Deploying an LLM powered coding assistant on VMware Private AI

vmware

Introduction

We have previously introduced a comprehensive guide [1] that provides the architecture design, implementation, and best practices for deploying an enterprise-ready Generative AI (GenAI) on VMware's Private AI infrastructure. In this series of papers, we delve into the starter packs designed for deployment on this infrastructure, beginning with SafeCoder [2] which was initially announced at VMware Explore Las Vegas in Aug 2023.

When comparing to other GenAI applications within the enterprise sector [5] (for example, chatbots, content generation, and document summarization, etc.), coding assistants shine as a cost-effective and practical choice, highly regarded by customers. They demand fewer resources and significantly boost developer productivity. The introduction of Github Copilot in 2021 has further fueled their popularity, with 83% of developers adopting such tools [6]. Notably, 85% of them reported increased productivity, with 55% noting faster coding.

Yet, in response to stringent data privacy, security, and governance criteria, a surging demand exists for on-premises solutions that not only align with coding guidelines but also ensure the generation of code compliant with permissive licenses in a private and secure environment.

To address this challenge, VMware has partnered with Hugging Face to offer an on-premises solution, named "SafeCoder," which is built upon the open-source codebase LLM: *Starcoder*. By adopting SafeCoder, customers can empower their development teams with a highly capable coding assistant while retaining control over proprietary data.

This paper will lead you through the deployment of StarCoder to demonstrate a coding assistant powered by LLM. The process involves the initial deployment of the StarCoder model as an inference server. Subsequently, users can seamlessly connect to this model using a Hugging Face developed extension within their Visual Studio Code (VSC) editor, enabling them to begin enjoying the benefits of the code assistant service in a private and secure environment.

It is worth noting that customers also have an option to fine-tune the underlying model using proprietary enterprise code to align with your specific coding style and guidelines. Although this aspect is not included in this paper, we invite interested readers to explore our earlier blog [3]. Additionally, there is an enhanced VMware developed VSC extension available, and you are welcome to contact your VMware account team to assist you to obtain it, and VMware Profession Services can be engaged to provide expert guidance for deploying the full spectrum of features offered by SafeCoder.

Enterprises need a Secure Coding Assistant

Coding assistants like GitHub Copilot from Microsoft and even ChatGPT undeniably boost developers' productivity by streamlining common coding tasks, such as library calls and template code adjustments, especially in personal or small-scale projects. However, their adoption in enterprise settings presents distinct challenges. Several key concerns have been raised, reflecting the sentiments of many customers who visited our VMware Explore booth.

- **Privacy**: The primary apprehension centers around the protection of proprietary code and data. Enterprises are wary of exposing their intellectual property to public cloud services, necessitating a more secure approach.

- **Copyright Violations**: There is uncertainty regarding the code generated by these assistants and whether it falls under copyright protection. It is conceivable that code may be extracted from private proprietary websites (such as Stack Overflow, GitHub etc.), raising concerns about potential copyright violations.
- **Legal and Licensing Agreements**: Enterprises often have intricate legal and licensing agreements with partners and third parties. These agreements can dictate the specific usage of software (for example, device drivers) only in certain areas, where training and coding assistant is not included.
- **Cost Considerations**: The expense of maintaining 24/7 cloud services to support a large number of engineers in an enterprise becomes a substantial financial burden, making on-premises solutions more appealing.

Additionally, public cloud services lack awareness of an enterprise's proprietary code and standards, further necessitating the adoption of on-premises solutions. In response to these concerns, VMware is committed to providing robust on-premises solutions based on the VMware's AI ethical principles [4] to meet the strict requirements of enterprise customers.

Improving Model Performance using your proprietary code base

StarCoder is a 15.5 billion parameters Large Language Model (LLM) trained for code completion and suggestion. Its training data includes code from more than 80 programming languages, carefully curated from open-source projects and bug reports on github.com. Notably, StarCoder selectively excludes projects that could raise commercial concerns. Hence, StarCoder is considered one of the best commercial-friendly open-source coding assistant models.

Customers may choose to further improve performance of the coding assistant by further training (or fine-tuning) StarCoder using curated proprietary enterprise code. For instance, at VMware, we fine-tuned the StarCoder model with carefully selected source code from specific projects, thereby enabling it to acquire domain-specific knowledge related to VMware. This tailored approach was showcased on stage and at our booth during VMware Explore 2023. It is vital to emphasize the meticulous selection of projects within your enterprise for fine-tuning, as the quality of the data directly impacts the final quality of the generated code.

Again, our primary focus of this paper is on guiding you through the deployment solely of the StarCoder model inference server as the fine-tuning of SafeCoder has been discussed in our earlier blog [3].

Resource Planning

VM Preparation

Table 1 shows the hardware configuration of the VM used to run the StarCoder inference.

Table 1: VM setup to run the StarCoder inference server.

Component	Requirements
vCPU	4 x Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz. • More vCPUs can be used if deployed on Kubernetes with load balancer services to support more users' requests.
RAM	75 GB (around 48 GB active after loading) • Memory requirements should match, at a minimum, the size of the desired model. In the case of the 15B StarCoder model, a substantial amount of CPU memory is necessary, requiring at least 75GB, as confirmed with the "watch free -h" command. It is worth noting that CPU memory usage decreases once the model is loaded onto the GPU.
GPU	1 x NVIDIA A100 40GB PCIe. • We use this GPU as our baseline. The response time from the TGI container ranges from 1 to 3 seconds per suggestion, which translates to generating 60 tokens in approximately 25 to 40 milliseconds per token.
Network	1 x Intel(R) Ethernet Controller X710 for 10GbE SFP+ • Facilitates the connection for requests originating from users' IDEs.
Storage	150 GB • Storage requirements are a minimum of twice the size of the downloaded model. In the case of the full 15.5B StarCoder model, this translates to a minimum of 150 GB of storage. This requirement arises from the conversion of model files into "*.safetensors" files after downloading, which occupy the same amount of space. • For those interested in experimentation, it is essential to account for storage space for both the base StarCoder model and any fine-tuned models, necessitating an additional 2x increase. A storage capacity of 500GB should suffice, but for those looking to retain multiple experiments, it is advisable to plan accordingly.

Software Configuration

The table below demonstrates the software resources we used in our test on Ubuntu 22.04 LTS. And you can find the scripts to install them in our *github repo*.

Table 2: Software Resources

Software	Purpose & Installation	Version
NVIDIA Guest Driver	Guest Driver for Single VM. Disable MIG in case your python code cannot access the GPU. Install the open-source NVIDIA driver and CUDA on a preconfigured Ubuntu desktop machine. Please refer to the Installing NVIDIA Grid GPU Drivers and CUDA 11.8 section and follow the instructions	525.125.06
CUDA		12.0
Docker Engine	Allow us to use Hugging face container. Follow the steps to Install Docker Engine on Ubuntu.	latest
NVIDIA Container Toolkit	It allows us to give GPU resources to the container. Follow the steps to install and configure NVIDIA Container Toolkit.	latest

Hugging Face Account

- Registering a free Hugging Face account and create an API token.
- Ensure that you consent to the licensing terms in order to access the Hugging Face StarCoder model. Failure to do so will result in an error message when the Text Generation Inference container will try to download the StarCoder model.

Architecture

The provided example primarily focuses on a single VM scenario. However, in a real-world enterprise deployment, there is a vast potential to leverage additional resources, including vCPU, memory, and GPU. Furthermore, the system can be seamlessly expanded by deploying it on Kubernetes with load balancer support to cater to a larger user base.

Figure 1 shows that users engage with this system through the Visual Studio Code (VSC) extension, enabling them to generate JSON REST requests. These requests are then routed to the load balancer, which effectively dispatches traffic to different models, each hosted on a distinct GPU. We adopt this strategy to facilitate comparison between your original model and models that have been fine-tuned or those with larger capacities. This dynamic approach empowers users to switch between models by simply updating the URL, offering flexibility and adaptability in catering to specific needs and requirements.

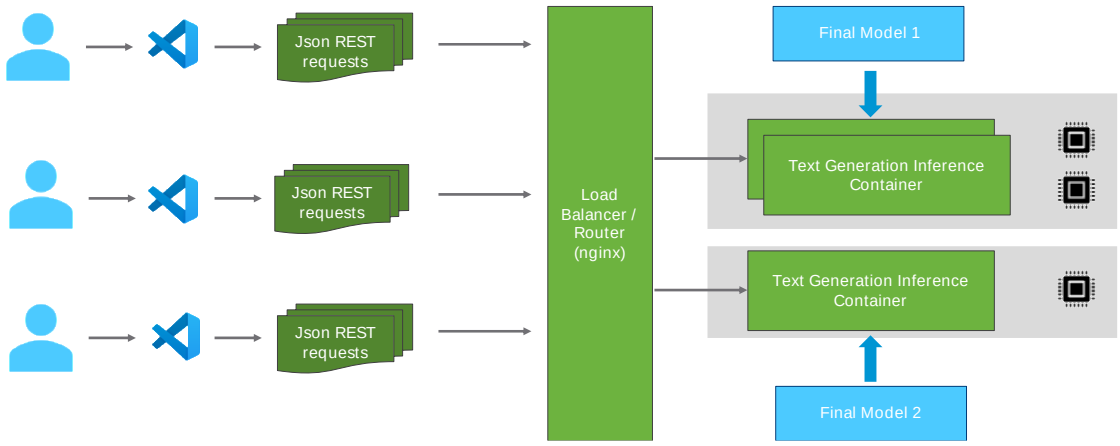


Figure 1: Architecture of SafeCoder

Deploy Code Completion Inference Service

Now we can run the following command to launch the publicly available StarCoder container provided by Huggingface for code completion service on the prepared Ubuntu vSphere VM:

```
sudo docker run --gpus 0 -v /home/octo/hf_text_gen:/data -e HUGGING_FACE_HUB_TOKEN=YOUR_HF_TOKEN --network host ghcr.io/huggingface/text-generation-inference:0.8 --model-id bigcode/starcoder --max-concurrent-requests 400 --max-input-length 4000 --max-total-tokens 4512 -p 8080
```

Take note of the following details:

- We set the variable **–gpus 0** so that it exclusively utilizes the first GPUs available on the VM. If there are two GPUs on the VM, we can reserve the other GPU (GPU1) for activities like training or running an additional code-completion server using a different model.
- We specify the server parameters **--max-input-length** to **4000 tokens** and **--max-total-tokens** to **4512**, which determine the maximum input token length the server can handle and the total tokens it can generate. The difference accommodates the model-generated code (4512 – 4000 = 512 tokens). Approximately 3 characters equate to one token, thus the max 4000 input token setting approximately equivalent to 12,000+ characters.
    - The max-input-length should be larger than the source file. If we use smaller values on the server side, the server will start returning errors to the client when the size of the source file being edited is bigger than the max-tokens limits. However, the errors are ignored by the VSC extension, so the user only notices that code-completion has stopped suggesting code after their source file reaches a certain size.
    - The "context window" within the VSC extension, introduced in the next section, refers to the **characters** transmitted to the server for code suggestions. It has a default value of 4000 characters.
- We specify **-p 8080** as the port the server will listen to.

- We pass **--network host** so that we expose the port to the VM's network and make the server accessible from outside the VM.

After downloading the model (which is skipped if it is already downloaded), the process of loading the model into the GPU takes approximately 2-3 minutes. You can confirm the server is established by observing the log, which appears as follows:

```
2023-11-03T15:22:41.113441Z  INFO text_generation_launcher: Successfully downloaded weights.
2023-11-03T15:22:41.113670Z  INFO text_generation_launcher: Starting shard 0
2023-11-03T15:22:51.124885Z  INFO text_generation_launcher: Waiting for shard 0 to be ready...
...
2023-11-03T15:24:00.771957Z  INFO shard-manager: text_generation_launcher: Server started at unix:///tmp/text-generation-server-0 rank=0
2023-11-03T15:24:00.803275Z  INFO text_generation_launcher: Shard 0 ready in 79.689004555s
2023-11-03T15:24:00.878569Z  INFO text_generation_launcher: Starting Webserver
2023-11-03T15:24:00.949517Z  INFO text_generation_router: router/src/main.rs:178: Connected
```

You can now test the server's response by generating code for **def hello()** from your laptop, provided that your laptop can establish a connection with the server. You can initiate this test using the following command by replacing with <your_server_ip> and <port>:

```
$ curl --location 'http://<your_server_ip>:<port>/generate' --header 'Content-Type: application/json' --data '{"inputs":"\n  def hello()","parameters":{"max_new_tokens":256}}'
```

The performance (time to return a code-completion sequence to the client) is based on both the GPU and CPU speeds, and on the size of the generated output. The server reports between 25ms and 40ms per token; and inference time between 1 and 3 seconds.

Use VSCode extension for Code assistant

In this paper, we used the “llm-vscode” extension developed by Hugging Face to serve as the users' endpoint. VMware also developed an enhanced called "VMware Code AutoComplete” which provides the following features:

- Accept generated code by pressing tab or ctrl+z or reject by ESC key.
- Use ctrl+e to turn on or off the code assistant feature at any time.
- On-demand one-time suggestion when the code assistant feature is disabled by pressing ctrl+x.
- Captures telemetry into internal database.
- Use Alt key instead of Ctrl in Windows. More features are upcoming.

To access the latest version of VMware Code AutoComplete, please contact to our account team.

Here are the steps to install and configure the Hugging Face extension:

1) **Disable other extensions**: To prevent conflicts with other extensions, disable any existing extensions like Tabnine.

2) **Install the extensions**: Search the “llm-vscode” in the "Extensions" and click “install”.

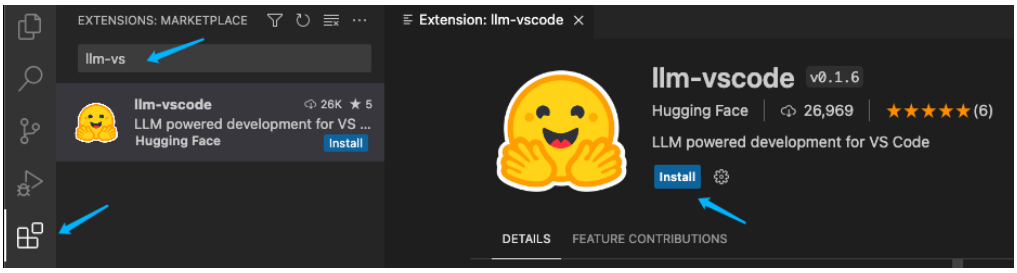


Figure 2: Install VMware Code AutoComplete Extension

3) **Update context window an Inference Server URL**: Click the “Extension Settings” and update the inference server URL with the IP address of your server VM and its port with format “https://<ip>:<port>/generate”. This ensures that the VSC extension connects to the server for code completions. If you want to change the context window, you can also change it here.

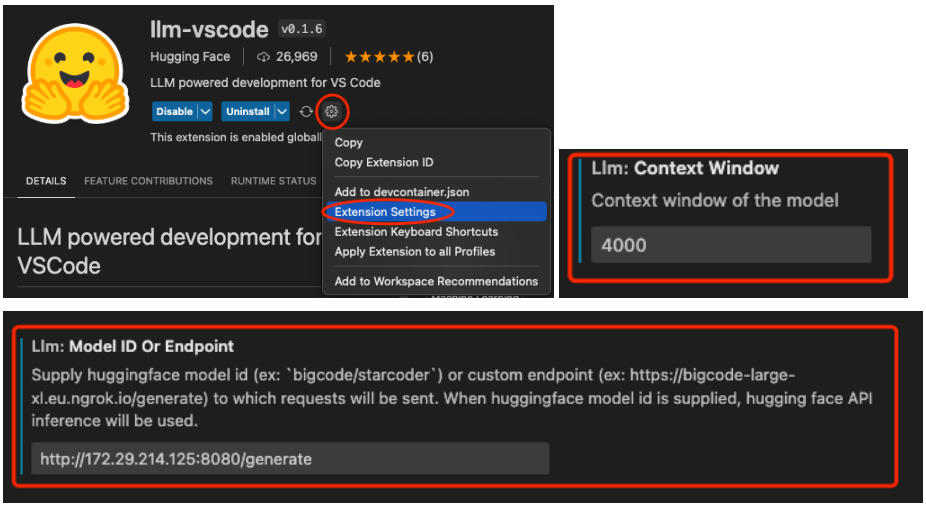


Figure 3: Edit VSC Extension Settings

4) **Check Extension Runtime Status**: Confirm that the VSC extension is running correctly by checking the “Runtime Status”.

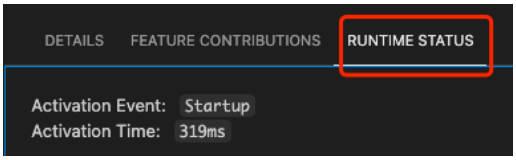


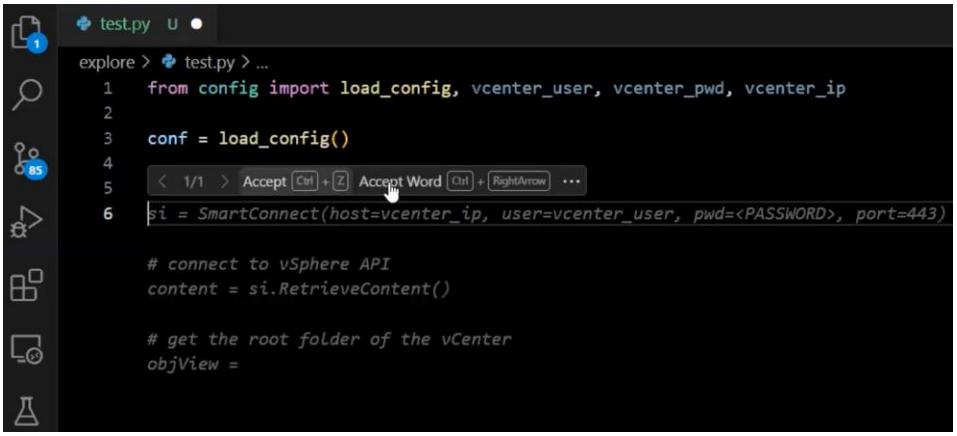
Figure 4: Check Extension Runtime Status

5) **Test Code AutoComplete**:To verify the functionality, create a new test.py file or open an existing one, and utilize the Code AutoComplete feature to experience its capabilities.

We start by adding a comment on line #3 to indicate our intention for a statement to connect to vCenter.

```
from config import load_config, vcenter_user, vcenter_pwd, vcenter_ip

conf = load_config()

# connect to vCenter
```

Shortly after, code appears with a gray color, prompting us to either accept or reject the suggestions. Note that his accept or reject feature is exclusive to the VMware-developed plugin, and the Hugging Face extension does not provide this functionality.



On line #6, we choose to accept the suggestion but also take the opportunity to replace "<PASSWORD>" with our actual password. This action reinforces the importance of ensuring sensitive data is handled appropriately during the fine-tuning of our training data.

Next, we insert a new line at #5 to import the pyVmomi library. Almost instantly, the model offers suggestions on what to import.

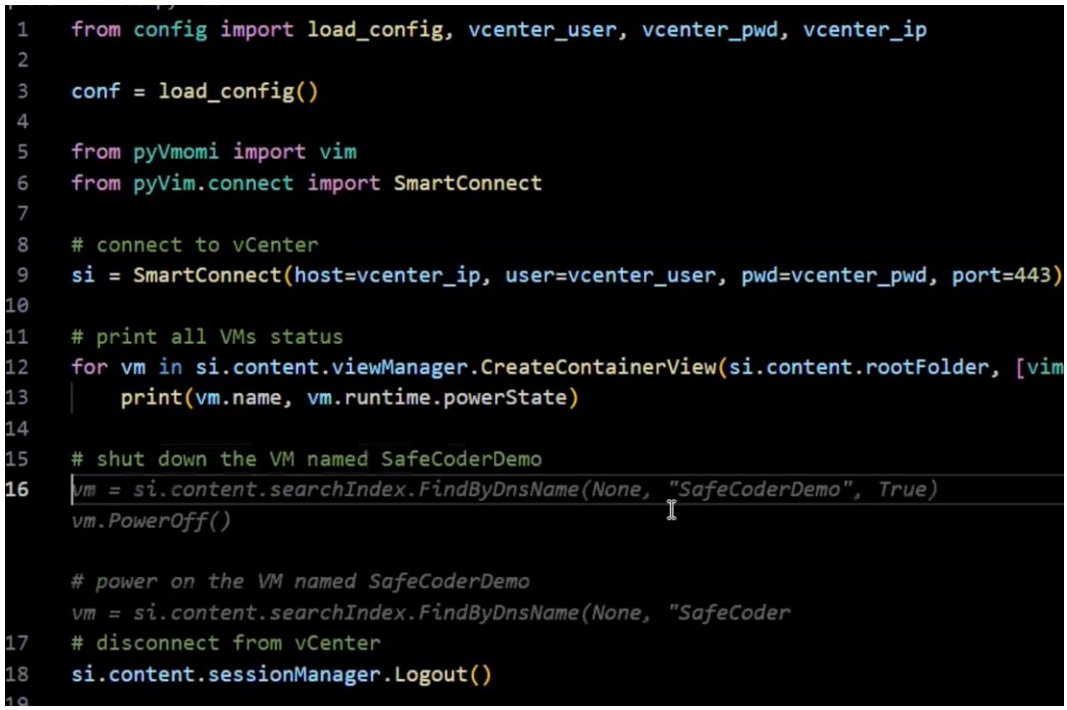
```
from config import load_config, vcenter_user, vcenter_pwd, vcenter_ip

conf = load_config()

from pyVmomi import vim
from pyVim.connect import SmartConnect

# connect to vCenter
si = SmartConnect(host=vcenter_ip, user=vcenter_user, pwd=vcenter_pwd, port=443)
```

Then, another comment is added on line #11, outlining our objective to print all VMs.

```
from pyVmomi import vim
from pyVim.connect import SmartConnect

# connect to vCenter
si = SmartConnect(host=vcenter_ip, user=vcenter_user, pwd=vcenter_pwd, port=443)

# print all VMs status
for vm in si.content.viewManager.CreateContainerView(si.content.rootFolder, [vim.VirtualMachine], True).view:
    print(vm.name, vm.runtime.powerState)

# disconnect from vCenter
si.content.sessionManager.Logout
```

Then we complete the code by adding **the closing parenthesis** after “Logout”. Additionally, we add a comment on line #15, specifying our

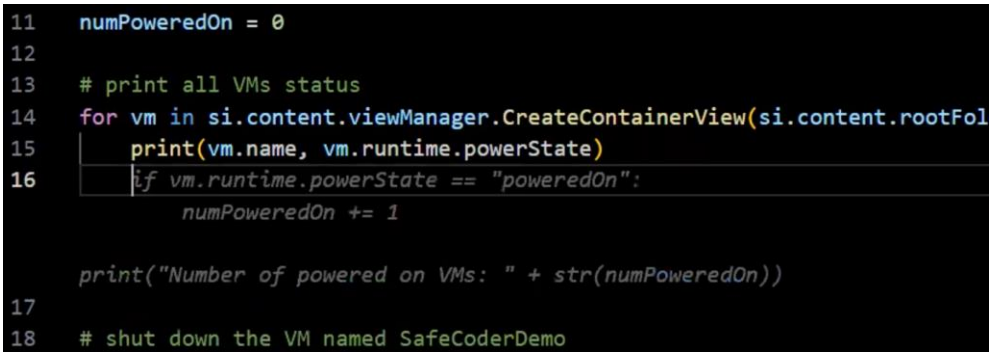
intention to shut down the VM named "SafeCoderDemo."



```
1   from config import load_config, vcenter_user, vcenter_pwd, vcenter_ip
2
3   conf = load_config()
4
5   from pyVmomi import vim
6   from pyVim.connect import SmartConnect
7
8   # connect to vCenter
9   si = SmartConnect(host=vcenter_ip, user=vcenter_user, pwd=vcenter_pwd, port=443)
10
11  # print all VMs status
12  for vm in si.content.viewManager.CreateContainerView(si.content.rootFolder, [vim
13      print(vm.name, vm.runtime.powerState)
14
15  # shut down the VM named SafeCoderDemo
16  vm = si.content.searchIndex.FindByDnsName(None, "SafeCoderDemo", True)
    vm.PowerOff()

    # power on the VM named SafeCoderDemo
    vm = si.content.searchIndex.FindByDnsName(None, "SafeCoder
17  # disconnect from vCenter
18  si.content.sessionManager.Logout()
```

The above outcomes highlight the versatility of the generated code, which not only leverages existing code from its training data but also demonstrates an understanding of context by dynamically generating code in response to specific requests.

Continually, we write a new line at #11 to initialize the count of powered-on VMs to 0.



```
11  numPoweredOn = 0
12
13  # print all VMs status
14  for vm in si.content.viewManager.CreateContainerView(si.content.rootFol
15      print(vm.name, vm.runtime.powerState)
16      if vm.runtime.powerState == "poweredOn":
            numPoweredOn += 1

    print("Number of powered on VMs: " + str(numPoweredOn))
17
18  # shut down the VM named SafeCoderDemo
```

Notably, the model comprehends our intent and anticipates that we wish to count the number of VMs and print related information. It is fascinating to observe how, by simply introducing a variable, the model adapts to the context and discerns our next steps.

Finally, we execute the Python script to test its functionality, and the results affirm that the model performs as expected.

```
(/mnt/extraspace/cg/fastchat) csodev@prme-hs2-i1021:/mnt/er/explore ; /usr/bin/env /mnt/extrasp
ace/cg/fastchat/bin/python /home/csodev/.vscode-server/extensions/ms-python.python-2023.16.0/py
thonFiles/lib/python/debugpy/adapter/../../debugpy/launcher 55295 -- /mnt/extraspace/safecoder/
explore/test.py
Rick_Ubuntu2004_A100 poweredOn
SafeCoderDemo poweredOn
Jeff_Ubuntu2004 poweredOff
Number of powered on VMs: 2
```

This showcases the model's remarkable ability to not only generate code but also anticipate and adapt to the developer's intentions based on context. By following these steps, you can seamlessly install, configure, and use the VMware Code AutoComplete extension in your Visual Studio Code environment.

Conclusion

In summary, this paper has explored the deployment of a coding assistant on the Private AI infrastructure stack, for customers that want to evaluate an enterprise-ready Generative AI coding assistant built on open source and running on their own controlled environment. This approach effectively addresses fundamental enterprise concerns, such as data privacy, licensing agreements, and developer productivity, while ensuring compliance. By operating on-premises, it provides a secure and cost-effective solution, offering a balance between innovation and data protection, making it an ideal choice for enterprises seeking to leverage GenAI technology internally.

References

1. Deploying Enterprise-Ready Generative AI on VMware Private AI
2. Introducing SafeCoder
3. Fine-tuning StarCoder to Learn VMware's Coding Style
4. Ethics in the Age of Generative AI: A Closer Look at the Ethical Principles for VMware's AI
5. Top GenAI use cases & 'no regrets' moves
6. *Research: Quantifying GitHub Copilot's impact on code quality*

About the Authors

Yuankun Fu, Giampiero Caprino and Heejeong Shin wrote the original content of this paper.

- Yuankun Fu, Senior Member of Technical Staff, Office of the CTO in VMware
- Giampiero Caprino, Staff 2 Software Engineer, VPS
- Heejeong Shin, Staff 2 Software Engineer, VPS

The following reviewers also contributed to the paper content:

- Ramesh Radhakrishnan, Technical Director, Office of the CTO in VMware
- Steve Liang, Engineering Manager, VPS
- Catherine Xu, Senior Manager of Workload Technical Marketing in VMware

Feedback

Your feedback is valuable.

To comment on this paper, contact VMware Office of the CTO at genai_tech_content_feedback@vmware.com.

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com.

vmware®