EVA: An Experiment in Introducing AI Assistants in Linux Operating Systems

Obanadab E. Archibong

Lincoln University

ob.archibong455@my.lincolnu.edu

Abstract

Generative technologies are becoming more commonplace in society, and as a result, the bar of entry for interfacing with such has fallen drastically. Software applications have become more generalized and accessible, with the latest features being the implementation of artificial intelligence (AI). However, desktop digital assistants in mainstream operating systems today are predominately neglected, even by their targeted audience of in-experienced users who often overlook such assistive technologies.

This project serves as an experiment in implementing a desktop digital assistive tool for Linux, which in application highlights AI's strengths in supporting, rather than interjecting the user experience. In developing the system multiple dependent packages will be used to grant the system the desired functionality. In recent years software suites like Windows have notoriously been criticized for including assistants due to the operating systems' already simplified interfaces. In contrast, Linux, a popular open-source operating system, is often overlooked due to being less user-friendly. Implementing assistive features for command-line interfaces could serve to bridge the gap in terms of user interpretation of the Linux platform.

Keywords

AI, AI Desktop Assistant, LLM, Voice recognition, Ollama, Linux.

Introduction

Elevated Verbose Agent (EVA) is a personal project aimed at developing an AI desktop assistant for Linux platforms. Serving as a helpful on-hand resource and companion with the ability to execute rudimentary system functions. Tools similar to EVA already exist; one popular tool Co-Pilot closely achieves the desired goals of this project. Common today are Amazon Alexa or Google’s Gemini. On most modern devices some form of assistive program is one click or speech phrase away. Such companions are useful; they can organize schedules, prioritize tasks, and provide timely updates on public information broadcasts. However, most assistants are limited in actions aside from just searching the web and launching local applications.

The primary goal in developing EVA was to create an open-source, personalized desktop assistant utility supported by Linux/Unix platforms. The DIY culture of the Linux/Unix space does not advocate for such progressive accessibility features. Ideally, the tool will be able to familiarize new users with the command line. Exchanges involving guided interactive tutorials which the assisting agent could better accumulate new users. From suggesting software packages to correcting malformed commands, EVA should encourage new users to frequent the terminal interface. The system Lowers the barrier of entry, as Instead of spending time consulting forums,

EVA can assist in early navigation of the OS. Extensive documentation allowing more seasoned power users to create and develop modded extensions for their unique use cases.

In summary, EVA's functionality encapsulates elements of chatbots (companion applications), and digital assistants. In its default state, the system's vernacular is heavily inspired by modern-day science fictional caricatures like Iron Man's JARVIS, or the Portal Series Series G.L.A.D.O.S. However configuration behavior can be modified to fit the user's preferences. Not just voice options, but personality traits, dialect, and custom caricatures. Unlike common desktop digital assistants requests never leave the host machine assuring that interactions are private. EVA could be open source, in favor of being maintained by any corporate entity. As questionable practices involving user agreement policy have led to user privacy becoming a growing concern. In developing EVA we've established the following goals:

Important Criteria:

1. The system should respond accurately to user prompts.
2. The system should prioritize system confidentiality and integrity of user data.
3. The system should be accessible to new users.
4. The system should be flexible, adapting to varied scenarios.

These criteria stand as the guidelines that lead project decision-making. Criteria one establishes that at its core the system must maintain availability, timeliness, and accuracy in its deployment. The second and third stating elements of confidentiality, and security to maintain the integrity of devices. The final criteria refers to the overall utility of the system as a whole, posing the question; Is this system practical?

Main System Components

Workstation

The workstation for this project is a Steamdeck OLED; a portable handheld device running SteamOS 3, an arch-like operating system recently released by Valve. "Specs include an 8-core Sephiroth processor with 16GB memory released in November 9th 2023, as reported on techpowerup.com / steamdeck.com (see, AMD).

Distrobox

Distrobox is a containerization utility that allows for multiple separate distributions of Linux to be run in a terminal. Security features are built into the creation of containers, separating host file systems and processes from chroot environments. Linux by default is a highly integral Operating system, though further steps will be taken to secure the environment. The sandbox environment within which all operations will be conducted is the latest version of Ubuntu, Noble Numbat.

Ollama

Ollama a popular open-source repository of LLM's is the key cornerstone of this system. Enabling us to create and models offline naively by installing their API. Ollama offers a privatized seamless experience without the need for outbound request. However, a model

in its default state is not conducive to the project goals, in order to achieve proper replies a custom model must be derived from it.

Customizing a model that replies in properly formatted commands requires loading a model file.

Implementation

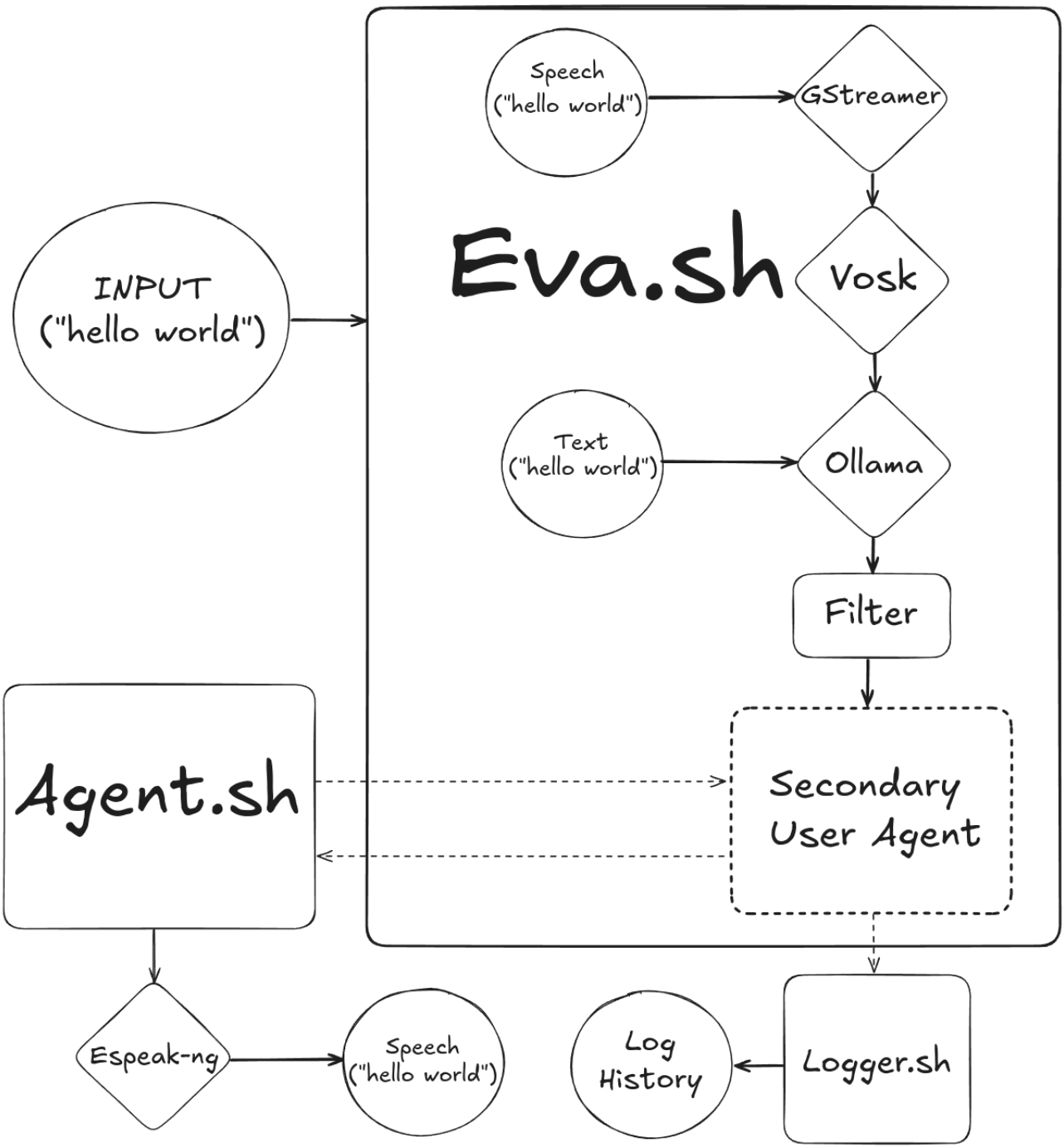
The main sections of a model file include the ‘FROM’, ‘TEMPLATE’, ‘SYSTEM’, ‘PARAMETER’, and ‘MESSAGE’ tags. ‘FROM’ references the paths to binary large object files or BLOB which contain all model data. ‘TEMPLATE’ is defined by the original model's data retrieval structures, and for this project, they will remain unmodified. ‘SYSTEM’ defines the model's behavior; which will include information on available commands and the structure of output. Finally, variables like ‘PARAMETER’ further adjust model behavior by modifying variables such as temperature. Temperature controls the level of creativity and flexibility when generating responses, higher values decreasing accuracy and vice versa. Lastly, ‘MESSAGE’ acts as a role indicator to inject artificial chat logs to model history. The model defined as 'assistant' and the user as 'user'.

Developing a desktop digital assistant on Linux at first is not very practical, because Linux’s distributions (Distros) do not come prepackaged with a desktop. So the design of EVA is strictly implemented with a non-graphical user interface (GUI) methodology in mind. The inherit design of EVA is to be a wrapper for Ollama, a command line. Acting as an extension to receive well-formed user input from a variety of sources. The design was inspired by Fabric a popular AI framework, developed by Daniel Miessler that emphasizes prompt engineering (see, Miessler 5). Project EVA is composed of multiple languages, primarily Bash and Python. Shell scripts control the overall coordination of external software packages. Python's primary purpose is within the transcription features of the application.

The schema of the system can be broken down into three main components, Eva.sh, Actor.sh, and Logger.sh. Eva.sh acts as the main controller, accepting user input, and executing low privilege commands under the secondary user within a restricted shell environment. When the command 'eva' is run, it makes logical decisions on what actions to take with the arguments it is provided. Actor.sh executes filtered high privilege scripted commands requested by Eva.sh. Lastly, Logger.sh collects all command output, timestamps systems actions, then modifies any output logs.

Unlike humans, to accept new or recall past information, the model must be re-calibrated with the necessary data. The model file is modified by Logger.sh with the appropriate roles utilizing the 'MESSAGE' and role tags to specify who said what. Mistakes happen, in the event of a syntactical error output Logger.sh re-encapsulated and then truncated to the model file to be redressed later. Actor.sh acts as an authoritative figure, handling appeals that involve lower-level system actions. Accepting parameters for scripted actions declared in the secondary user's bin directory. Actor.sh utilities an allow list denying any command or argument outside of its dictionary.

To envision the data flow of the prior mentioned interaction in more detail, here is a logistical diagram:


Speech
("hello world")
GStreamer
Eva.sh
Vosk
INPUT
("hello world")
Text
("hello world")
Ollama
Filter
Agent.sh
Secondary
User Agent
Espeak-ng
Speech
("hello world")
Log
History
Logger.sh

A prompt, composed of user input is accepted through many pathways including; piping, passed as arguments, or transcription. The transcription path differs in that prompts are spoken aloud and recorded as sound bites with Gstreamer. Then transcribed with Vosk-Transcriber, an audio transcription software, before being interpreted by models.

The system then passes the text prompt to a local model that interprets the query, and generates commands. Commands are filtered and formatted to then be executed within a sandboxed environment by a secondary user-restricted shell. Commands vary, but all serve to orchestrate unique and appropriate system action; from speaking aloud, to file steam operations. File permissions allow one-way modification of files from the host to a less-privileged secondary user account to support integrity.

On the note of execution, such ambitious actions pose significant risks. To mitigate or mediate such susceptibilities; we must consider a schema to adopt from a security perspective. To be blunt, allowing a less than reliably morally artificial intelligence to have the reigns in command execution is not appropriate. Separating the model from a direct means of execution is crucial. To illustrate, imagine an uneducated toddler at the wheel of a stationary vehicle. Even if such an actor is not readily able to enact any tangible action, the situation still poses a significant risk in its potentiality. Action.sh, an external script, acts as a lawful parental agent meditating situations like these. Whatever buttons or pedals the AI wishes to interact with are limited by Action.sh, the model fulfilling the role of passenger. Only certain higher privileged commands in its dictation are permissible, while parameters are given more leniency depending on context. Middle grounds can be met with conservative actions, for instance, playing music files passed as parameters are respected. Though the direct utility of system devices is strictly forbidden, but requests for scripted actions are gladly fulfilled. Low privilege commands like ‘ls’, ‘date’, or ‘whoami’ not being moderated as intently. However, Like improper manners any malformed commands, are receded and accounted for.

EVA operates within a restricted environment but needs flexible constrain when it comes to file access permissions. Linux default built-in Access control list (ACL) in this project is supplemented with the package Acl which includes a robust system of file permission management for specific directories. To keep the configuration brief, the basic requirements are to implement an allow-list in favor of a blacklist. The goal is to allow the secondary user access to certain files within the sandbox. This applies to the home directory as well with more leniency given in directory and file modification outside of configuration files and binary executables. Such as disallowing the modification of binaries within ~/bin to the user account 'eva'. Also enabling the accessibility of the primary user to access the secondary user's home directory.

Further modifications can be applied as necessary, such as groups to create shared directories that allow for access among both the EVA and primary user accounts. Where Logger.sh shines in is acting as a reliable auditor/accountant of all system activity for later reference. Before execution commands are properly sanitized, escaped, and then executed by either Eva.sh or Agent.sh. In the process, Logger.sh separates the output by file stream forwarding output to their respective log files. Such as feeding an error message or concatenation of a read file back to the model file.

Multi-step operations with direct references to the environment are beyond the systems current capabilities. Due to that the model’s knowledge base being a separate entity. So actions must be tackled in subsequent order, building on environmental knowledge gathered from previous command output.

Here is an example of a use case of the system via a command line that requests a summary of a file:

```
user@localhost $~ eva --talk
record: [ENTER]
recording...
stop: [ENTER]
stopping...
eva@localhost $~ espeak-ng "It's currently $(date +%-I:%M%P).";
```

In this short snip-it the model received a prompt “What time is it?” vocally in interactive talk mode. The model itself is not aware of the time, so intelligently substitutes the command ‘date’ with proper format arguments within the command espeak-ng. The higher privilege command is then requested to be executed by Agent.sh, which gladly accepts the argument body text. Finally, the response is spoken aloud to the user. Now that the rudimentary system is operational a few more optional packages give EVA its intended functionality.

Additional Components

Tmux

Tmux, a terminal multiplexer, allows for the management of terminal sessions. Creating, splitting, attaching, and detaching windows/sessions for multi-user systems is the predominant feature used in his project. The secondary user will have its commands executed in this view, conveniently enabling live output to be interpreted by the user verbosely. EVA is deeply intertwined with Tmux, it being a primary dependency. All actions taken by EVA are projected ti uniquely named EVA session. EVA's shell is restricted the path within a Tmux session, the command 'eva' being disabled outside of a valid Tmux session.

Espeak-ng

Espeak-ng is a command line Text-To-Speach software package; it accepts textual input and converts to mnemonic voice modulations that mimic human speech (see, MBROLA). Espeak-ng has repositories containing its collection of Mbrola voices, also allowing for further customization of pitch, volume, and dictation speed through declarations. This project utilizes the library file us-mbrola-1 with unique specifiable arguments. All to give the assistant a voice, though a purely synthetic one.

GStreamer

Gstreamer is a command audio recording software that supports a variety of audio formats and possesses key features to support the system (see, Gstreamer). Multiple specifications on audio devices, sampling rates, and file types can be given in a single command. The primary format used by EVA is

single channel wave, or WAV. Audio prompts are recorded in short 10-30 second clips to be transcribed.

Vosk-transcriber

Vosk transcriber is a voice transcription software written in Python. Vosk transcriber is open source, maintaining a large repertoire of models for specific use cases. It utilizes a local speech recognition models to transcribe human speech. This tool is used to transcribe WAV files reorder by GStreamer to text; a suitable input for the model (see, VOSK).

The command ‘eva’ can only be executed within a active Tmux session, due to default scripted actions being dependent on the functions only possible within a Tmux environment. Here is an example to ask Eva to open w3m, a command-line web browser that utilizes Ncurses as its primary interface:

```
user@localhost $~ echo "Can you open the web browser?" | eva
eva@localhost $~ w3m; # Default script request browser access
[ ^ system tmux new-window "w3m -6 -no-cookie www.duckduckgo.com";]
user@localhost $~ eva "thank you";
eva@localhost $~ espeak-ng "You're welcome, happy browsing!";
```

In this example the user requests for the browser to be opened by echoing a prompt to eva. In response ‘w3m’ is executed by user eva, which is a wrapper function that requests the following system command to Agent.sh. Agent.sh searches its on-hand dictionary and launches the command-line application in a new window.

Closing statements

Shortcomings

Though still in development the EVA is not without inherent design flaws. For example, its hard-coded scripting elements reduce the system's configuration capacity, enabling a more modular design that could remedy this. Custom user-defined scripts could be defined in configuration directories and generate wrapper functions that pass the requested command parameters to the listening Agent.sh. For instance, scripts could be accessible and modifiable by the host within the restricted bash users bin directory. Flexible environment variables defined within model files could auto-populate available commands to the model's dictionary. Secondly, model performance lacks consistency as Logger.sh truncates chat logs to Model-files load times exponentially increase. Response times arrive within 20-30 seconds,

while after several consecutive interactions times peak at 90 sounds or 1.5 minutes. Available resources are the noticeable limitation affecting both local models. Lastly, hands-free speech recognition with the current standards is subpar. Recording short prompts by invoking the command ‘eva –talk’ in interactive mode is recommended over server mode. Currently, the System lacks significant accuracy in detecting speech phrases in live session mode. More algorithmic efficiency can only be obtained by improving methods of detecting human speech from background noise. All so that the assistant can detect and comprehend hands-free commands without negatively impacting performance.

Tuning

The next step would be to improve the overall accuracy of the Models via a process known as tuning. Models created by large-scale cooperation like Meta and Open-AI are general purpose Models. Tuning improves a model by retraining the model for a specific task and pruning unnecessary information in the process. The resulting trained model is reduced in size; more focused, accurate, and reliably more efficient at its intended purpose. Currently, the system relies upon two models. Llama3.2:2B; a two-gigabyte general model developed by Meta. The other; is Vosk transcribers small-en-us-15, a lightweight 40-megabyte voice transcription model. Tuning would not only reduce the size of the models decreasing system resource strain. But also make replies more accurate to the specified system, for example, the model could be trained on 500 or so temples of responses to have a deeper understanding of the system's environment without having to retrieve basic information in prompts. Tools like Usloth allow direct support for tuning models and have large quantities of documentation for achieving positive results in Ollama models (see, Unsloth AI).

RAG System

A second milestone could be undertaken Implementing retrial augmented generation or (RAG) would also extensively improve the model's capabilities. Currently, the model relies strictly on information that is explicitly passed to it; (being prompts and recycled command output). RAG enables models to retrieve relevant information regarding a request from a local database or data store. However, this would also increase overhead costs as maintaining the concurrency of databases and processing queries can become resourcefully intensive (see, RAG).

The Bigger The Better?

A final solution that lies beyond the workstation's constraints is to include larger or more sophisticated language models. A model's size greatly impacts performance, increasing with chat history length from consecutive interactions. The currently used model is 2 GB, with a context window of 1028, averaging load times of 38-50 seconds on initial responses. While one models occupy memory replies tending to take 20–30 sounds. Larger models take up more memory thus have longer response times all around. Though by modifying parameters in model files such as num_ctx, and --keepalive arguments could aid in reducing load/response times with concurrent requests.

Limitless Potential

By default EVA is not attuned to any particular user's goals; integration with user-defined actions with custom scripted functions can improve such. Linux has multiple applications for managing the clipboard. By default, EVA only outputs text to files it creates or terminal screen. However, with custom user-defined scripts users would be able to implement their custom script to allow models to copy or paste text into their clipboard manager of choice. The script could be stored in the secondary user's

'~/bin' directory of the previously mentioned potential project file system structure. Here is a theoretical example of such an interaction:

```
user@localhost $~ eva "Summarize the selected text and add it to my clipboard."
eva@localhost $~ echo "$(clipboard --out)"; #latest clipboard entry
eva@localhost $~ espeak-ng "summary of text" && clipboard --in "summary of text";
```

In this example, the user copies a body of text, then makes a requests a summary to be appended to clipboard history. The model reads the contents of the clipboard to its working memory and then saves a summery of the output to a new clipboard entry.

Conclusion

In conclusion, the system became predominately successful in goals two and three, and reasonably effective though not exceeding in the remaining. Model components overall noticeably struggled to accurately respond to ambiguous queries. Given the limitations of the model size; adequate amounts of context is necessary to accomplish complex tasks. It's conclusive that queries of that nature are beyond the system's current capabilities. Use cases should highlight its strengths many of which include executing custom-scripted actions. Without a well-defined method of redistributing the system, it also falls short of succeeding in the final criteria of accessibility in installation.

Due to a lack of time availability development has been suspended. The ideal system design spans far beyond this small experiment. The blueprint may eventually become available publicly on GitHub, Feel free to embrace the nature of generative technologies and fork off of EVA.

Relevant Links

[1] AMD Steam Deck OLED GPU Specs. (2025). In TechPowerUp. https://www.techpowerup.com/gpu-specs/steam-deck-oled-gpu.c4185

[2] Distrobox. (n.d.). In distrobox.it. Retrieved March 20, 2025, from https://distrobox.it/

[3] GStreamer: open source multimedia framework. (n.d.). In gstreamer.freedesktop.org. Retrieved March 20, 2025, from https://gstreamer.freedesktop.org/

[4] MBROLA Voices. (n.d.). In chromium.googlesource.com. Retrieved March 20, 2025, from https://chromium.googlesource.com/chromiumos/third_party/espeak-ng/+/HEAD/docs/mbrola.md#adding-new-mbrola-voice-entry-to-espeak-ng

[5] Miessler, D., Eisler, E., github-actions[bot], xssdoctor, Connor, J., Joyce, M., JM, Guerra, A., Sylvan, K., EugenEisler, dependabot[bot], jaredmontoya, Siegel, N., buerbaumer, Degges, R., Tamim, A., David, Luo, S., Niemiec, M., … Eckii24. (2025). danielmiessler/fabric (Version v1.4.163). https://github.com/danielmiessler/fabric

[6] Ollama. (n.d.). In ollama.com. Retrieved March 20, 2025, from https://ollama.com/

[7] Steam Deck. (n.d.). In Steam Deck. Retrieved March 20, 2025, from https://www.steamdeck.com/en/

[8] Unsloth AI - Open Source Fine-Tuning for LLMs. (n.d.). In Unsloth - Open source Fine-tuning for LLMs. Unsloth - Open source Fine-tuning for LLMs. Retrieved March 20, 2025, from https://unsloth.ai/

[9] VOSK Offline Speech Recognition API. (n.d.). In VOSK Offline Speech Recognition API. Retrieved March 20, 2025, from https://alphacephei.com/vosk/

[10] What is Retrieval-Augmented Generation (RAG)? (2024). In GeeksforGeeks. https://www.geeksforgeeks.org/what-is-retrieval-augmented-generation-rag/