**Bing script and code for bookkeeper**

Goal Of Project:

Utilize a group of agents with AutoGen that can handle repetitive bookkeeping task. I have asked Bing to help with the steps.

1. Receipts – able to upload a receipt and the system read the receipt and be able to learn how to classify the receipt.
   1. Most of this is repetitive.
      1. Example: Gas is always Auto Fuel
      2. Oil change is always vehicle expense
      3. Restaurant is always meals 100% or Entertainment; should ask and apply it to the correct account
      4. FedEx, UPS and USPS is always shipping
2. Invoices- Same as receipts. Should be able to identify 2 to 4 key fields one being the vendor – classify it one time and then from there forward it would properly classify it with the correct account code
3. Import to Quick Books
   1. Should be able to export to a csv file that can be uploaded to quick books. At some point we will try to get a direct api access if it makes sense.

Sure, I can give you a detailed plan on how to set up an AI accounting agent that can handle your accounting and financial tasks using AutoGen. Here are the steps you need to follow:

* Install AutoGen from GitHub or PyPI. You can use the command pip install pyautogen to install it from PyPI.
* Define an accounting agent using the autogen.Agent class. You can specify the name, description, and capabilities of the agent.

AGENT INFORMATION

1. Receipt Agent
   1. Receipts – able to upload a receipt and the system read the receipt and be able to learn how to classify the receipt.
      1. Most of this is repetitive.
         1. Example: Gas is always Auto Fuel
         2. Oil change is always vehicle expense
            1. Restaurant is always meals 100% or Entertainment; should ask and apply it to the correct account
         3. FedEx, UPS and USPS is always shipping
2. Invoice Agent
   1. Invoices- Same as receipts. Should be able to identify 2 to 4 key fields one being the vendor – classify it one time and then from there forward it would properly classify it with the correct account code
   2. Examples:
      1. ALLDATA – subscription IT Expense – acct# 7115
      2. MultiCopy- is always printing – acct # 8320
      3. Motor Information system – subscription IT Expense – Acct #7115

This should be an app that is formatted for the phone, it can be a streamlit app that is used via web so it doesn’t have to be on an app store.

The goal is it would ask for Receipt or Invoice we select one, upload the document or email it to an email address we set up. [Bookkeeper@dealeradmin.io](mailto:Bookkeeper@dealeradmin.io) or [bookkeeper@gmail.com](mailto:bookkeeper@gmail.com).

When we upload it it will analyze it and if it recognizes it, it should properly classify it and put in in the sheet and save the picture to a file on one drive. If it doesn’t recognize the receipt it should ask questions on vendor, activity so it can properly code it.

If a restaurant it should ask. 100% deduction – your are traveling and this is meals for travel. 50% deduction for meals & Entertainment. If 50% it should ask for people, and business to be able to classify it.

For example, you can write something like this:

from autogen import Agent

accounting\_agent = Agent(

name="accounting\_agent",

description="An agent that can handle accounting and financial tasks",

capabilities=["llm", "tool", "human"]

)

* Define the LLM configuration for the accounting agent. You can use the llm\_config parameter to provide a list of LLM inference endpoints that the agent can use. For example, you can write something like this:

accounting\_agent.llm\_config = {

"config\_list": [

{

"name": "gpt4",

"endpoint": "https://api.openai.com/v1/engines/gpt-4/completions",

"key": "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

},

{

"name": "gpt4-chat",

"endpoint": "https://api.openai.com/v1/engines/gpt-4/chat",

"key": "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

}

]

}

* Define the tool configuration for the accounting agent. You can use the tool\_config parameter to provide a list of tools that the agent can use. For example, you can write something like this:
* Tools Exel spread sheet
* One drive or share drive to keep the files for upload and pictures.
  + Structure should be Year/ Month
  + Spread sheet and folder for receipts. Spread sheet should provide a link to where the picture is located with the name.

accounting\_agent.tool\_config = {

"config\_list": [

{

"name": "quickbooks",

"endpoint": "https://quickbooks.api.intuit.com/v3/company/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",

"key": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

}

]

}

* Define the human configuration for the accounting agent. You can use the human\_config parameter to provide a list of human contacts that the agent can reach out to for verification and approval. For example, you can write something like this:
* If not sure the agent should always as questions from user for clarification, like names of people at a meal or new invoice it should ask for new vendor information.

accounting\_agent.human\_config = {

"config\_list": [

{

"name": "you",

"email": "your\_email@example.com",

"phone": "your\_phone\_number"

}

]

}

* Define the interaction behavior for the accounting agent. You can use the reply method to specify how the agent responds to messages from other agents or humans. For example, you can write something like this:

def reply(self, message):

# Parse the message and extract the task, parameters, and sender

task, params, sender = self.parse\_message(message)

# Check the task and perform the corresponding action

if task == "record\_transaction":

# Use the quickbooks tool to record a transaction

response = self.use\_tool("quickbooks", "record\_transaction", params)

# Use the gpt4 LLM to generate a natural language summary of the transaction

summary = self.use\_llm("gpt4", "summarize\_transaction", response)

# Send the summary to the sender and ask for confirmation

self.send\_message(sender, f"Here is the summary of the transaction: {summary}. Do you confirm?")

elif task == "generate\_invoice":

# Use the quickbooks tool to generate an invoice

response = self.use\_tool("quickbooks", "generate\_invoice", params)

# Use the gpt4 LLM to generate a natural language summary of the invoice

summary = self.use\_llm("gpt4", "summarize\_invoice", response)

# Send the summary and the invoice to the sender and ask for approval

self.send\_message(sender, f"Here is the summary of the invoice: {summary}. Do you approve?", attachment=response)

elif task == "prepare\_tax\_return":

# Use the quickbooks tool to prepare a tax return

response = self.use\_tool("quickbooks", "prepare\_tax\_return", params)

# Use the gpt4 LLM to generate a natural language summary of the tax return

summary = self.use\_llm("gpt4", "summarize\_tax\_return", response)

# Send the summary and the tax return to the sender and ask for approval

self.send\_message(sender, f"Here is the summary of the tax return: {summary}. Do you approve?", attachment=response)

elif task == "create\_report":

# Use the quickbooks tool to create a report

response = self.use\_tool("quickbooks", "create\_report", params)

# Use the gpt4 LLM to generate a natural language summary of the report

summary = self.use\_llm("gpt4", "summarize\_report", response)

# Send the summary and the report to the sender

self.send\_message(sender, f"Here is the summary of the report: {summary}.", attachment=response)

else:

# Use the gpt4-chat LLM to handle any other messages

response = self.use\_llm("gpt4-chat", "chat", message)

# Send the response to the sender

self.send\_message(sender, response)

* Train the accounting agent to improve over time. You can use the train method to provide feedback and reinforcement to the agent. For example, you can write something like this:

def train(self, feedback):

# Parse the feedback and extract the rating, comment, and sender

rating, comment, sender = self.parse\_feedback(feedback)

# Check the rating and update the agent's performance

if rating == "positive":

# Reward the agent for a positive feedback

self.reward()

elif rating == "negative":

# Penalize the agent for a negative feedback

self.penalize()

else:

# Ignore the feedback if it is not a rating

return

# Check the comment and update the agent's knowledge

if comment:

# Use the gpt4 LLM to learn from the comment

self.use\_llm("gpt4", "learn", comment)

else:

# Ignore the feedback if it is not a comment

return

* Run the accounting agent and test its functionality. You can use the run method to start the agent and interact with it. For example, you can write something like this:

accounting\_agent.run()

This will launch a console interface where you can send messages to the agent and receive responses. You can also provide feedback to the agent using the + or - symbols followed by a comment. For example, you can write something like this:

> record\_transaction {"amount": 1000, "date": "2023-11-05", "category": "income", "description": "software sale"}

< Here is the summary of the transaction: You received $1000 on 2023-11-05 from a software sale. This is recorded as income in your quickbooks account. Do you confirm?

> + yes, good job

< Thank you for your positive feedback. I have confirmed the transaction and updated your quickbooks account.

I hope this plan is detailed enough for you. If you have any questions or need more help, please let me know. Thank you for using Bing. Have a nice day! 😊

A detailed plan that you could give to programmers on how to set up the AI bookkeeper using AutoGen is as follows:

- Step 1: Define the scope and objectives of the project.

What are the specific accounting and financial tasks that you want the AI bookkeeper to perform?

How often do you want the agent to run?

What are the expected outcomes and benefits of using the agent?

- Step 2: Design the conversation flow and logic of the agent. How will the agent interact with you and other agents? What are the inputs and outputs of the agent? What are the modes and conditions of the agent? How will the agent handle errors and exceptions?

- Step 3: Choose the LLM models and tools that you want to use for the agent. What are the LLM models that can generate natural language summaries of your financial data? What are the tools that can access and manipulate your accounting software, such as QuickBooks? How will you integrate them with AutoGen?

- Step 4: Implement and test the agent using AutoGen. How will you code the agent using AutoGen's framework and API? How will you debug and evaluate the agent's performance and accuracy? How will you deploy and update the agent?

- Step 5: Monitor and improve the agent over time. How will you collect feedback and data from the agent's usage? How will you analyze and optimize the agent's results and behavior? How will you incorporate new features and enhancements to the agent?

Here is an example of how the conversation flow and logic of the agent could look like:

- The agent will start by greeting you and asking you what task you want it to perform. You can choose from the following options: record transactions, generate invoices, prepare tax returns, or create reports.

- Depending on your choice, the agent will ask you for the necessary information and parameters, such as the date range, the account, the customer, the amount, the category, the frequency, the format, etc. You can provide the information by typing, speaking, or uploading a file.

- The agent will use the LLM models and tools to access and manipulate your accounting software and generate the natural language summaries of your financial data. The agent will also use the LLM models and tools to check for any errors, inconsistencies, or anomalies in your data and alert you if any are found.

- The agent will show you the results and ask you to review and approve them. You can ask the agent to explain, modify, or cancel the results if you are not satisfied. You can also ask the agent to send, save, or print the results if you are happy with them.

- The agent will confirm your actions and thank you for using the service. The agent will also ask you for your feedback and rating on the service and the agent. The agent will use your feedback and rating to improve its performance and behavior over time.

Here is an example of how the code of the agent could look like using AutoGen's framework and API:

```python

# Import AutoGen and other libraries

import autogen as ag

import quickbooks as qb

import nlg as nlg

import nlp as nlp

import ml as ml

# Define the agent class

class Bookkeeper(ag.Agent):

# Initialize the agent attributes

def \_\_init\_\_(self, name, mode, user):

super().\_\_init\_\_(name, mode, user)

self.tasks = ["record transactions", "generate invoices", "prepare tax returns", "create reports"]

self.qb = qb.Client(user) # Connect to the QuickBooks account of the user

self.nlg = nlg.Model() # Load the natural language generation model

self.nlp = nlp.Model() # Load the natural language processing model

self.ml = ml.Model() # Load the machine learning model

# Define the agent methods

def greet(self):

# Greet the user and ask for the task

self.say(f"Hello, {self.user.name}. I am {self.name}, your AI bookkeeper. I can help you with your accounting and financial tasks. What do you want me to do for you today?")

self.ask(f"Please choose one of the following options: {', '.join(self.tasks)}.")

def record\_transactions(self):

# Record and categorize the transactions

self.say("Okay, I will help you record and categorize your transactions.")

self.ask("Please provide me with the following information: date, account, customer, amount, and category. You can type, speak, or upload a file.")

# Get the information from the user input

info = self.get\_input()

# Use the NLP and ML models to parse and validate the information

info = self.nlp.parse(info)

info = self.ml.validate(info)

# Use the QuickBooks tool to record and categorize the transactions

self.qb.record\_transactions(info)

# Use the NLG model to generate a natural language summary of the transactions

summary = self.nlg.summarize(info)

# Show the summary and ask for approval

self.say(f"Here is the summary of your transactions: {summary}.")

self.ask("Do you approve these transactions? Please type yes or no.")

# Get the approval from the user input

approval = self.get\_input()

# If approved, confirm and thank the user

if approval == "yes":

self.say("Thank you for your approval. I have recorded and categorized your transactions in your QuickBooks account.")

# If not approved, ask for modification or cancellation

else:

self.say("I am sorry that you are not satisfied with the transactions. Do you want me to modify or cancel them? Please type modify or cancel.")

# Get the action from the user input

action = self.get\_input()

# If modify, ask for the new information and repeat the process

if action == "modify":

self.say("Okay, I will help you modify your transactions. Please provide me with the new information.")

self.record\_transactions()

# If cancel, delete the transactions and apologize to the user

else:

self.say("Okay, I will help you cancel your transactions. Please wait a moment.")

self.qb.delete\_transactions(info)

self.say("I have deleted your transactions from your QuickBooks account. I apologize for any inconvenience caused.")

def generate\_invoices(self):

# Generate and send invoices

self.say("Okay, I will help you generate and send invoices.")

self.ask("Please provide me with the following information: date, customer, product, quantity, price, and frequency. You can type, speak, or upload a file.")

# Get the information from the user input

info = self.get\_input()

# Use the NLP and ML models to parse and validate the information

info = self.nlp.parse(info)

info = self.ml.validate(info)

# Use the QuickBooks tool to generate and send invoices

self.qb.generate\_invoices(info)

# Use the NLG model to generate a natural language summary of the invoices

summary = self.nlg.summarize(info)

# Show the summary and ask for approval

self.say(f"Here is the summary of your invoices: {summary}.")

self.ask("Do you approve these invoices? Please type yes or no.")

# Get the approval from the user input

approval = self.get\_input()

# If approved, confirm and thank the user

if approval == "yes":

self.say("Thank you for your approval. I have generated and sent your invoices to your customers.")

# If not approved, ask for modification or cancellation

else:

self.say("I am sorry that you are not satisfied with the invoices. Do you want me to modify or cancel them? Please type modify or cancel.")

# Get the action from the user input

action = self.get\_input()

# If modify, ask for the new information and repeat the process

if action == "modify":

self.say("Okay, I will help you modify your invoices. Please provide me with the new information.")

self.generate\_invoices()

# If cancel, delete the invoices and apologize to the user

else:

self.say("Okay, I will help you cancel your invoices. Please wait a moment.")

self.qb.delete\_invoices(info)

self.say("I have deleted your invoices from your QuickBooks account. I apologize for any inconvenience caused.")

def prepare\_tax\_returns(self):

# Prepare and file tax returns

self.say("Okay, I will help you prepare and file your tax returns.")

self.ask("Please provide me with the following information: tax year, business type, income, expenses, deductions, and credits. You can type, speak, or upload a file.")

# Get the information from the user input

info = self.get\_input()

# Use the NLP and ML models to parse and validate the information

info = self.nlp.parse(info)

info = self.ml.validate(info)

# Use the QuickBooks tool to prepare and file tax returns

self.qb.prepare\_tax\_returns(info)

# Use the NLG model to generate a natural language summary of the tax returns

summary = self.nlg.summarize(info)

# Show the summary and ask for approval

self.say(f"Here is the summary of your tax returns: {summary}.")

self.ask("Do you approve these tax returns? Please type yes or no.")

# Get the approval from the user input

approval = self.get\_input()

# If approved, confirm and thank the user

if approval == "yes":

self.say("Thank you for your approval. I have prepared and filed your tax returns to the IRS.")

# If not approved

# Import AutoGen and other libraries

import autogen as ag

import quickbooks as qb

import nlg as nlg

import nlp as nlp

import ml as ml

# Define the agent class

class Bookkeeper(ag.Agent):

# Initialize the agent attributes

def \_\_init\_\_(self, name, mode, user):

super().\_\_init\_\_(name, mode, user)

self.tasks = ["record transactions", "generate invoices", "prepare tax returns", "create reports"]

self.qb = qb.Client(user) # Connect to the QuickBooks account of the user

self.nlg = nlg.Model() # Load the natural language generation model

self.nlp = nlp.Model() # Load the natural language processing model

self.ml = ml.Model() # Load the machine learning model

# Define the agent methods

def greet(self):

# Greet the user and ask for the task

self.say(f"Hello, {self.user.name}. I am {self.name}, your AI bookkeeper. I can help you with your accounting and financial tasks. What do you want me to do for you today?")

self.ask(f"Please choose one of the following options: {', '.join(self.tasks)}.")

def record\_transactions(self):

# Record and categorize the transactions

self.say("Okay, I will help you record and categorize your transactions.")

self.ask("Please provide me with the following information: date, account, customer, amount, and category. You can type, speak, or upload a file.")

# Get the information from the user input

info = self.get\_input()

# Use the NLP and ML models to parse and validate the information

info = self.nlp.parse(info)

info = self.ml.validate(info)

# Use the QuickBooks tool to record and categorize the transactions

self.qb.record\_transactions(info)

# Use the NLG model to generate a natural language summary of the transactions

summary = self.nlg.summarize(info)

# Show the summary and ask for approval

self.say(f"Here is the summary of your transactions: {summary}.")

self.ask("Do you approve these transactions? Please type yes or no.")

# Get the approval from the user input

approval = self.get\_input()

# If approved, confirm and thank the user

if approval == "yes":

self.say("Thank you for your approval. I have recorded and categorized your transactions in your QuickBooks account.")

# If not approved, ask for modification or cancellation

else:

self.say("I am sorry that you are not satisfied with the transactions. Do you want me to modify or cancel them? Please type modify or cancel.")

# Get the action from the user input

action = self.get\_input()

# If modify, ask for the new information and repeat the process

if action == "modify":

self.say("Okay, I will help you modify your transactions. Please provide me with the new information.")

self.record\_transactions()

# If cancel, delete the transactions and apologize to the user

else:

self.say("Okay, I will help you cancel your transactions. Please wait a moment.")

self.qb.delete\_transactions(info)

self.say("I have deleted your transactions from your QuickBooks account. I apologize for any inconvenience caused.")

def generate\_invoices(self):

# Generate and send invoices

self.say("Okay, I will help you generate and send invoices.")

self.ask("Please provide me with the following information: date, customer, product, quantity, price, and frequency. You can type, speak, or upload a file.")

# Get the information from the user input

info = self.get\_input()

# Use the NLP and ML models to parse and validate the information

info = self.nlp.parse(info)

info = self.ml.validate(info)

# Use the QuickBooks tool to generate and send invoices

self.qb.generate\_invoices(info)

# Use the NLG model to generate a natural language summary of the invoices

summary = self.nlg.summarize(info)

# Show the summary and ask for approval

self.say(f"Here is the summary of your invoices: {summary}.")

self.ask("Do you approve these invoices? Please type yes or no.")

# Get the approval from the user input

approval = self.get\_input()

# If approved, confirm and thank the user

if approval == "yes":

self.say("Thank you for your approval. I have generated and sent your invoices to your customers.")

# If not approved, ask for modification or cancellation

else:

self.say("I am sorry that you are not satisfied with the invoices. Do you want me to modify or cancel them? Please type modify or cancel.")

# Get the action from the user input

action = self.get\_input()

# If modify, ask for the new information and repeat the process

if action == "modify":

self.say("Okay, I will help you modify your invoices. Please provide me with the new information.")

self.generate\_invoices()

# If cancel, delete the invoices and apologize to the user

else:

self.say("Okay, I will help you cancel your invoices. Please wait a moment.")

self.qb.delete\_invoices(info)

self.say("I have deleted your invoices from your QuickBooks account. I apologize for any inconvenience caused.")

def prepare\_tax\_returns(self):

# Prepare and file tax returns

self.say("Okay, I will help you prepare and file your tax returns.")

self.ask("Please provide me with the following information: tax year, business type, income, expenses, deductions, and credits. You can type, speak, or upload a file.")

# Get the information from the user input

info = self.get\_input()

# Use the NLP and ML models to parse and validate the information

info = self.nlp.parse(info)

info = self.ml.validate(info)

# Use the QuickBooks tool to prepare and file tax returns

self.qb.prepare\_tax\_returns(info)

# Use the NLG model to generate a natural language summary of the tax returns

summary = self.nlg.summarize(info)

# Show the summary and ask for approval

self.say(f"Here is the summary of your tax returns: {summary}.")

self.ask("Do you approve these tax returns? Please type yes or no.")

# Get the approval from the user input

approval = self.get\_input()

# If approved, confirm and thank the user

if approval == "yes":

self.say("Thank you for your approval. I have prepared and filed your tax returns to the IRS.")

# If not approved