به نام خدا

ریحانه منتظری

پروژه ی اضافه برای درس کامپایلر(چت بات)

در این پروژه قصد داریم یک سیستم پاسخگویی به سوالات با کمک هوش مصنوعی chat gpt ساخته که قادر به تشخیص سوالات و تبدیل آنها به متن (text to speech) و همچنین نگهداری روند سوالات و پاسخگویی به آنها می باشد.

در ابتدا کتابخانه speech_recognition را ایمپورت می‌کنیم و به آن نام اختصاری sr را می دهیم.این کتابخانه اجازه می‌دهد که تشخیص گفتار را روی فایل‌های صوتی یا ورودی میکروفون انجام دهیم. این کتابخانه از چندین سرویس تشخیص گفتار از جملهGoogle Speech Recognition ، Microsoft Bing Voice Recognition، IBM Speech to Textو CMU Sphinx پشتیبانی می‌کند. در این کد، از سرویس Google Speech Recognition که ابزاری رایگان است استفاده می‌شود.

کتابخانه pyttsx3 را ایمپورت میکنیم. این کتابخانه یک موتور تبدیل متن به گفتار برای پایتون است که از چندین موتور تبدیل متن به گفتار از جمله nsss،SAPI5 و espeak پشتیبانی می‌کند.

کتابخانه openai را ایمپورت می‌کنیم. این کتابخانه یک پوشش برای API OpenAI است که دسترسی به مدل‌های قدرتمند برای پردازش زبان طبیعی مانند GPT-4 وGPT-3.5 را فراهم می کند. در این کد، از مدل text-davinci-003 که یک مدل تولید متن بر اساس GPT-3 است استفاده می‌شود.

یک نمونه از موتور pyttsx3 را برای تبدیل متن به گفتار ایجاد می‌کنیم و آن را به متغیر engine اختصاص می‌دهیم.

لیست صداهای موجود برای موتور pyttsx3 دریافت شده و به متغیر voices برای انتخاب صدای مورد نظر برای تبدیل متن به گفتار اختصاص داده می شود. این متغیر یک لیست از شیءهای صدا است که هر کدام شناسه، نام و زبان را دارند.

ویژگی صدا را برای موتور به صدای دوم در لیست (شاخص ۱) تنظیم می‌کنیم. با انتخاب شاخص های دیگر می توان صداهای دیگر که از ورودی میکروفون دریافت شده اند را انتخاب کرد.(برای مثال برای حذف صداهای بک گراند و غیره کافیست شاخص مورد نظر را انتخاب کرده تا نویزهای بک گراند به عنوان ورودی به برنامه داده شود.)

یک نمونه از کلاس sr.Recognizer را ایجاد کرده و آن را به متغیر r اختصاص می‌دهیم. این متغیر برای تشخیص گفتار از منابع صوتی استفاده می‌شود و متدهای مختلفی برای تشخیص گفتار از فایل های صوتی و یا میکروفون دارد.

یک نمونه از کلاس sr.Microphone را ایجاد و آن را به متغیر mic اختصاص می‌دهیم. در واقع در اینجا میکروفون کاربر به عنوان ورودی از دستگاه وی انتخاب شده است.

یک رشته خالی برای conversation ایجاد می کنیم تا پیام‌های تبادل شده بین کاربر و دستیار را ذخیره کند. به عبارتی وظیفه ی این رشته ارسال و دریافت پیام‌ها به مدل openai است تا پیام های پیشین کاربر را به عنوان دیتای اضافه برای تولید جواب مرتبط تر در اختیار مدل ai قرار دهیم. این متغیر در ابتدا خالی است و در هر بار گفتگو با پیام جدید از طرف کاربر و بعد با پاسخ از طرف دستیار به‌روز می‌شود.

سه متغیر رشته‌ای که نام کاربر، نام دستیار و مکالمه را ذخیره می‌کنند، تعریف می کنیم. این متغیرها برای نمایش و ارسال پیام‌ها به مدل openai استفاده می‌شوند.

یک حلقه بی‌نهایت تعریف می کنیم که تا زمانی که برنامه متوقف نشود ادامه می‌یابد. این حلقه برای ایجاد یک گفتگو پیوسته بین کاربر و دستیار است و در هر دور از حلقه، یک پیام از کاربر گرفته می‌شود و یک پیام از دستیار تولید می‌شود.

در ابتدای حلقه با استفاده از میکروفون به گفتار کاربر گوش داده و گفتار به متن تبدیل میشود. این بخش کد با استفاده از کتابخانه speech_recognition ورودی صوتی کاربر را تشخیص می‌دهد.وقتی ارتباط با میکروفون برقرار شد، یک پیام ”Listening…“ را در کنسول چاپ می‌کنیم که نشان می‌دهد برنامه آماده گوش دادن است. سپس آستانه انرژی تشخیص‌گر بر اساس سطح نویز محیطی منبع تنظیم می شود که کمک می‌کند تا نویز پس‌زمینه را فیلتر کرده و دقت بالاتر برود. سپس برنامه به منبع که همان میکروفون کاربر است برای دریافت داده های صوتی مراجعه میکند. جمع آوری داده های صوتی تا زمانی که سکوت از طرف ورودی صدا دریافت شود ادامه می یابد. سپس یک پیام ”no longer listening“ را در کنسول چاپ می‌کنیم که نشان میدهد برنامه دیگر به گفتار کاربر گوش نمی‌دهد. حالا داده های صوتی دریافتی را با استفاده از سرویس تشخیص گفتار گوگل شناسایی کرده و متن تبدیل شده رابه متغیر user_input اختصاص دهیم. اگر هنگام تشخیص خطایی رخ دهد، کد با دستور continue به حلقه برمی‌گردد و ورودی جدیدی را گوش می‌دهد.

در مرحله بعد متن ورودی کاربر را به عنوان یک پیام از نقش user به مدل openai می‌فرستیم. جواب این بخش کد که با استفاده از کتابخانه openai پاسخ دستیار را تولید می‌کند در متغیر assistant ذخیره می شود. ابتدا یک رشته ایجاد می‌کنیم که شامل نام کاربر، متن ورودی کاربر، یک خط جدید و نام دستیار است که با دو نقطه از هم جدا شده‌اند. این رشته را به متغیر prompt اختصاص می‌دهیم. این متغیر برای ارسال به مدل openai می‌شود. سپس این متغیر را به انتهای متغیر conversation اضافه می‌کنیم که نشانگر اضافه شدن یک پیام جدید به مکالمه است. سپس یک درخواست تکمیل متن به مدل openai می‌فرستیم و پاسخ دریافت میشود. این درخواست شامل چندین پارامتر است که تأثیر بر روی خروجی مدل دارند. مثلاً، پارامتر model نام مدل را مشخص می‌کند که در اینجا text-davinci-003 است. پارامتر prompt متن ارسالی به مدل هوش مصنوعی را مشخص می‌کند که در اینجا متغیر conversation است. پارامتر temperature میزان تصادفی بودن و خلاقیت متن تولید شده را کنترل می‌کند. این پارامتر یک عدد بین ۰ تا ۱ است که نشان می‌دهد که مدل چقدر از متن ارسالی به open ai فاصله بگیرد. هر چه این عدد بزرگتر باشد، متن تولید شده تصادفی‌تر و خلاقانه‌تر خواهد بود. پارامتر max_tokens حداکثر تعداد توکن‌هایی را که می‌تواند تولید کند نگهداری می کند. . توکن‌ها واحدهای کوچکتری از متن هستند که می‌توانند حروف، کلمات یا علائم نگارشی باشند. هر چه این عدد بزرگتر باشد، متن تولید شده طولانی‌تر خواهد بود. پارامتر top_p میزان احتمال توکن‌هایی را که می‌تواند تولید کند مشخص می‌کند. این پارامتر یک عدد بین ۰ تا ۱ است که نشان می‌دهد که مدل چقدر از توکن‌های کم احتمال فاصله بگیرد. هر چه این عدد کوچکتر باشد، متن تولید شده پیش‌بینی‌پذیرتر و منطقی‌تر خواهد بود. پارامترهای frequency_penalty و presence_penalty جریمه‌هایی را که مدل برای تکرار یا حضور توکن‌ها در متن تولید شده دریافت می‌کند مشخص می‌کنند. این پارامترها عددهای بین ۰ تا ۱ هستند که نشان می‌دهند که مدل چقدر از تکرار یا حضور توکن‌ها کاسته شود. هر چه این عددها بزرگتر باشند، متن تولید شده تنوع بیشتری خواهد داشت. همچنین پاسخی که از مدل دریافت می‌شود شامل چندین فیلد است که اطلاعات مختلفی را درباره متن تولید شده دارد. مثلاً، فیلد choices یک لیست از شیءهای choice است که هر کدام شامل متن تولید شده و اطلاعات دیگری درباره آن هستند. در این کد، فقط اولین شیء choice را در نظر می‌گیریم و متن تولید شده را به متغیر response_str اختصاص می‌دهیم. این متغیر حاوی پیام دستیار به کاربر است.این متغیر را در ادامه در کنسول چاپ می‌کنیم و با استفاده از موتور pyttsx3به صوت تبدیل و پخش می‌کنیم.

متن خروجی مدل را پردازش می‌کنیم و فقط بخشی که مربوط به پیام دستیار است را استخراج می‌کنیم. این بخش کد با استفاده از توابع رشته‌ای پایتون متن را تقسیم، جستجو و جایگزینی می‌کند. ابتدا متن را بر اساس خط های جدید تقسیم می‌کنیم و آخرین عنصر را به متغیر response اختصاص می‌دهیم. این عنصر شامل پیام دستیار و بخشی از متن ورودی است. سپس متن را بر اساس دو نقطه تقسیم می‌کنیم و دومین عنصر را به متغیر response اختصاص می‌دهیم. این عنصر شامل پیام دستیار و یک فاصله است. سپس فاصله را از ابتدای متن حذف می‌کنیم و متن را به متغیر response_str اختصاص می‌دهیم. این متغیر حاوی پیام دستیار به کاربر است. سپس این متن را به کنسول چاپ می‌کنیم و با استفاده از موتور pyttsx3 به صوت تبدیل و پخش می‌کنیم. تمامی عملیات های جایگزینی و حذف با استفاده از متدهای splice و replace انجام می شوند.

متن تولید شده را در کنسول چاپ کرده و با استفاده از موتور pyttsx3 به صوت تبدیل و پخش می‌کنیم. این بخش کد با استفاده از تابع print و متد engine.sayمتن را نمایش و پخش می‌کند. متد engine.say متن را به موتور pyttsx3 می‌فرستد که آن را به صوت تبدیل و پخش می‌کند. این متد فقط متن را به playlist اضافه می‌کند و بلافاصله برمی‌گردد. برای اجرای playlist و پخش صوت، باید متد engine.runAndWait را فراخوانی کنیم که در جلوتر انجام می‌شود.

متد engine.runAndWait را فراخوانی می‌کنیم که play list را اجرا و صوت را پخش می‌کند. این متد تا زمانی که تمام متن‌های playlist پخش شوند، منتظر می‌ماند و در واقع برای اطمینان از پخش کامل صوت قبل از ادامه برنامه از آن استفاده کردیم.

درنهایت متن تولید شده را به انتهای متغیر conversation اضافه می‌کنیم که نشان می‌دهد یک پیام جدید به مکالمه اضافه شده است. تا زمانی که کاربر همچنان ارتباط خودش با دستیار را حفظ کند، این حلقه انجام شده و با خروج کاربر از برنامه، تمام داده های فعلی که در conversation ذخیره شده بود پاک می شود.

با تشکر از حسن توجه شما

ریحانه منتظری