# Table of Contents

# Welcome to Rebyte

# Welcome to Rebyte

## What is ReByte?

ReByte is a platform that allows everyone to develop their AI-powered applications without needing too much coding skill. On ReByte, anyone with or without programming experiences, can create various types of applications and publish them on our platform.

## What can I do with ReByte?

Firstly, LLMs provide a vehicle for **every software** to use LLM as human intention understanding. Secondly, LLMs provide a way for **end users** to write small pieces of code without the need to learn traditional programming. Soon, anyone with a computer will be able to develop their own software tools by simply describing the desired features and the changes they want to implement in the software they are already using.

This will open the hood of software development to a much larger group of people. Everyone can use their personal computers to create their **personal** software tools. Every company can use their private data to create their **personal** software tools for internal or external uses.

**ReByte** is built with one mission: to give everyone the ability to create their own software tools without needing too much coding skills.

In terms of functionality, **Rebyte** has some core components:

- Language Model Agent: A serverless function that can be executed on cloud. Those functions usually use LLMs to perform some tasks, but it's not required.
- Knowledge: Private data ingestion pipeline that feed data to Rebyte system, later can be used by LLM agents.
- App Builder: User interface builder that allows developers to wire up LLM agents and knowledge to create their own tools.

Also, **ReByte** is built to solve the following problems in LLM application development:

1. **Predicable LLM application**: It works on the principle of treating each LLM interaction as a single step or functional transformation on working memory, offering a predictable and manageable way to guide the thought process of a LLM. This approach results in consistent, easier-to-follow interaction flows.
2. **Iterative Nature of LLM Development**: Creating tools on top of modern LLM models can be quite challenging, due to the nature of LLMs. LLMs' output are not deterministic, and their performance can vary from time to time. We want ReByte's LLM agent builder to make this process as easy as possible.
    - We provide a unified interface for users to build, test and deploy their LLM application.
    - We also keep logs of the application, so that users can easily fix the bugs when things go wrong.
3. **Painless Private Data integration**: Today's LLM tools heavily rely on private data. Frameworks such as Langchain and LlamaIndex, together with various vector databases, provide the necessary pieces for private data embedding and search. But even for someone experiences in coding and engineering, it's hard to get the system running the first time, and usually takes hours to debug. That's why we want to make this data integration process as easy and painless as possible.
    - We provide a hosted solution for users to import data from popular data sources, such as Notion, Google, GitHub, etc., as well as from user's local computers.
    - We handle data source synchronization automatically, so users don't need to worry about data consistency.

4. **Production Serving Ready**: Provide a robust runtime ready for immediate deployment and scalable growth.
    - It's rather hard to scale LLM applications, because of its unpredictable and asynchronous nature.
    - Being inspired by serverless functions, we provide a scalable runtime for users to scale their LLM applications without having to know the details of the underlying infrastructure.
5. **Deliver Tool instead of Prompt**: LLM applications are meant to be ready-to-use tools, not just prompts. We make sure that users can deliver their LLM applications as actual tools, instead of prompts.

At its core, ReByte defines an **Agent DSL** as an intermediate representation of any LLM application, and provides a hosted runtime for users to execute their LLM applications. Introduction to Agent DSL has obvious benefits:

- can be used to build a GUI builder for users to create their own LLM application.
- can be generated by fine-tuned models
- can be edited by human

We believe that building good agents must take efforts from both human and machines, and DSL is the perfect medium for this collaboration.

# Understanding Rebyte Architecture

# understanding-rebyte-architecture

## Agent

Subroutine for your AI application.

There are two types of agents in ReByte:

**Stack Agent**

- Stack Agent is a piece of sequential actions that can be executed on the LLM serverless runtime. It is the core building block of ReByte, and the main way for end users to create their own tools. Rebyte provides a GUI builder for end users to create/edit their own LLM agents. Rebyte provides a list of pre-built actions for common use cases, also private SDK for *software engineer* to build their own actions, and seamlessly integrate with the agent builder. Pre-built actions includes:
    - LLM Actions
        - Language Model Completion Interface
        - Language Model Chat Interface
    - Data Actions
        - Dataset Loader, load pre defined datasets for later processing
        - File Loader, extract/transform/load user's provided files
        - Semantic Search, search for similar content over user's knowledge base
    - Tools Actions
        - Search Engine, search for information on Google/Bing
        - Web Crawler, crawl web pages and extract information
        - Http Request Maker, make any http request to any public/private API
    - Control flow Actions
        - Loop Until, run actions until a condition is met
        - Parallel, execute multiple actions in parallel
        - Vanilla Javascript, execute any vanilla javascript code, useful for doing pure data transformation
    - Code Interpreter Actions
        - Without relying on OpenAI, rebyte provides a code interpreter that can execute javascript code.

**Group Agent**

- Group Agent is a group of Stack Agents, the biggest difference between Group Agent and Stack Agent is that state transition between Stack Agents are solely controlled by LLM, yielding a non-deterministic behavior. AutoGpt is a great example of Group Agent. In Rebyte we build in a Group Agent builder that allows users to create their own Group Agents.

## User Facing App Builder - build your own tools

End user facing UI for your AI application.

- We believe chat UI might be the best UI for end user to interact with LLM, but definitely not the only UI. Rebyte provides builders for end users to build their own UI for their LLM agents. We call this customized UI. Customized UI can be

any valid UI code generated by LLM, rebyte hosts the UI code and provide a URL for end users to access the UI.
- Rebyte customized UI builder can be magically integrated with LLM agents, so that end users can easily build their own tools without the need to write a single line of code.

## Knowledge - capture private data

Ingredient for your AI application.

- Knowledge is private data that is stored in rebyte managed vector database. Rebyte currently provides following connectors for end users to import their knowledge:
    - Local file, supported file types are:
        - "doc", "docx", "img", "epub", "jpeg", "jpg", "png", "xls", "xlsx", "ppt", "pptx", "md", "txt", "rtf", "rst", "pdf", "json", "html"
    - Notion
    - Discord
    - GitHub
    - More connectors are coming soon
- Knowledge can be used in LLM Agents to do semantic search, or to do data augmentation. A great example is to use knowledge to do semantic search on a user's private knowledge base, and use the search result to do data augmentation for a language model, aka **Retrieval Augmented Generation**.

## Agent DSL

Each Rebyte agent can be described as a sequential list of actions, and each action can be described as a JSON object. We call this JSON object **Agent DSL**. Agent DSL is the intermediate representation of any LLM application, and can be used to build a GUI builder for end users to create their own LLM application. Agent DSL can be generated by fine-tuned model, or can be edited by human. We believe that agent building process must be a collaborative process between human and machine, and DSL is the perfect medium for this collaboration.

## ReByte Runtime

Interpreter for agent DSL.

All Rebyte Agents are executed on ReByte Runtime, which is a serverless function runtime that serves similar functionality as AWS Lambda.

# Quick Start

Quick Start

We will show you how to build a grammar agent in 5 minutes.

## Step 1: Create an Agent

- Navigate to the "My Agents" tab in the sidebar and then click on "Create Agent".
- Describe what you want to do with this agent and choose the right template for your agent.

- Click the "Generate Agent Template" button and we will generate a basic template for you to build from.
- You can change the name, description and visibility of the agent and add tags for more information. Or you can simply regenerate the template if you don't like the current one.

- Click the "Create Agent" and you will have your own agent in just a few seconds.

## Step 2: Design Your Agent

- In the auto-generated template, we have already created some actions for you.
- To build a simple agent like our grammar checker, there's no need to add more actions. For more complex agents, you can add actions by clicking the plus button between different actions.
- Write the instructions more the model in the editor, describing what you want the model to do.

## Step 3: Test Your Agent

- Click the "Datasets" tab on the top, then click "Create Datasets".
- Fill in the name and description of your dataset.
- Since this is a chatbot, the testing dataset will be in the form of a (list of) json object representing a conversation.

- After you've created the dataset, go back to the "Design" panel and choose the new dataset as the input.

  /figure> * Click "Run Testcases" to test your agent with the dataset. * The results will be shown below each action. See if the output is what you want. If not, change the settings of your agent and try again.

  ## Step 4: Deploy the Agent

- Click "Deploy Agent" on the top right, then click "Deploy A New Version".
- You can use your agent on your ReByte app or integrate it into your own app using the code we provide.

🎉 **Congratulations, you have created your first agent!**

View all your agents in the "My Agents" tab. You can also clone, save, or delete your agents here.

# Design Your Agent

Design Your Agent

# Actions

## Actions

# Input

# Input

- The `Input Action` is used to send input information to the Agent. Other actions are based on the input information to perform the right operation.
- The first action in the Agent must be an `Input Action`, you can not delete it or copy it.
- You can select different input datasets to run your agent. For more information about datasets, please refer to Datasets.

## Usage

You can use the input by using the `{{INPUT.message}}` variable in the instructions or `env.state.INPUT.messages` in the code editor.

## Data Format

- When used in agent page, the input is extracted from the predefined datasets.
- When connected to Apps, the input is from the app user's input and conversation history. By default, We will put the last 10 messages to the agent.
- The input data format is as follows:

```
{
"messages":[{
        "role": "user",
        "content": "The content."
    },
    {
        "role": "assistant",
        "content": "The content."
    }]
}
```

# Output

# Output

- The `Output` action is used to send output information to the Agent.
- The last action in the Agent must be an `Output` action, you can not delete it or copy it.
- The last action before the output action will give the output to the `Output` action. The `Output` action then give the output to agent's users or apps calling the agent.

## Usage

- You can insert a `Code` action before the `Output` action and specify the results you want to return to the user.
- With no special settings, the `Output` action will return the last action's output to the user.

## Data Format

- When used in agent page, the output is shown under the `Output` action.
- When connected to Apps, the output is sent to the App's user.
- The output data format is as follows:

  ```
  {
      "role": "assistant",
      "content": "The content."
  }
  ```

# Model

## Model Actions

# Language Model Chat

Language Model Chat

## Usage

We provide the `Language Model Chat` action to easily chat with the large language model and create complex uses.

To use this action you just need to write your specifications and configure the model, and the large language model will generate the response.

### Specification

#### Instruction

- This is the message that will be sent to the model.
- Write your prompt here and tell the large language model what to do.
- Supports Tera format. For example, you can use `` to get the content of the agent's input first message.

#### Messages

- This is the messages that will be sent to the LLM.
- You can refer to the output of other actions here using Javascript or Tera format.
- Messages can be a list of strings or list of objects. If it's a list of objects, make sure each object has a `role` field as well as a `content` field.:
  - The `role` field specifies the role of the message, allowed roles are `user`, `assistant` and `system`.
  - The `content` contains the message to be sent to the model.
- Uses **Javascript** format, for example, you can use `env.state.INPUT.messages` to get the agent's input.
- NOTE: Make sure to return an array.

#### Functions

- Function calling allows you to connect large language models to external tools by describing functions in an API call, enabling the model to intelligently generate JSON objects containing arguments for calling one or multiple functions.
- This capability provides a way to obtain structured data from the model, enabling tasks such as creating assistants that interact with external APIs and converting natural language into API calls.
- Uses **Javascript** format.
- NOTE: Make sure to return an array.

### Configuration

You can choose the model you want to use by clicking the model's name, the default model is "gpt-3.5-turbo-1106".

Click this button in the bottom right corner of the `Language Model Chat` action to open the configuration panel.

There are five settings in the configuration panel, as shown below.

**Temperature**

- "Temperature" controls the randomness of the model's output.
- The higher the model temperature, the more random the output is.

**Maximum Output Tokens**

- "Maximum Output tokens" specifies the maximum number of tokens to generate.
- Can use up to 40,000 tokens(the limit for models vary), including prompt and model returned content.

**JSON Response**

- "JSON Response" button enable JSON mode, which guarantees the messages the model generate are in JSON format.
- NOTE: This feature is a beta feature and only supported by OpenAI's "gpt-4-1106-preview" model now.
- NOTE: When you use this feature, make sure the word "JSON" is in the context. Otherwise, the OpenAI's API will throw an error.

**Seed**

- The "Seeds" is a parameter that can be specified when using the `Language Model Chat` and `Language Model Completion` actions.
- It helps to ensure consistent outputs by making the system sample deterministically, resulting in the same result for repeated requests with the same seed and parameters.
- NOTE: This feature is a beta feature and may not be supported by all models.

**Stop Words**

- Stop words are used to make the model stop at a desired point, such as the end of a sentence or a list.

On the top right of the action, there are two more things to configure: "Stream Mode" and "Cache Mode".

**Stream**

- This option allows you to receive partial chat responses as they are being generated, rather than waiting for the entire completion to be finished before receiving a response.

- By setting stream mode, you can start processing or displaying the beginning of the chat before the full response is received.

**Cache**

- Caching involves storing frequently accessed data to improve response times without making repeated calls to a model.
- If you use the cache mode, the model will cache the response and return the cached response when the same request is made again. This will make your agent run faster.

## Message Format

Rebyte uses a similar message format as OpenAI. The message format is a JSON object with the following fields:

- Role: could be one from 'user', 'system', or 'assistant'.

- Content: content of this message.
- Name(optional): name of role.
- Context(optional): context of this message.

**Message Format Examples**

1.Simple message

```
{
    "content": "Hello, how are you?",
    "role": "user"
    "name": "meowoof"
    "context": {
        "files": [
    {
                "uuid": "f3610fef-f490-4675-81fe-
df04735f5058",
                "name": "file1.pdf"
            },
            {
                "name": "f3610fef-f490-4675-81fe-
df04735f5059",
                "content": "file2.txt"
            }
    }
}
```

2.List of messages

```
[
    {
        "content": "You're talking to a chatbot!",
        "role": "system"
    },
    {
        "content": "Hello, how are you?",
        "role": "user"
        "name": "meowoof"
    },
    {
        "content": "I am fine, thank you.",
        "role": "assistant"
        "name": "rebyte"
    },
    {
        "content": "what's your plan today?",
        "role": "user"
        "name": "meowoof"
    }
]
```

## Example Agent

- Language Model Chat

# Language Model Completion

Language Model Completion

## Usage

We provide `Language Model Completion` action to let the language model complete your prompt.

### Specification

#### Prompt

- This is the prompt that will be sent to the model.
- The model will complete your prompt and respond with the completed content.

### Configuration

The configuration is the same as Language Model Chat.

You can choose the model you want to use by clicking the model's name, the default model is "gpt-3.5-turbo-1106".

Click this button in the bottom right corner of the `Language Model Completion` action to open the configuration panel.

There are five settings in the configuration panel, as shown below.

#### Temperature

- "Temperature" controls the randomness of the model's output.
- The higher the model temperature, the more random the output is.

#### Maximum Output Tokens

- "Maximum Output tokens" specifies the maximum number of tokens to generate.
- Can use up to 40,000 tokens(the limit for models vary), including prompt and model returned content.

#### JSON Response

- "JSON Response" button enable JSON mode, which guarantees the messages the model generate are in JSON format.
- NOTE: This feature is a beta feature and only supported by OpenAI's "gpt-4-1106-preview" model now.
- NOTE: When you use this feature, make sure the word "JSON" is in the context. Otherwise, the OpenAI's API will throw an error.

#### Seed

- The "Seeds" is a parameter that can be specified when using the `Language Model Completion` and `Language Model Completion` actions.
- It helps to ensure consistent outputs by making the system sample deterministically, resulting in the same result for repeated requests with the same seed and parameters.
- NOTE: This feature is a beta feature and only supported by OpenAI's model.

**Stop Words**

- Stop words are used to make the model stop at a desired point, such as the end of a sentence or a list.

On the top right of the action, there are two more things to configure: "Stream Mode" and "Cache Mode".

**Stream**

- This option allows you to receive partial chat responses as they are being generated, rather than waiting for the entire completion to be finished before receiving a response.

- By setting stream mode, you can start processing or displaying the beginning of the chat before the full response is received.

**Cache**

- Caching involves storing frequently accessed data to improve response times without making repeated calls to a model.
- If you use the cache mode, the model will cache the response and return the cached response when the same request is made again. This will make your agent run faster.

### Message Format

Rebyte uses a similar message format as OpenAI. The message format is a JSON object with the following fields:

- Role: could be one from 'user', 'system', or 'assistant'.

- Content: content of this message.

- Name(optional): name of role.

- Context(optional): context of this message.

#### Message Format Examples

1.Prompt Example

```
Your role is that of a text editor.
You are expected to peruse the texts provided to you,
comprehending them fully, and then distill and summarize
them for me. The summary should encapsulate the main theme
and essential details of the original text. It should be
succinct and expressed in your own words.
The contents that need to be summarized will be enclosed
within three single quotation marks.

The summary should be conducted in accordance with the
following regulations:
1. Thematic Statement: Succinctly summarize the main theme
or key point of the original text.
2. Key Details: Enumerate the crucial details or facts from
the original text that support the main theme or point.
```

```
3. Overall Conclusion: Distill the conclusion of the
original text or the position of the author.

Please organize the summary according to the following
structure and reply me:
Introduction: Introduce the main theme or background of the
original text.(New line)
Body Paragraph: List and explain the key details and
arguments from the original text, summarizing them in your
own words. (New line)
Conclusion: Summarize the main points of the original text
or present the author's conclusion.(New line)

This is the content that requires summarization:

please reply to me with the phrase: "I apologize for being
unable to retrieve content from the URL you provided. Please
verify the correctness of the web address"

{{CODE_1.content}}
```

## Example Agent

- Language Model Completion

# Data

## Data

# File Loader

# File Loader

The `File Loader` action gives you the ability to upload files to your agent.

We support various data types, including: DOC, DOCX, IMG, EPUB, JPEG, JPG, PNG, XLS, XLSX, PPT, PPTX, MD, TXT, RTF, MD, TXT, RTF, RST, PDF, JSON, HTML and EML.

## Usage

- Add a `File Loader` action to your agent and select the file you want to upload to your agent.
- Specify the `file_id` of this file and use this unique `file_id` whenever you want to refer to this file.
- If you connect your agent to an app, app users can upload files in the app, and the `file_id` will be passed to the agent.
- The output of this action is in JSON format, containing extracted data from the file.

## Example Agent

- File Loader

# Dataset Loader

# Dataset Loader

The `Dataset Loader` action allows you to load data from the agent's datasets. The data can be used in subsequent actions.

### Usage

- Add a `Dataset Loader` action to your agent and select the dataset you want to load from.
- Refer to this data by using the dataset's name, like: `` in downstream actions.

### Example Agent

- [Dataset Loader]

# Code

## Code

Here you can write any Javascript code to give your agent more power.

# Tools

## Tools

# Call Agent

# Call agent

### Description

You can call another Agent.

### Parameters

- AgentId
  - The AgentId of the Agent being called.
- projectId
  - The projectId where the called Agent is located.
- apiKey
  - The apiKey used to access the corresponding Agent.
- version
  - The version number of the agent that's being called; The default setting is the lastest version.
- inputArgs
  - The input arguments corresponding to inputs, refer to the API; the parameters passed here must be a string.
- blocking
  - Just fill in true.

### Output

JSON

| parameter | type | description |
|---|---|---|
| status.run | object | the result of the calling(success or fail) |
| results | object | the response |
| others | object | the status of the agent being called |

# Curl Request

# Http Request

### Description

Send Http/Https Requests

**Parameters:**

### spec

| Parameter | Description |
|---|---|
| scheme | HTTP or HTTPS |
| method | GET or POST |
| host | host of URL |
| header | header of HTTP request |
| body | post body, only available in POST request |

### config

None

# Google Search

Google Search

### Description

Search the web using Google Search API

### spec

| Parameter | Description |
|---|---|
| query | query sent to Google |
| number of results | number of search results returned from Google |

### config

None

#### Example

- Google Search

# Http Request Maker

Http Request Maker

### Description

This module supports making requests to specific URLs and retrieving results.

### Parameters

- Type
  - POST
  - GET
- Protocol
  - HTTP
  - HTTPS
- Address
  - Request address
  - Supports tera syntax, use to reference the output of the previous Action.

- Headers

  - Some URLs require sending specific headers with the request, usually to provide more metadata about the operation being performed.

  - You can define what needs to be sent, or you can get the content output from the previous Action using `env.state.actionname`

    ```
    _fun = (env) => {
    return {
    "Content-Type": "application/json"
      }
    }
    ```

- Body
  - Whenever you need to add or update structured data, you need to send body data with the request. For example, if you want to send a request to add a new customer to the database, you can include the details in the Body.
  - You can define what needs to be sent here, or you can get the content output from the previous Action.

### Output

Depends on the specific request URL.

# Knowledge Search

## Knowledge Search

Search action allow agents to make queries to specified knowledge base. This is useful for retrieving information from a knowledge base , later those retrieved information can be used as context for further generation. Retrieval Augmented Generation (RAG) agent is a good example of this use case.

Search Action consist of the following fields:

- **Query:** Provide the desired prompt to look for inside your knowledges
- **Knowledge:** Knowledge that you want to search in. You can select multiple knowledge bases. Technically, each knowledge in rebyte can be unique identifier by its knowledge name and project who owns it.
- **Include Tags:** Add tags that you want to include in your search. Notes tags are exact match which means if you add "tag1" it will only include documents that have "tag1"
- **Exclude Tags:** Add tags that you want to exclude in your search. Notes tags are exact match which means if you add "tag1" it will exclude all documents that have "tag1"
- **Max Documents:** Maximum number of chunks to return. Chunk size is determined when knowledge is created.

### Spec

| Parameter | Description |
|---|---|
| query | query for searching the knowledge |

### Config

| Parameter | Description |
|---|---|
| knowledge to search | list of knowledge to search in |
| number_of_results | number of results you'd like to have |
| include tags | |
| exclude tags | |

# Web Page Crawler

## Web Scraping

Goes to a specified URL and retrieves the HTML from the page.

**Parameters:**

- **URL:** The URL that you want to get HTML data from

### spec

| Parameter | Description |
|---|---|
| URL | URL to get content from |
| css_selector | only get content from specific CSS selector |

### config

none <!-- **Example**

- Web Crawl -->

# Control Flow

Control Flow

# If-Else

## If Else

This action will run the actions if some condition is met.

The condition is a Javascript code that returns a boolean value.

The code can reference the output of previous actions by using the `env.state.ACTION_NAME` syntax.

For example, if the previous action is named `ACTION_1`, then the code can reference the output of `ACTION_1` by using `env.state.ACTION_1`.

## Loop-Until

## Loop

This action will run the actions inside the loop until the condition is met. The condition is a javascript code that returns a boolean value.

The code can reference the output of previous actions by using the `env.state.ACTION_NAME` syntax.

For example, if the previous action is named `ACTION_1`, then the code can reference the output of `ACTION_1` by using `env.state.ACTION_1`.

## Map-Reduce

## Map Reduce

Map over an array and executes a sequence of actions in parallel. This is useful for doing multiple actions in parallel, such as scraping multiple web pages in parallel.

**Spec:**

- **MapOver:** Reference of an action name that outputs an array
- **Repeat:** Integer value that specifies the maximum iteration count
    - If repeat is not specified, then the iteration will stop when the array is exhausted. There is a hard limit of **64** iterations.
    - If repeat is specified, then the iteration will stop when the array is exhausted or the iteration count reaches the repeat value, whichever comes first.

**Config:**

- None

**Output**

- Each action inside map-reduce action will output **an array** of values.

**Error Handling**

- MapOver must refer an action that outputs a non-empty array.

## Early Return

## Early Return

This Action will return the result immediately without running the rest of the actions in the agent.

This is useful when you want to return the result early without running the rest of the actions.

# Overview

## Overview

Group agent allows you to combine the abilities of multiple agents into one agent.

You can specify the agents you want to use in the group agent, the goal you want to achive using this group agent.

You can also set the maximum number of previous runs on thread and maximum rounds of conversation between agents.

It is useful when you want to combine multiple agents to solve a complex problem.

# Quick Start

Quick Start

👷 Content in production, check this out sometime later!

# Overview

## Overview of Testing

Here, we will introduce some basic concepts and show you how to test your agent.

### Create Testing Dataset

- Datasets are used to test your agent. You can create a testing dataset by clicking the "Create Dataset" button in the dataset list page.

### Test and Debug

- Once you've created the testing dataset, you can go the the "design" page and choose the dataset.
- click the "Run TestCases" button to run the agent with your selected dataset.
- The running results will be shown under each action.
- If there's no bug in the running process, you will see green check marks on the left side.

- Otherwise, you will see red cross marks on the action that went wrong.

- After every run, we record the running results in the "Runs" tab. We provide the RunID, the time of the run, the status of the run, and the agent used in the run.
- Click on the "Run ID" of each run to see the details of the run.

### Playground

- If you would like your testing to more flexible, you can use the playground to test your agent.
- You should make extensive use of this capability to test your agent before publishing.

# Datasets

# Datasets

After designing your agent, you can test it with datasets.

You can run the dataset with the "Run Testcases" and observe the agent's response and maybe troubleshoot issues based on it.

## Dataset Creation

- Navigate to the `Datasets` tab on the top.
- Click `Create Dataset`.
- Use "Add Column" to add a new field to your dataset. "Add Row" to add a new dataset item.

## Data types supported

We support the following data types:

- String
- Number
- Boolean
- JSON Object

## Usage

Datasets can be used in two ways:

#### 1.Load data for subsequent actions

The loaded dataset can be used by subsequent actions.

This is very useful when the dataset contains example data for few shots prompting.

#### 2.Determine the input shape of the agent

The the dataset of the `Input` action determines the shape of the input of the whole agent.

# Playground

## Playground

Here, you can test your agent with arbitrary input data.

Note that the data you use in playground must be a json object.

# Runs

## Runs

Here we display all of your running records.

You can filter them by "Status" and "Last Update".

Click on the "Run ID" to see the detailed information of you run.

# Deploy Your Agent

## Publish Your Agent

Once you have finished designing and testing your agent, click "Deploy" to publish a new version or checkout the current version.

- After deploying your agent, you can use it to create your ReByte app.

- Or you can integrate it into your own applications.

- To check the different versions you've already published, click the "Versions" tab.

# Apps with Chat Interface

## Apps with Chat Interface

# Quick Start

# Quick Start

We will show you how to build an app with chat interface.

## Step 1: Create an App

- Navigate to the "My Apps" tab in the sidebar and then click "Create App" on the top right. Choose "App With Chat Interface".

- Fill in the information about the app.

- Select the agent you want to base your app on, remember to select the version of your agent.

- Click "Create App with Chat UI" and that's it!

## Step 2: Use your App

🎉 Now you can use your App in the chat interface!

## Apps with Customized UI

Apps with Customized UI

# Quick Start *

# Quick Start

We will show to how to build an app using the customized UI.

Before building an app, you need to think about **what you want your app to do**. Then, think about the **input,data flow and output** of the agent this app will connect with.

For example, if you want to build an App that can get the 7-day weather forcast for you, first think about that the user's query will be, how the agent will get the weather forcast, and how the agent should return the following 7 days' to the user(perhaps JSON format). Then, you can start building your app and agent.

## Step 1: Create an App

- Navigate to the "My Apps" tab in the sidebar and then click on "Create App" on the top right.
- Choose "App With Customized UI".
- Use the tools on the bottom of this page to draw a basic draft of your app interface.
- Click "Make It An App" on the bottom right.
- Fill in the text prompt describing what you want your app to do.
- You can then click "Draw it" or "Make It An App".

## Step 2: Design you App

If you want to make some changes to your app, you can change your app in two ways:

- Image prompt: Draw the changes you want to make on the canvas.
- Text prompt: Click the "Make It An App" and input the text prompt describing the changes you want to make.

Click "Make It An App" again and we will make changes for you based on your prompt and create a new version of your app.

- NOTE: This may not give you the desired results on just one try. We recommend you to try and improve one thing at a time and create more versions until you get the best results.

## Step 3: Connect your App with Agent

- Click the "Connect to Agent" on the top right and choose the agent you want to connect with.
- NOTE: You must be familiar with the agent's function. Make sure the agent is suitable for your app, otherwise it will not connect successfully.

- Use the prompts to describe how you want to use the agent in your app.

- Click "Make It An App" and we will automatically connect your app with the chosen agent.

## Tips

- To get better results, try describing your functions/changes you'd like to make one at a time and in details.

- Make good use of the drawing tools to tell us the changes you want to make to the current app.

- Each time you click "Make It An App", we will generate a new version of your app. If you find the results unsatifactory, simply go back to previous versions and start from there again.

# Overview

# Knowledge

In ReByte, we provide knowledge base capability, enabling agents to interact with your private data. Users can store and manage their personal data in the knowledge.

Many data sources are supported, including: local files, notion, github, discord, etc.

## How Knowledge Works?

### Document

Each knowledge contains a list of documents, each document is identified by a unique document id within the knowledge.

### Chunk

Document will be chunked into many chunks, each chunk identified by a unique chunk id within the document. Chunks will be sent to LLM embedding service to get the embedding vector. When creating knowledge, user can specify the chunk size, which will determine how many chunks a document will have. Typically, chunk size range from a hundred to a few thousand tokens.

# Quick Start

# Quick Start

Here we tell you now to create and use knowledge.

## Step 1: Create Knowledge

- Navigate to the "Knowledge" tab in the sidebar and then click "Create Knowledge".
- Fill in the knowledge information and choose the source type for your knowledge. Here we choose "Self Managed" since we are going to use local files.

## Step 2: Upload Files

- Click "Upload File" and upload your local file.
- Once uploaded, we will generate a unique document id for your file.
- Click "Insert Document" to upload this file.

**Now you have created your first knowledge!**

## Step 3: Use Your Knowledge

- You can search knowledge in the **search box** and hit ENTER.

Search Results

Or you can use the knowledge in your **agent**(shown in Step 4).

## Step 4: Use Knowledge in Agent

- Use "Search Knowledge" action and choose the knowledge you want to use.

Search Results

- You can fill in the query to the knowledge search and set the number of query results you'd like to have.
- The results are the same as searching in the "Knowledge" page

# Knowledge Types

# Knowledge Types

Many data sources are supported, including: local files, Notion, Github, Discord, etc.

## Local files

You can upload your files and store them in the knowledge.

Supports files types: DOC, DOCX, IMG, EPUB, JPEG, JPG, PNG, XLS, XLSX, PPT, PPTX, MD, TXT, RTF, MD, TXT, RTF, RST, PDF, JSON, HTML, EML

## Notion

Connect your Notion account and import your Notion pages into the knowledge.

- Choose Notion as the data source for your knowledge.

  notion

- In the pop-up page, login to your Notion account and choose the pages you want to import into your knowledge.

  notion-2

- We stay synced with Notion, makingå sure the content is updated in time.

  notion-3

# Prompt Template Language

## Prompt Template Language

The Prompt Template Language is a simple language that allows you to create a prompt template that can be used to generate prompts for your LLM models. The Prompt Template Language is a subset of the Jinja templating language. Here are some examples of how you can use the Prompt Template Language, for more information on the syntax, see the Jinja documentation.

{{ and }} for expressions

```
{{ 1 + 1 }}
```

For referencing variables

```
{{ variable_name }}
```

for statements

```
{% raw %}
{% for i in range(10) %}

    {{ i }}

{% endfor %}
{% endraw %}
```

{# and #} for comments To comment out part of the template, wrap it in {# #}. Anything in between those tags will not be rendered.

```
{# This is a comment #}
```

Construct and attributes can be accessed by using the dot (.) like {{ product.name }}. Specific members of an array or tuple are accessed by using the .i notation, where i is a zero-based index. In dot notation variable can not be used after the dot (.).

```
{{ product.name }}
{{ product[0] }}
```

Tests can be used against an expression to check some condition on it and are made in if blocks using the is keyword. For example, you would write the following to test if an expression is odd:

```
{% raw %}
{% if number is odd %}

    This is an odd number

{% endif %}
{% endraw %}
```

Tests can also be negated:

```
{% raw %}
{% if number is not odd %}

    This is not an odd number

{% endif %}
{% endraw %}
```

# Visibility

## Visibility

In ReByte, you can set the visibility of your agent, app and knowledge.

### Agent

For agents, we provide 3 visibility options:

- Public: Anyone can see and clone your agent, but only you can edit the agent and see the runs.
- Private: Only you can see your agent.
- Unlisted: Anyone with the link can see your agent, but only you can edit the agent and see the runs.

### App

For apps, we provide 3 visibility options:

- Public: Anyone can see and use your app.
- Private: Only you can see and edit your app.
- Unlisted: Anyone with the link can use your app, but only you can edit the app.

### Knowledge

For knowledge, we provide 2 visibility options:

- Public: Anyone can see your knowledge.
- Private: Only you and people you invite can see your knowledge.

# Search Engine Agent

# Search engine agent

Summarize web page contents for you.

### Setup the testing data in `Datasets`

```
[
  {
    "role": "user",
    "content": "Who won the last UEFA Champions League?"
  },
  {
    "role": "assitant",
    "content": "The last winner of the UEFA Champions League
was Manchester City in the 2022-23 season. This was their
first title in the competition. Real Madrid holds the record
for the most victories in the Champions League, having won
it 14 times [0][1]."
  },
  {
    "role": "user",
    "content": "Who scored the winning goal?"
  }
]
```

### Use `Code Action` to get the last message

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
  return env.state.INPUT.messages.slice(-1)[0].content
}
```

### Use `Code Action` to get the chat history

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
 return env.state.INPUT.messages.slice(0,
env.state.INPUT.messages.length - 1).map((m) =>
m.content).join("\n")
}
```

### Use `Code Action` to extract content

```
{# Your prompt here, for example: 'Answer those question
based on the following content.' #}
{# Begin your prompt below: #}
This is the question: {{EXTRACT_QUESTION}}
This is the conversation history: {{HISTORY}}

Add necessary contexts to the question with the help of
conversation history. If the contexts are irrelevant don't
change the question. Give the question here:
```

### Use `Google_search` to search for related contents

```
{{REFINED_QUESTION.completion.text}}
```

### Use `Code Action` to extract content

```
// Extracts title, snipet and link from the google search's
organic results.
// capped at 3
_fun = (env) => {
  if (!env.state.GOOGLE_SEARCH.organic_results) {
    return [
      {
        title: "",
        link: ""
      }
    ]
  }

  return env.state.GOOGLE_SEARCH.organic_results.map((r) =>
{
    return { title: r.title, link: r.link, snippet:
r.snippet };
  }).slice(0, 3);
}
```

### Map the results for parallel processing

```
SEARCH_EXTRACT
```

### Use `WEB_CRAWL` to scan the page

```
{{SEARCH_RESULT_LOOP.link}}
```

### Use `Code Action` to extract content

```
// for each link, crawl its content and capped at 2000 bytes
_fun = (env) => {
  return {
    content: env.state.WEB_CRAWL_1.data ?
env.state.WEB_CRAWL_1.data[0].results[0].text.slice(0, 2000)
: "",
  };
}
```

### Use LLM to analyze the contents

```
{# Your prompt here, for example: 'Answer those question
based on the following content.' #}
{# Begin your prompt below: #}

Given the following question:
"""
{{EXTRACT_QUESTION}}
"""

Extract the text from the following content relevant to the
question and summarize it:
"""
{# {{GOOGLE_REFERENCES.content}} #}
{{SEARCH_RESULT_LOOP.snippet}}
"""

Extracted summarized content:
"""
```

### Use `Code Action`` to extract content

```
_fun = (env) => {
  return {
    summary:
```

```
env.state.MODEL_SUMMARIZE.completion.text.trim(),
    link: env.state.SEARCH_RESULT_LOOP.link
  };
}
```

Use `Code Action` to extract content and make prompt

```
const _example = (example) => {
  // prompt = `QUESTION: ${example.question}\n`;
  let prompt = "CONTENT:\n";
  prompt += '"""\n';
  example.forEach((d, i) => {
    if (i > 0) {
      prompt += '\n';
    }
    prompt += `link [${i}]: ${d.link}\n`;
    prompt += `content: ${d.summary.replaceAll('\n', '
')}\n`;
  });
  prompt += '"""\n';
  console.log(prompt)
  return prompt;
}

_fun = (env) => {
  prompt = 'Given the following questions, reference links
and associated content, create a final answer with
references. If some of answer can be formatted in table
format, format in table format. Answer should be accurate
and concise. \n\n Never tell me "As a language model ..." or
"As an artificial intelligence...", I already know you are a
LLM. Just tell me the answer. \n\n';
  // env.state.EXAMPLES.forEach((e) => {
  // prompt += _example(e);
  // prompt += `FINAL:\n"""\n${e.final}\n"""\n`;
  // prompt += "\n";
  // });
  prompt += "QUESTION:" + env.state.EXTRACT_QUESTION + "\n";
  prompt += _example(env.state.FORMAT_SUMMARY);

  prompt += `FINAL:\n"""\n`;

  return { prompt }
}
```

### Send the prompt to LLM

```
{{FINAL_PROMPT.prompt}}
```

### Extract `OUTPUT_STREAM` as output

```
{{FINAL_PROMPT.prompt}}
```

# Knowledge Based Chatbot

# Knowledge Based Chatbot

### Create Knowledge

Under Knowledge tab, click `Create Knowledge`. Fill in the name and description, choose the correct knowledge type and set up the maximum chunck of size.

For more details, please check the Knowledge section.

### Create Agent

To create an agent, choose `retrieval based chatbot with own data` as template.

In `RETRIEVALS`, choose the `Knowlegde` you created above. Click `run TestCases`. If there's no problem with the test, you can deploy the agent. For more details, please check the Agent section.

### Create Apps

Under the `Apps` tab, choose `Create App` and fill in the `Chat App Name` and `Description`. Choose the agent and knowledge that you just created in the above steps.

After completion, you can communicate with the chatbot, and your chatbot will have the knowledge in the knowledge base choose.

# Web Page Summary Agent

# Web page summary agent

Summarize web page contents for you.

### Setup the testing data in `Datasets`

```
[
 {
    "role": "user",
    "content": "https://book.douban.com/subject/30360449/"
  }
]
[
  {
    "role": "user",
    "content": "Introduce this web page:
https://towardsdatascience.com/an-introduction-to-openai-
function-calling-e47e7cd7680e"
  }
]
```

### Extract content from `Code Action`

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
  return env.state.INPUT.messages.slice(-1)[0].content
}
```

### Use LLM to process user's input

```
You're a link extractor, which extracts http or https links
from given text, output the result as a json format, output
contains two fields, first is link, which contains the link
extracted from text, second is language, which contains
language code of given question.
Here's the given text, let's begin:
{{EXTRACT_QUESTION}}
```

### Extract content from `Code Action`

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
  const output =
JSON.parse(env.state.REFINE.completion.text)
  // if link exists, output link
  // if links exists, output the first element
  return {
    link: output.link ? output.link : (output.links ?
output.links[0] : ""),
    language: output.language ?? 'en'
  }
}
```

### Use `WEB_CRAWL Action` to collect information

### Extract content from `Code Action`

```
_fun = (env) => {
  let content = env.state.WEB_CRAWL_1.data ?
env.state.WEB_CRAWL_1.data[0].results[0].text.slice(0, 3000)
: "";
  content = content.replace(/\n/g, ""); // 去除换行符
  return {
    content: content,
  };
}
```

**Send the web page content to the LLM**

```
Your role is that of a text editor.
You are expected to peruse the texts provided to you,
comprehending them fully, and then distill and summarize
them for me. The summary should encapsulate the main theme
and essential details of the original text. It should be
succinct and expressed in your own words.
The contents that need to be summarized will be enclosed
within three single quotation marks.

The summary should be conducted in accordance with the
following regulations:
1. Thematic Statement: Succinctly summarize the main theme
or key point of the original text.
2. Key Details: Enumerate the crucial details or facts from
the original text that support the main theme or point.
3. Overall Conclusion: Distill the conclusion of the
original text or the position of the author.

Please organize the summary according to the following
structure and reply me:
Introduction: Introduce the main theme or background of the
original text.(New line)
Body Paragraph: List and explain the key details and
arguments from the original text, summarizing them in your
own words. (New line)
Conclusion: Summarize the main points of the original text
or present the author's conclusion.(New line)

This is the content that requires summarization:

please reply to me with the phrase: "I apologize for being
unable to retrieve content from the URL you provided. Please
verify the correctness of the web address"

{{CODE_1.content}}
```

**Extract content from `Code Action`**

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
 return {
   role: "assistant",
   content: env.state.OUTPUT_STREAM.completion.text,
   retrievals: env.state.RETRIEVALS
 }
}
```

**Output the final results**

# Bilibili Subtitle Agent

# Bilibili subtitle agent

Help you get bilibili subtitles and summarize the content.

### Setup the testing data in `Datasets`

```
[
  {
    "role": "user",
    "content": "BV1Lu411J7Z8"
  }
]
```

### Extract content from `Code Action`

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
  return env.state.INPUT.messages.slice(-1)[0].content
}
```

### Use LLM to process user's input

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
 return env.state.GET_BV.completion.text
}
```

### Extract content from `Code Action`

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
 return env.state.GET_BV.completion.text
}
```

### Use `Http Request Maker` and request for CID

```
api.bilibili.com/x/player/pagelist?bvid={{BV}}
```

### Extract content from `Code Action`

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
 return env.state.GET_CID.body.data[0].cid
}
```

### Use `Http Request Maker` and request for the url for subtitles

```
api.bilibili.com/x/player/v2?bvid={{BV}}&cid={{CID}}
```

### Extract content from `Code Action`

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
  // const data = JSON.stringify(env.state.GET_URL.body)
  // const regex = /<subtitle>(.*?)<\/subtitle>/;
  // const match = data.match(regex);
  // const subtitleURL = match ? match[1] : "";
  // const obj = subtitleURL.replace(new
RegExp("\\\\\"","gm"),"\"")
  // const result = JSON.parse(obj)['subtitles'][0]
['subtitle_url'].replace("//", "")
  const url =
env.state.GET_URL.body.data.subtitle.subtitles[0].subtitle_u
rl
  const result = url.replace(/\/\//g, "");
  return result

}
```

**Use `Http Request Maker` and request for the subtitles**

```
{{GET_CC_URL}}
```

**Extract content from `Code Action`**

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
  let arr = []
  for(let i in env.state.GET_CC.body.body) {
    arr.push(env.state.GET_CC.body.body[i].content)
  }
 return arr.join(',')
}
```

**Send subtitles to the LLM to summarize**

```
{% if GET_URL.body.data.subtitle.subtitles[0].subtitle_url
== "" %}
please reply to me with the phrase: "I apologize for being
unable to retrieve content from the URL you provided. Please
verify the correctness of the web address"

{% else %}
You are an AI with advanced comprehension and summarization
skills. Your task is to read the following passage and
provide a concise, clear summary that captures the main
points and key details.

Passage:
{{CC}}

Please provide a summary of the above passage and respond to
me in Chinese.

{% endif %}
```

**Extract content from `Code Action`**

```
_fun = (env) => {
  // use `env.state.Action_NAME` to refer output from
previous Actions.
 return {
   role: "assistant",
   content: env.state.OUTPUT_STREAM.completion.text,
   retrievals: env.state.RETRIEVALS
 }
}
```

**Output the final results**

# Collaboration

## Collaboration

To avoid incosistency, ReByte has provided features that allow you to work with your teams on building agents.

- If **two or more people** are on the same agent's page, we will automatically "lock" the page, which means no one can edit this agent.
- To unlock, click the "lock" icon hovering at the bottom.

# Teams

## Teams

# Overview

## Team Overview

Team is a concept that allows multiple users in same organization to collaborate on the same knowledge and agent, also enforce access control. Team can have multiple members, and each member can have different access level. Each login user has one default team, named after the user's display name, for example, if your display name is "John Smith", then your default team name will be "John Smith's Personal Project".

For example, if you are a developer in a company, you can create a team called "ACME" and invite your colleagues to join the team. Then you can create knowledge and agent under the team, and all your colleagues can access the knowledge and agent.

- User can belong to multiple teams, he can change his default team in the team switcher in upper left corner of the UI.
- Each team has its own API Key, which can be used to access all agents and knowledge that belongs to the team.
- Team has a unique ID to identify the team, you can find the team ID in the team settings page. Team ID is particularly useful when you want to use ReByte via SDK.

# Access Control

# Access Control

Rebyte is built with fine-grained access control. You can control the visibility of your agents, knowledge, apps and other resources in Rebyte. We believe that this is of vital importance for collaboration and sharing AI applications in enterprise environments.

## Types of Accounts

In Rebyte, there are two types of accounts: Personal and Team.

We have designed corresponding permission rules for the core functions of the Rebyte system: agent, knowledge, and app.

## Personal Accounts

Personal accounts are for individual users, who can develop agents and applications the way they like and use applications in the Rebyte community.

For personal accounts, upgrading to a "pro" account allows the sharing of agents, knowledge, and apps with other users so that they can view and use.

Sharing an agent means others can view the agent you have created, including its workflow and datasets. However, they cannot run or modify it. If another user wants to use it, they can clone the agent into their own account to run and make adjustments.

Sharing knowledge allows other users to view the content in your knowledge base.

Sharing an app allows other users to use the app you have built.

Further details are available in the following table:

| Personal-Access | Other users | Personal user |
|---|---|---|
| App | | |
| Private | | list,use,edit,create |
| Public | list,use | list,use,edit,create |
| Knowledge | | |
| Public | list,view | list,use,edit,create |
| Private | | list,use,edit,create |
| Agent | | |
| Public | list,view,clone | list,use,edit,create |
| Unlisted | view,clone with link | list,use,edit,create |
| Private | | list,use,edit,create |

## team accounts

Team accounts allows for collaboration among teams, and there are three roles in a team account.

- Builders (builder, admin, owner): develop and maintain the team's agents, knowledge, and apps.
- Users: use applications within the team.
- Non-team users: users who are not part of the team.

For team accounts, builders can develop, design, and use the agents, knowledge, and apps within the team account. Users can view and use the apps, knowledge, and agents in the team account.

Regarding agents, users can view all agents' workflows and datasets within the team and can copy them to their personal accounts to modify and run. When an agent is set to External Unlisted, users outside the team can use a link to view the agent. When an agent is set to Public, it appears in the team agent list for external users.

Regarding knowledge, it can be set so only admins and owners can create knowledge, while builders can maintain it. External Public set knowledge can be viewed by users outside the team.

Regarding apps, there are four access-levels. Private apps can only be seen by builders, which is useful for maintaining apps that are still under development.team Unlisted apps can be accessed and used by team users via a link. Apps set to Team Public can be viewed and used by all team members on the team app page. External Public means that users outside the team can also view and use the app.

For example, in an administrative team, the owner and admin manage the team's resources, builders are responsible for designing and building agents, knowledge, and apps. And users are responsible for viewing and using the agents, knowledge, and apps designed by the builders. When an app is not yet complete, a builder can set its access-level to Private. Once the design is finished, it can be set to Team Public for team members to use. Setting it to External Public allows sharing with users outside the team.

Further details are available in the following table:

| **Team-Access** | **Non-team user** | **Team user（member）** | **Team Builder（admin、owner）** |
|---|---|---|---|
| Team-App | | | |
| Private | | | list,use,edit,create |
| team Unlisted | | Use with link | list,use,edit,create |
| team Public | | list,use | list,use,edit,create |
| External Public | list,use | list,use | list,use,edit,create |
| Team-Knowledge | | | |
| External Public | list,use | list,use | list,use,edit,create |
| team Public | | list,use | list,use,edit,create |
| Team-Agent | | | |
| team Public | | list,view（design,dataset）,clone | list,view,edit,create |
| External Unlisted | view,clone with link | list,view,clone | list,view,edit,create |
| External Public | list,view,clone | list,view,clone | list,view,edit,create |

# Overview

## Overview

In ReByte community, we present you with a lot of carefully-designed apps and agents that are ready to use.

To enter the community, click the "Community" tab on the left side of the screen.

Here we have listed many different apps and agents, feel free to use them or clone them to add more functions.

🎊 We will regularly update the agents and apps in the ReByte community, making sure that you can always find the best ones here.

# Agents

## Community Agents

Under the "Agents" section, you can find agents that are built by the ReByte team.

You can use these agents freely and explore their functions.

# Apps

## Community Apps

Under the "Apps" section, you can find apps that are built by the ReByte team.

# Privacy Policies

## Privacy Policies

At RealChar AI, we understand the importance of your privacy and we are committed to protecting it. This Privacy Policy explains how we collect, use, and disclose your personal information when you use our website, products, and services.

We take your privacy seriously and strive to ensure that your personal information is always kept safe and secure. To that end, we have implemented a number of security measures to prevent unauthorized access to your data.

Information we collect We may collect personal information such as your name, email address, and phone number when you sign up for our services or contact us for support. Additionally, we may collect information about your usage of our website and services, such as your IP address and browser type. We believe that transparency is key when it comes to collecting personal information, which is why we are always upfront about the data we collect and how we use it.

How we use your information We use your personal information to provide and improve our services, to communicate with you, and to comply with legal obligations. Your information may also be used for research and analysis purposes to improve our products and services. We understand that your personal information is valuable and we take great care to ensure that it is only used for the purposes for which it was collected.

Disclosure of your information We do not sell or rent your personal information to third parties. However, we may share your information with our trusted partners who assist us in providing our services. We may also disclose your information if required by law or to protect our rights and interests. We believe that your personal information is just that - personal. That's why we will never share your data with anyone without your express consent.

Security of your information We take reasonable steps to ensure the security of your personal information and prevent unauthorized access, use, or disclosure. However, no data transmission over the internet or electronic storage system can be guaranteed to be 100% secure. We understand that your personal information is precious and we will always strive to keep it safe and secure.

Changes to this Privacy Policy We may update this Privacy Policy from time to time and will notify you of any changes by posting the new policy on our website. Your continued use of our services after any such modifications will constitute your acknowledgement of the modified Privacy Policy and agreement to abide and be bound by the modified policy. We believe that transparency is key when it comes to privacy, which is why we will always keep you informed of any changes we make to our policies.

Contact us If you have any questions or concerns about our Privacy Policy, please contact us at support@rebyte.ai. We are always happy to help and will do our best to address any issues you may have.

# Terms of Service

## Term of Service

Welcome to RealChar AI! Please read these terms of service carefully before using our services. By accessing or using our services, you agree to be bound by these terms.

1. Use of Services RealChar AI provides its services to you, subject to these terms of service. You may use our services only for lawful purposes and in accordance with these terms. By using our services, you represent and warrant that you are at least 18 years old.

2. Intellectual Property Rights The content and materials available onRealChar AI, including but not limited to text, graphics, logos, button icons, images, audio clips, data compilations, and software, are the property of RealChar AI or its content suppliers and are protected by copyright laws.

3. User Content You retain all rights in, and are solely responsible for, the content you submit to RealChar AI. You grantRealChar AI a non-exclusive, worldwide, perpetual, irrevocable, royalty-free, sublicensable, and transferable license to use, reproduce, distribute, prepare derivative works of, display, and perform your content in connection with the services provided byRealChar AI and its successors and affiliates.

4. Limitation of Liability RealChar AI shall not be liable for any indirect, incidental, special, consequential, or punitive damages, or any loss of profits or revenues, whether incurred directly or indirectly, or any loss of data, use, goodwill, or other intangible losses resulting from your access to or use of or inability to access or use the services provided by RealChar AI, including, without limitation, any damages resulting from or related to any failures of performance, error, omission, interruption, defect, delay in operation or transmission, computer virus, or line failure.

5. Indemnification You agree to indemnify, defend, and hold harmless RealChar AI, its affiliates, officers, directors, employees, agents, licensors, and suppliers from and against all claims, liabilities, damages, judgments, awards, losses, costs, expenses, or fees (including reasonable attorneys' fees) arising out of or related to your violation of these terms of service or your use of the services provided by RealChar AI.

6. Termination RealChar AI reserves the right to terminate your access to all or any part of the services provided by RealChar AI at any time, with or without cause, with or without notice, effective immediately. If you wish to terminate your account, you may simply discontinue using the services provided by RealChar AI.

These terms of service constitute the entire agreement between you andRealChar AI and govern your use of the services provided by RealChar AI, superseding any prior agreements between you and RealChar AI. If any provision of these terms of service is held to be invalid or unenforceable, such provision shall be struck and the remaining provisions shall be enforced.