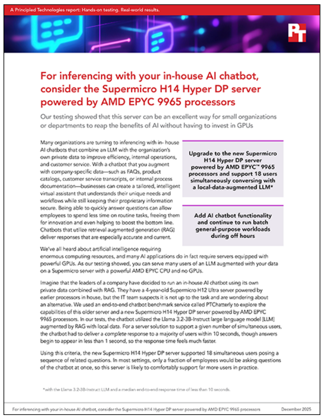
For inferencing with your in-house AI chatbot, consider the Supermicro H14 Hyper DP server powered by AMD EPYC 9965 processors

Our testing showed that this server can be an excellent way for small organizations or departments to reap the benefits of AI without having to invest in GPUs

The science behind the report:

# For inferencing with your in-house AI chatbot, consider the Supermicro H14 Hyper DP server powered by AMD EPYC 9965 processors

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report For inferencing with your in-house AI chatbot, consider the Supermicro H14 Hyper DP server powered by AMD EPYC 9965 processors.

We concluded our hands-on testing on June 28, 2025. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on June 26, 2025 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

## Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: Results of our testing on the H14 Hyper DP server.

| Results | |
|---|---|
| Max users reached | 18 |
| Cut off on question response time (s) | 20 |
| Threshold percentage | 95% |
| Median threshold (s) | 10 |

Table 2: PTChatterly parameters used in testing the H14 Hyper DP server.

| Test parameters | |
|---|---|
| Date | 06/28/2025 |
| Server | AS-2126HS-TN |
| Operating System | Ubuntu 24.10 |
| CPU | AMD EPYC™ 9965 |
| GPU | None |
| Memory | 2,304 GB |
| LLM model | meta-llama/Llama 3.2-3B-Instruct |
| Corpus | airbnb |
| Corpus # of listings | 487,974 |
| Corpus total text size (GB) | 2.3 |
| # of questions in conversation | 5 |
| Think time (s) | 2.5 |
| Think time range (s) | 1 |
| Ladder delay (s) (divided by number of users for each run) | 1 |
| User delay basis (s) | 2 |
| Max user delay (s) | 0.1 |
| Maximum # of tokens per prediction | 128 |
| Benchmark mode (Run mode is a set number of runs at a set number of users and Ladder mode ramps up a set number of users at a time until the threshold time is surpassed) | Timed Ladder |
| Begin run with # users | 1 |
| Increment by # users | 1 |
| Broker endpoint | http://supermicro-9:8072/prompts |
| Broker version | 2025.04.2.11 (21 April 2025) |
| Client version | v2025.05.2.11 (27 May 2025) |

## System configuration information

Table 3: Detailed information on the systems we tested.

| System configuration information | Supermicro Ultra DP (AS-2124US-TNRP) | Supermicro H14 Hyper DP (AS-2126HS-TN) |
|---|---|---|
| BIOS name and version | American Megatrends 3.0 | American Megatrends 1.1 |
| Operating system name and version/ build number | Ubuntu 24.10 | Ubuntu 24.10 |
| Date of last OS updates/patches applied | 11 June 2025 | 11 June 2025 |
| Processor | | |
| Number of processors | 2 | 2 |
| Vendor and model | AMD EPYC 7532 | AMD EPYC 9965 |
| Core count (per processor) | 32 | 192 |
| Core frequency (GHz) | 2.4 | 2.25 |
| Stepping | 0 | 0 |
| Memory module(s) | | |
| Total memory in system (GB) | 512 | 2,304 |
| Number of memory modules | 32 | 24 |
| Vendor and model | Micron® 18ASF2G72PDZ-3G2R1 | Micron MTC40F204WS1RC56BB1 |
| Size (GB) | 16 | 96 |
| Type | PC4-25600 | PC5-44800 |
| Speed (MHz) | 3,200 | 5,600 |
| Speed running in the server (MHz) | 2,933 | 5,600 |
| Storage controller | | |
| Vendor and model | Supermicro® AOC-S3808L-L8IT-O | Direct-attached |
| Cache size (GB) | N/A | N/A |
| Firmware version | 29.00.00.00 | N/A |
| Local storage (SATA) | | |
| Number of drives | 8 | 14 |
| Drive vendor and model | Micron HDS-M2T-MTFDDAK3T8TGA1BC | Kioxia HDS-TUN-KCD8XPUG3T84 |
| Drive size | 3.84 TB | 960 GB |
| Drive information (speed, interface, type) | 6Gbps, SATA, SSD | NVMe PCIe Gen4 |
| Local storage (NVMe / SSD) | | |
| Number of drives | 2 | N/A |
| Drive vendor and model | HDS-TUN-KCD6FLUL1T92 | N/A |
| Drive size (TB) | 1.92 | N/A |
| Drive information (speed, interface, type) | NVMe SSD Gen4 | N/A |

| System configuration information | Supermicro Ultra DP (AS-2124US-TNRP) | Supermicro H14 Hyper DP (AS-2126HS-TN) |
|---|---|---|
| Network adapter | | |
| Vendor and model | Supermicro AOC-2UR68G4-i4XTS | Supermicro AOC-S25G-B2S-O |
| Number and type of ports | 4x 10GbE | 2x 25GbE |
| Driver version | i40e 6.11.0-26-generic | bnxt_en 6.11.0-26-generic |
| Cooling fans | | |
| Vendor and model | Supermicro | Supermicro |
| Number of cooling fans | 4 | 6 |
| Power supplies | | |
| Vendor and model | Supermicro PWS-1K62A-1R | Supermicro PWS-2K63A-1R |
| Number of power supplies | 2 | 2 |
| Wattage of each (W) | 1,600 | 2,600 |

# How we tested

## Summary of software and AI models

We used the following versions of software and AI models for the project. We obtained, installed, and configured them using the procedures in the following sections:

- OS:
  - Ubuntu Server 24.10
  - Kernel: 6.11.0-26-generic)
- Docker: 28.2.2, build e6534b4
- HAproxy: 2.9.10-1ubuntu1.2 2025/04/091
- VectorDB:
  - Version: qdrant/qdrant:v1.14.1
  - Model: sentence-transformers/msmarco-distilbert-cos-v5
  - Precision: FP32
- Embedding server:
  - Version: michaelf34/infinity:0.0.76-amd
  - Model: sentence-transformers/msmarco-distilbert-cos-v5
  - Precision: FP32
- Go: 1.24.3
- PTChatterly
  - Client: v 2025.05.2.11 (27 May 2025)
  - Broker: v 2025.04.2.11 (21 April 2025)
- LLM server:
  - Version: vllm 0.9.0.1, built from Dockerfile.cpu.
  - Model: meta-llama/Llama-3.2-1B-Instruct
- Nmon: 16q

## Installing the Ubuntu operating system on the servers under test (SUTs) and client

1. Install the base Ubuntu Server (version as of last testing was 24.10), making sure that sshd is included and running.
2. Enable password-less sudo:

```
echo "$USER ALL=(ALL:ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/$USER
sudo chmod 640 /etc/sudoers.d/$USER
```

3. Set time zone, e. g.:

```
sudo timedatectl set-timezone America/New_York
```

4. Disable unattended package updates:

```
sudo systemctl stop unattended-upgrades.service
sudo systemctl disable unattended-upgrades.service
```

5. Modify the value of the unattended-upgrade variable to 0 in file /etc/apt/apt.conf.d/20auto-upgrades:

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "0";
```

6. Install the latest updates:

```
sudo apt update
sudo apt upgrade
```

7. Install standard Ubuntu packages needed in subsequent steps:

```
sudo apt install ca-certificates curl wget lsb-release sysstat smartmontools vim nmon numactl
```

8. Reboot the system:

```
sudo shutdown -r now
```

## Installing additional software on both the SUTs and client

### Installing Git and other auxiliary software

```
sudo apt install build-essential python3 python3-venv git wget tmux
```

### Installing Go

1. Remove previous versions of Go:

```
sudo rm -rf /usr/local/go/
```

2. Download version 1.24.3:

```
mkdir ~/sw
cd ~/sw
wget https://go.dev/dl/go1.24.3.linux-amd64.tar.gz
```

3. Extract the package to `/usr/local`:

```
sudo tar -C /usr/local -xzf go1.22.3.linux-amd64.tar.gz
```

4. Update your shell's path to include Go's bin directory:

```
export PATH=$PATH:/usr/local/go/bin
echo 'PATH=$PATH:/usr/local/go/bin' >> ~/.bashrc
```

5. Download the PTChatterly source:

```
cd ~/sw
git clone git@gitlab.com:principled.technologies/tools/pt-llm.git
```

## Installing the software needed only on the SUTs

### Installing auxiliary software and creating data directories

```
sudo apt install python3 python3-venv ncurses-dev
sudo mkdir -p /data/{hugginface,infinity}
sudo mkdir -p /data/qdrant/{config,storage}
```

### Installing Docker

1. Remove any previous Docker packages:

```
sudo apt remove docker.io docker-doc docker-compose docker-compose-v2 \
  podman-docker containerd runc
```

2. Add Docker's official GPG key:

```
sudo apt update
sudo apt install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

3. Add the Docker repository to the system:

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://
download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4. Install Docker CE:

```
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin \
  docker-compose-plugin
```

5. Add current user to Docker permissions group, and add Docker group to the user's current session:

```
sudo usermod -aG docker $USER
newgrp docker
```

6. Enable Docker services:

```
sudo systemctl enable --now docker.service
sudo systemctl enable --now containerd.service
```

## Installing and starting HAproxy

1. Install the Ubuntu package:

```
sudo apt update
sudo apt install haproxy
sudo systemctl disable haproxy
```

2. Configure and start HAProxy:

```
cd ~/sw/pt-llm/Scripts
bash start_haproxy.sh 127.0.0.1 8080 2
```

## Installing and starting Qdrant, the vector database

1. Download Qdrant image:

```
docker pull qdrant/qdrant:v1.14.1
```

2. Configure and start Qdrant:

```
cd ~/sw/pt-llm/Scripts
bash start_qdrant.sh vectordb-server v1.14.1 6334 cpu 32 1 0
```

## Installing and starting Infinity, the embedding service

1. Download Infinity image:

```
docker pull michaelf34/infinity:0.0.76-amd
```

2. Configure and start Infinity:

```
cd ~/sw/pt-llm/Scripts
bash start_infinity.sh embedding-server 0.0.76-amd 7997 1 31 1
```

## Installing and starting two instances of vLLM, the LLM inference service

Note 1: We ran Docker with the following options:

```
--shm-size 16G
--ipc=host
--security-opt seccomp=unconfined
--security-opt apparmor=unconfined
--cap-add=SYS_PTRACE
```

Note 2: We ran vLLM with the following options:

```
--swap-space 16
--disable-log-requests
--dtype bfloat16
--kv-cache-dtype auto
--max-model-len 8192
--max-num-batched-tokens 8192
--num-scheduler-steps 1
--max-seq-len-to-capture 256
--max-num-seqs 256
--distributed-executor-backend mp
--enable-chunked-prefill
--enable-prefix-caching
```

1. Download vLLM, version 0.9.0.1:

```
cd ~/sw/
wget https://github.com/vllm-project/vllm/archive/refs/tags/v0.9.1.tar.gz
tar xf v0.9.1.tar.gz
```

2. Create a CPU-only image:

```
cd vllm-0.9.1
docker build -f docker/Dockerfile.cpu --tag vllm-cpu-env --target vllm-openai .
```

3. Start two instances of vLLM, each pinned to one NUMA core:

```
cd ~/sw/pt-llm/Scripts
bash start_vllm_cpu.sh vllm-cpu-env  "" meta-llama/Llama-3.2-3B-Instruct 8080 2 1 1
```

## Installing nmon, a resource monitoring service

1. Download nmon:

```
mkdir ~/sw/nmon_build
cd ~/sw/nmon_build
wget 'https://sourceforge.net/projects/nmon/files/lmon16q.c/download' -O lmon.c
wget 'https://sourceforge.net/projects/nmon/files/makefile/download' -O Makefile
```

2. Compile and install nmon:

```
make
sudo install nmon_X86_Ubuntu22_16q /usr/local/sbin/nmon
```

## Installing and starting the PTChatterly broker service

1. Compile the broker:

```
cd ~/sw/pt-llm/Broker/chat
go build broker.go
```

2. Modify the runBroker.sh script so that these variables have the following values:

```
vectorEP="127.0.0.1:6334"
embedEP=http://127.0.0.1:7999/embeddings
apiType="openai"
llmType="vllm"
llmMo="meta-llama/Llama-3.2-3B-Instruct"
vectorDb="qdrant"
```

3. Start the broker inside a tmux session:

```
tmux new -s "broker-${HOSTNAME}"
./runBroker.sh
```

## Installing the software needed only on the client

### Installing nmonchart, which processed nmon data from the SUTs

```
sudo apt install ksh
cd ~/sw
wget 'https://sourceforge.net/projects/nmon/files/nmonchart42.tar/download' -O nmonchart42.tar
tar -xf nmonchart42.tar ./nmonchart
sudo install nmonchart /usr/local/sbin/nmonchart
```

### Compiling the PTChatterly client and data parser

1. Compile the programs:

```
cd ~/sw/pt-llm/Client/gohome/src/async_client
go build ptchatterly.go
go build parseResults.go
```

2. Install them:

```
mkdir ~/tests
cp ptchatterly parseResults system_info.csv runClient.sh ~/tests
```

## Ingesting the corpus data into the vector database

1. Download the Airbnb listing data, `AirbnbProps-20240830.json`, to the SUT:

```
cd ~/sw/pt-llm/Scripts
ln -s AirbnbProps-20240830.json.gz AirbnbProps.json.gz
```

2. Ingest the data into the database:

```
OMP_NUM_THREADS=12 python ingestAirBnB_qdrant.py -m -d cpu -a numa -p 16
```

## Performing PTChatterly testing

1. Log onto the client:

```
cd ~/tests
```

2. Modify the script so that these variables have the following values:

```
maxTime="20"
medThreshold="10"
mode="ladder"
```

3. Run the testing script:

```
./intRunClient.sh |& tee Test-out.txt
```

Read the report at https://facts.pt/5kPexfD

This project was commissioned by AMD and Supermicro.

Principled Technologies®

Facts matter.®