template to help you get started!

Video Walkthrough/Tutorial

Create a pathway now! Click here to get started!

If you have any additional questions, reach out on our Discord and one of our engineers will help.

Basic Tutorials

Custom Tools

Interact with the real world mid-call using custom tools.

​

Introduction

Custom tools allow your agent to interact with any web API mid-call. Do things like:

Send a message

Dispatch SMS or emails using the person’s contact info.

Schedule an appointment

Set appointments using live calendar availability.

Create a support ticket

Generate support tickets in your issue tracker.

Update your CRM

Update your CRM with relevant details during the call.

​

Background

To understand how custom tools work, let’s take a peek under the hood of the Bland AI phone agent.

During the conversation, the phone agent is constantly listening to figure out when it’s supposed to respond. When the phone agent realizes it’s time to respond, it reviews the tools in its toolbox and picks between them.

Those tools include a speak, wait, and button press tool. When you create a custom tool, you add it to the existing ‘toolbox’ for the phone agent to pick from.

A few natural questions arise:

How do I define my custom tool?

How do I make sure my tool gets picked at the right time?

How does information from the call get passed to my custom tool’s API request?

How do I fill the silence (when my custom tool is running)?

How does the response from my custom tool get added to the call?

Keep reading to find out.

​

Creating your custom tool

​

Custom Tool Example

{

"name": "BookAppointment",

"description": "Books an appointment for the customer",

"url": "https://your-api.com/book-appointment",

"method": "POST",

"headers": {

"Authorization": "Bearer YOUR\_API\_KEY"

},

"body": {

"date": "{{input.date}}",

"time": "{{input.time}}",

"service": "{{input.service}}"

},

"input\_schema": {

"example": {

"speech": "Got it - one second while I book your appointment for tomorrow at 10 AM.",

"date": "2024-04-20",

"time": "10:00 AM",

"service": "Haircut"

},

"type": "object",

"properties": {

"speech": "string",

"date": "YYYY-MM-DD",

"time": "HH:MM AM/PM",

"service": "Haircut, Coloring, Trim, or Other"

}

},

"response": {

"succesfully\_booked\_slot": "$.success",

"stylist\_name": "$.stylist\_name"

}

}

​

From API request to custom tool

The next step is to convert the API request into a custom tool. Custom tools have the following properties:

name - the agent will see this in the list of tools

description - a short explanation of what the tool does

input\_schema - a JSON schema describing the input data

speech (optional) - a string that will be spoken to the agent while your tool waits for a response

response\_data - An array of objects that describe how to extract data from the response. Within the response data, you can create variables that the phone agent can reference in its prompt.

​

Name & Description

The agent will see the name in the list of tools. The name, plus the description, help the AI phone agent when it decides which tool to use.

For this example we’ll set the name to BookAppointment and the description to Books an appointment for the customer.

​

Input Schema

The input schema is critical. It defines the shape of the API request, the different inputs the request can take, and also includes an example (which helps our system when creating requests).

Here’s what the input schema could look like:

"input\_schema": {

"example": { // "example" is a special property that shows an example of what the input object the agent creates should look like

"speech": "Got it - one second while I book your appointment for tomorrow at 10 AM.",

"date": "2024-04-20",

"time": "10:00 AM",

"service": "Haircut"

},

"type": "object",

"properties": {

"speech": "string",

"date": "YYYY-MM-DD",

"time": "HH:MM AM/PM",

"service": "Haircut, Coloring, Trim, or Other"

}

}

Two important notes about input schema:

input\_schema is converted into the variable "{{input}}" that you can use in the request body/query/headers

To access nested properties, use dot notation: "{{input.property.subproperty}}"

For example, later on you could use "{{input.service}}" to have whatever type of appointment that the customer wants

What you’re doing here is describing the structure of the variables that the agent will create.

Special Note: If you need to gather detailed letter-by-letter information from the user, raise your interruption\_threshold parameter to about 200 so that the AI doesn’t interject so quickly.

Scroll down to see the full example.

​

Speech

Because requesting external APIs might take a while, we enable you to define a speech property. The phone agent will say the speech while it makes the request.

An example speech might look like: Perfect, I'll schedule that right now, give me just a second.

For the restaurant ordering example, the speech could be Thank you, placing that order now.

​

Response data

Once your API request comes back, you need to extract the response data, and then make the phone agent aware of the new information.

The data field determines how you extract the data while the name field determines the variable name for reference in the prompt.

Here’s an example response data:

"response": {

"succesfully\_booked\_slot": "$.success",

"stylist\_name": "$.stylist\_name", // if your API returns a JSON object with a key "stylist\_name"

}

​

Full example

Below is the entire API request for sending a phone call using the outlined custom tool:

{

"phone\_number": "+12223334444",

"prompt": "...",

"tools": [

{

"name": "BookAppointment",

"description": "Books an appointment for the customer",

"url": "https://your-api.com/book-appointment",

"method": "POST",

"headers": {

"Authorization": "Bearer YOUR\_API\_KEY"

},

"body": {

"date": "{{input.date}}",

"time": "{{input.time}}",

"service": "{{input.service}}"

},

"input\_schema": {

"example": {

"speech": "Got it - one second while I book your appointment for tomorrow at 10 AM.",

"date": "2024-04-20",

"time": "10:00 AM",

"service": "Haircut"

},

"type": "object",

"properties": {

"speech": "string",

"date": "YYYY-MM-DD",

"time": "HH:MM AM/PM",

"service": "Haircut, Coloring, Trim, or Other"

}

},

"response": {

"succesfully\_booked\_slot": "$.success",

"stylist\_name": "$.stylist\_name"

}

}

]

}

​

Frequently asked questions

How does the phone agent determine what it sends an API request?

The phone agent refers to the input schema and the example within it. Both of those pieces of information provide context about what data to pass.

Then the phone agent extracts the information from the transcript and passes it to the request body.

You can improve the accuracy of the input data by creating a very clear input\_schema. That includes providing a detailed example within.

How does the phone agent decide when to use the custom tool?

The phone agent looks at the tool’s name and description. Then it looks at the current context of the conversation to decide whether using the tool makes sense.

How do I access the info the custom tool responds with?

When the API response from the custom tool comes back, you can extract the API response and create variables. You can do this within the response\_data property.

Once you’ve given the variable a name, you can reference it in the prompt using double brackets ({{}}).

Note, with the current setup, you might reference variables that have null values. Here’s the restaurant example prompt:

You are taking a drive thru order from a customer. Find out everything that they want like a drive thru cashier. Continue until they say they’re done. Repeat the full order back to them after that, and ask if that’s correct. If they confirm that it’s correct, then and only then will you place their order using the PlaceOrder tool. After you place it, tell them their order total and to pull forward. Their order price is {{order\_price}}

In the above prompt, the order\_price will be null until the data comes back. That’s okay though. The prompt is structured to first take the order, then use the PlaceOrder tool, and then finally respond with the order price.

By the time the phone agent is asked for the order price, it will have the information.

Custom tools will continue getting more robust, to further prevent scenarios where variables without value from being referenced in prompts.

If you have any additional questions, reach out at hello@bland.ai and one of our engineers will help.

Conversational Pathways

Basic Tutorials

Send your first phone call

​

Step 1: Setup Your Authorization

Before making a call, you need to authenticate your request. Make sure you have your API key ready.

Sign up on the developer portal to get yours.

​

Step 2: Prepare the Call Data

You will need to provide specific details for the call. These include:

phone\_number: The number you want to call. Remember to include the country code.

task: Describe the purpose of the call and how the AI should handle the conversation.

voice\_id: Choose the voice persona (American male, Australian female, etc.) based on your preference.

Set parameters like reduce\_latency, record, amd (for navigating phone trees), and wait\_for\_greeting according to your call’s requirements.

​

Step 3: Customize the Call

You can further personalize the call by:

Setting a first\_sentence for the call.

Specifying dynamic\_data to incorporate external API data.

Adjusting voice\_settings for stability, similarity, and speed.

Choosing the language with the language parameter.

Setting a max\_duration for the call.

​

Step 4: Send the API Request

Use the provided JavaScript or Python code snippet to make the API request.

​

Step 5: Handle the Response

After the call, you will receive a response with the status and call\_id. If you set record to true, you can retrieve the recording using the /call/recording endpoint.

Here’s what an example response might look like:

{

"status": "success",

"call\_id": "9d404c1b-6a23-4426-953a-a52c392ff8f1"

}

​

Step 6: Monitor the Call

If you have set up a webhook, you will receive real-time notifications and transcripts once the call completes.

And that’s it! You’re now ready to make your first AI-powered phone call with Bland AI. Happy calling!

// Headers

const headers = {

authorization: "YOUR-API-KEY-HERE",

};

// Data

const data = {

phone\_number: "+11233456789",

task: "You are calling Fantastic Airlines on behalf of John Doe. Find out where John's Bags are located.",

voice\_id: 0,

language: "eng",

request\_data: {

calling: "Fantastic Airlines",

bag\_claim: "69683",

airline\_code: "UA123",

},

record: true,

reduce\_latency: true,

amd: true,

};

// API request

await axios.post("https://api.bland.ai/v1/calls", data, { headers });

# Headers

headers = {

'authorization': 'YOUR-API-KEY-HERE'

}

# Data

data = {

'phone\_number': '+11233456789',

'task': "You are calling Fantastic Airlines on behalf of John Doe. Find the location of out where John's Bags are located.",

'voice\_id': 0,

'request\_data': {

'calling': 'Fantastic Airlines',

'bag\_claim': '69683',

'airline\_code': 'UA123'

},

'record': True,

'reduce\_latency': True,

'amd': True

}

# API request

response = requests.post('https://api.bland.ai/v1/calls', json=data, headers=headers)

Basic Tutorials

Webhook Signing

Bland webhooks are signed with a secret key to ensure that they are not tampered with in transit and to confirm that they were sent by Bland.

​

Signing Webhooks

When Bland sends a webhook, it calculates a signature using the HMAC algorithm with the SHA-256 hash function. The signature is then included in the X-Webhook-Signature header of the request.

To create a webhook signing secret, first go to the Account Settings in the Dev Portal and click on the “Keys” tab.

Here you can create a new secret by clicking “Replace Secret”. It will only be shown once, so save it securely.

​

Verifying Webhooks

To verify a webhook, you need to calculate the HMAC signature of the request body using the secret key and compare it to the signature in the X-Webhook-Signature header.

Note that you must first create a webhook signing secret in the Account Settings in the Dev Portal.

Here is an example of how to verify a webhook in Node.js:

const crypto = require('node:crypto');

function verifyWebhookSignature(key, data, signature) {

const expectedSignature = crypto.createHmac('sha256', key)

.update(data)

.digest('hex');

return expectedSignature === signature;

}

//...

app.post('/webhook', (req, res) => {

const isValid = verifyWebhookSignature(

process.env.WEBHOOK\_SECRET,

JSON.stringify(req.body),

req.headers['x-webhook-signature']

);

//...

});

Send your first phone call

Basic Tutorials

Send 1000 phone calls

​

Step 1: Setup Your Authorization

To initiate a batch call, you must first authenticate your request. Ensure you have your API key from signing up on the developer portal.

​

Step 2: Create the Base Prompt

Craft a base prompt that will be common across all calls in the batch. Use placeholders {{curly braces}} for dynamic content.

Example:

"You are calling {{business}} to renew their subscription to {{service}} before it expires on {{date}}.";

​

Step 3: Define the Call Data

Specify the list of calls in the call\_data array. Each call must have a phone\_number and can include other properties corresponding to placeholders in your base prompt.

Example:

[

{

phone\_number: "1234567890",

business: "ABC co.",

service: "Netflix",

date: "September 4th",

},

{

phone\_number: "32176540987",

business: "XYZ inc.",

service: "Window Cleaning",

date: "December 20th",

},

];

​

Step 4: Additional Configuration

label: Assign a label to your batch for easy tracking.

campaign\_id: Organize related batches under a campaign.

test\_mode: Set to true for testing with the first call only.

batch\_id: Manually set or auto-generated for tracking.

Voice and Language Settings: Select voice\_id, reduce\_latency, and language.

request\_data: Include specific facts for the AI to know during the call.

webook: For real-time notifications and transcripts post-call.

max\_duration: Define the maximum length of each call.

amd: Enable for navigating phone trees or leaving voicemails.

wait\_for\_greeting: Control if the AI speaks immediately or waits.

​

Step 5: Send the API Request

Use the provided JavaScript or Python code snippet to make the API request.

​

Step 6: Handle the Response

After sending the batch request, you’ll receive a response with a message and the batch\_id. Monitor the progress of your calls and any responses via your specified webhook.

Here’s what an example response might look like:

{

"message": "success",

"batch\_id": "3p$7rQ3p9sT5bzmF-gen-batch"

}

​

Step 7: Monitoring and Analytics

Track the performance and outcomes of your batch calls through the provided batch\_id and campaign analytics. Adjust future batches based on the insights gained.

// Headers

const headers = {

authorization: "YOUR-API-KEY-HERE",

};

// Data

const data = {

base\_prompt:

"You are calling {{business}} to renew their subscription to {{service}} before it expires on {{date}}.",

call\_data: [

{

phone\_number: "1234567890",

business: "ABC co.",

service: "Netflix",

date: "September 4th",

},

{

phone\_number: "32176540987",

business: "XYZ inc.",

service: "Window Cleaning",

date: "December 20th",

},

],

label: "Renewal Reminder - Wednesday Afternoon with female voice",

voice\_id: 0,

max\_duration: 10,

reduce\_latency: true,

wait\_for\_greeting: true,

};

// API request

await axios.post("https://api.bland.ai/v1/batches", data, { headers });

Basic Tutorials

Dynamic Data

Interact with the real world by connecting your agent to external APIs.

​

Introduction

With Dynamic Data, you can make external API requests at the start and throughout your phone call. This allows you to load data from your database, or from any other API. You can then use that data in your AI responses, or to define circumstantial behavior for each call.

Some examples of what Dynamic Data enables:

Maintain conversation history between calls

Define behavior based on the user’s location

Handle real-time data like status updates or prices

Expand to see an example in action

We’ll cover the following features in this section:

System variables

External API requests

Extracting data from responses

Variables as parameters

Chaining requests

​

System Variables

Variables are defined with double curly braces, like {{variable}}. System variables are predefined variables that are available in every AI phone call. You can use them to access information about the current call, like the user’s phone number or the current time.

Note: Variables are NOT case sensitive, and outer spaces are trimmed automatically.

Base variables:

{{phone\_number}} - Always the other party’s number

{{country}} - The country code (ex. US)

{{state}} - The state or province’s abbreviation (ex. CA for California)

{{city}} - The full city name, capitalized

{{zip}} - The zip code

{{call\_id}} - The unique ID of the current call

{{now}}

{{now\_utc}}

{{from}} - The outbound number in E.164 format

{{to}} - The inbound number in E.164 format

{{short\_from}} - outbound number with country code removed

{{short\_to}} - inbound number with country code removed

Variables can be used mid-sentence, like this:

{

"task": "... Today is {{now}} ..."

}

​

External API Requests

External API requests are defined in the dynamic\_data parameter of your call request or inbound agent configuration. The dynamic\_data parameter is an array of objects, where each object represents an API request.

Here’s a simple request that can be used to load public data about the current price of Bitcoin, then store it in a variable called {{bitcoin\_price}}:

{

"dynamic\_data": [

{

"url": "https://api.coindesk.com/v1/bpi/currentprice.json"

"response\_data": [

{

"name": "bitcoin\_price",

"data": "$.bpi.USD.rate",

"context": "Current price of Bitcoin in USD is ${{bitcoin\_price}}"

}

]

}

]

}

Additional Parameters

​

Extracting Data from Responses

Rather than using the full response, you can extract specific data from the response using the data parameter. The data parameter follows JSON structuring, using dot notation and array indices. For example, if the response is:

{

"bpi": {

"USD": {

"code": "USD",

"rate": "9,000.00",

"description": "United States Dollar",

"rate\_float": 9000

},

"GBP": {

"code": "GBP",

"rate": "6,000.00",

"description": "British Pound Sterling",

"rate\_float": 6000

}

}

}

Then $.bpi.USD.rate would return 9,000.00.

More complex filters can be used if they follow the JSONPath format.

​

Variables as Parameters

Once defined with response\_data, variables can be used nearly anywhere.

In the task or prompt parameters

In the context parameter of response\_data

In the body, headers and/or query parameter of each request

Afterwards, in your webhooks and when retrieving call transcripts, the variables field will contain all variables that were defined during the call.

By far, the easiest way to test out your dynamic\_data configuration is via the /dynamic\_data/test endpoint. It returns the original configuration, every raw response, and the final variables after parsing is applied.

​

Chaining Requests

Each request is executed in order, and variables defined in previous requests can be used in the next request. For example, if you want to retrieve information from your database or ours, then take additional actions with that data then you could do something like the following:

For this example, imagine a delivery service that offers instant checkout for customers that have signed up to be a member. The first request retrieves their member\_id from your database like so:

{

"dynamic\_data": [

{

"url": "https://api.restaurant.com/customers",

"method": "GET",

"headers": {

"authorization": "ExtremelySecureCredentials"

},

"query": {

"phone\_number": "{{phone\_number}}"

},

"response\_data": [

{

"name": "member\_id",

"data": "$.customer.member\_id"

}

]

},

//...

]

}

We just created that {{member\_id}} variable - now we can use it in the next request.

This delivery service also can be called to check on an order status.

Note a difference: The cache parameter is set to false, so if the order status changes during the call, the agent will immediately know about it and be able to inform the customer.

{

"dynamic\_data": [

//...

{

"url": "https://api.restaurant.com/orders",

"method": "GET",

"cache": false,

"headers": {

"authorization": "ExtremelySecureCredentials"

},

"query": {

"member\_id": "{{member\_id}}"

},

"response\_data": [

{

"name": "order\_id",

"data": "$.orders[0].id"

},

{

"name": "order\_status",

"data": "$.orders[0].status"

}

]

}

]

}

Basic Tutorials

Setting max duration

​

Introduction

Controlling the length of your AI-powered phone calls is crucial for efficiency and cost-effectiveness. In this guide, we’ll walk you through how to set a max\_duration for your calls using Bland AI.

​

Step 1: Understand max\_duration

The max\_duration parameter allows you to specify the longest duration you want your call to last, measured in minutes. Once the set duration is reached, the call will automatically end.

​

Step 2: Setup Your Authorization

Ensure you have your API key ready for authentication. If you haven’t obtained one, sign up on the developer portal.

​

Step 3: Define max\_duration in Your Call Data

When preparing your call data, include the max\_duration parameter. You can set it as a float or a string.

Example values: "30", "5.5", 45, 2.8.

Here’s how you might include it in your request data:

{

"phone\_number": "+11233456789",

"task": "Inquire about the latest product updates.",

"max\_duration": "10", // Call lasts for 10 minutes max

}

​

Step 4: Send the API Request

Use the provided JavaScript or Python code snippet to make the API request, ensuring you include the max\_duration parameter.

​

Step 5: Test and Adjust

After setting max\_duration, test your calls to ensure they’re ending at the desired time. Adjust as necessary based on your needs and feedback.

​

Conclusion

Setting a max\_duration for your calls ensures they are concise and to the point, saving time and resources. You’re now ready to efficiently manage the length of your AI-powered calls with Bland AI.

Remember, the key is to find the right balance between giving enough time for meaningful interaction and keeping the calls concise. Happy calling!

# Headers

headers = {

'authorization': 'YOUR-API-KEY-HERE'

}

# Data

data = {

'phone\_number': '+11233456789',

'task': 'Inquire about the latest product updates.',

'max\_duration': '10', # Call lasts for 10 minutes max

}

# API request

response = requests.post('https://api.bland.ai/v1/calls', json=data, headers=headers)

// Headers

const headers = {

authorization: "YOUR-API-KEY-HERE",

};

// Data

const data = {

phone\_number: "+11233456789",

task: "Inquire about the latest product updates.",

max\_duration: "10", // Call lasts for 10 minutes max

};

// API request

await axios.post("https://api.bland.ai/v1/calls", data, { headers });

Basic Tutorials

Live transfer

​

Introduction

Implementing live transfer in your AI-powered phone calls enhances flexibility and customer experience. This guide will explain how to set up a live transfer during a call using Bland AI.

​

Step 1: Understand Live Transfer

Live transfer allows the AI agent to transfer the call to a human representative under certain conditions. This is crucial for scenarios where human intervention is preferred.

​

Step 2: Setup Your Authorization

Before initiating a live transfer, ensure your API key is ready. Obtain your key from the developer portal if you haven’t already.

​

Step 3: Configure the Transfer Settings

Include the transfer\_phone\_number parameter in your call data. This is the number the AI will transfer to. Additionally, define the conditions for transfer in the task parameter.

Example:

{

"phone\_number": "+11233456789",

"transfer\_phone\_number": "+19876543210",

"task": "If the caller requests to speak with a human, transfer the call to the representative."

}

​

Step 4: Send the API Request

Make the API request using the JavaScript or Python code snippet provided, ensuring the transfer\_phone\_number and conditions are included.

​

Step 5: Test and Monitor

After setting up the live transfer, test the functionality to ensure it works as expected. Monitor the calls to adjust the transfer conditions as necessary.

​

Conclusion

Setting up a live transfer offers a seamless experience for situations where AI needs to hand over to a human. You’re now ready to integrate this feature into your AI-powered calls with Bland AI.

Maintain a balance between automated and human interactions to optimize customer satisfaction. Happy calling!

Basic Tutorials

Webhooks

​

Introduction

This is a quick tutorial to help you set up and test your Bland AI webhook. We’ll include instructions both for inbound and outbound phone calls.

We’ll start with inbound because it’s more popular.

​

Step 1: Create your webhook

To create a test webhook visit Webhook.site

The website will automatically provide you a unique webhook URL.

​

Step 2: Connect to your inbound phone number

Open your developer portal and visit the inbound phone numbers page.

Paste your webhook into the webhook field. Make sure to remove the initial https:// when you insert the URL. Then click test webhook.

​

Step 3: Verifying your outputs

Navigate to webhook.site page, and check if the test webhook fired correctly. You’ll know it worked because a new record will populate.

At this point, if your record fails to populate, double check that you provided the correct URL - and that you REMOVED the initial https:// from it.

Otherwise, if issues persist, jump into the discord - one of our teammates will help you asap.

​

Step 4: test a live phone call

Call your inbound phone number. Once it ends, visit the Webhook site and confirm once again that a new record populated.

If that’s working, then you’re set!

​

Step 5: Testing for outbound calls

To test for outbound calls, once again create your webhook by referring back to step 1.

Then, follow the send phone call docs to create and send a phone call. Make sure you include the webhook as a parameter in your request. After, confirm that the webhook data populated on your webhook site page.

And again, if you encounter issues, jump into discord and message us - we will help asap.