AIvailable: A Software-Defined Architecture for LLM-as-a-Service on Heterogeneous and Legacy GPUs

A PREPRINT

Pedro Antunes[1], Ana Rita Ortigoso[*1], Gabriel Vieira[†1], Daniel Fuentes[‡1], Luís Frazão[§1], Nuno Costa[¶1], and António Pereira[‖1]

[1]*Computer Science and Communication Research Centre, Polytechnic University of Leiria, Portugal,* (pedro.m.antunes, ana.l.ortigoso, gabriel.m.vieira, daniel.fuentes, luis.frazao, nuno.costa, apereira)@ipleiria.pt

November 18, 2025

ABSTRACT

The rise of Large Language Models (LLM) has increased the need for scalable, high-performance inference systems, yet most existing frameworks assume homogeneous, resource-rich hardware, often unrealistic in academic, or resource-constrained settings. We introduce AIvailable, a low-cost, highly available LLM-as-a-Service (LLMaaS) platform, that uses a software-defined approach for running LLMs across heterogeneous and legacy GPU nodes, including NVIDIA and AMD devices, with a focus on fully utilizing each node's VRAM. AIvailable operates as a fully GPU-accelerated inference without CPU fallbacks, featuring a unified client interface that allows seamless interaction with all deployed LLMs through a single logical unit. The architecture comprises four main components: the Client Interface for user access, the Service Frontend for secure request routing and load balancing, the SDAI Controller for orchestration, deployment, and monitoring, and the Service Backend of heterogeneous GPU nodes executing workloads. By abstracting GPU-specific details and providing dynamic, VRAM-aware allocation and reallocation of models, AIvailable ensures efficient use of resources and resilience against failures or workload fluctuations. Targeting academic labs, private companies, and other constrained organizations, it supports diverse open LLMs helping democratize generative AI through the repurposing of legacy GPUs.

Keywords LLM · Heterogeneous · High Availability · Low-Cost · LLMaaS · SDAI

1 Introduction

LLMs are increasingly being integrated into various aspects of daily life and professional work [1]. However, not all individuals or organizations possess the resources to deploy high-end LLM solutions, which often require access to powerful GPUs. In practice, many institutions, such as schools, universities, and small-to-medium enterprises (SMEs), rely on mid-range hardware like older NVIDIA RTX series or older AMD series GPUs, and in some cases, even legacy devices such as the NVIDIA GTX series [2]. Beyond resource constraints, some organizations also prefer to host LLMs locally due to concerns over data privacy, compliance requirements, or the need for tight integration with existing development workflows. This further reinforces the demand for low-cost, locally deployable solutions that can operate effectively on heterogeneous and resource-limited infrastructures.

[*]ORCID: 0009-0001-7529-5857
[†]ORCID: 0009-0000-2300-8441
[‡]ORCID: 0000-0001-9726-1087
[§]ORCID: 0000-0003-2571-7940
[¶]ORCID: 0000-0002-2353-369X
[‖]ORCID: 0000-0001-5062-1241

To address this gap, we introduce AIvailable, a low-cost, highly available LLM-as-a-Service (LLMaaS) platform designed specifically for SMEs and institutions with limited GPU capacity. AIvailable enables effective LLM deployment without the need for high-end infrastructure, lowering barriers to entry while maintaining accessibility and performance.

This paper introduces an architecture designed to maximize the utilization of available hardware resources, with a particular focus on leveraging the full VRAM capacity of each computational node. The approach enables users to deploy and operate LLMs in a manner that fully exploits the capabilities of their selected hardware, regardless of heterogeneity across nodes. In addition, the architecture will utilize a unified client interface through which users can seamlessly communicate with all LLM instances they have deployed, across all chosen nodes, without the need to manage separate endpoints or configurations. In doing so, AIvailable is committed to democratising the use of LLMs for everyone.

2 Related Work

The deployment of LLMs on heterogeneous and resource-constrained infrastructures has become a growing area of interest, particularly as organizations seek alternatives to high-end datacenter solutions. Several approaches have been proposed to address challenges related to availability, efficiency, and accessibility.

On this matter, [3] introduces a distributed serving architecture designed to provide low-latency inference across GPU clusters while maintaining resilience to node failures. Although effective in high-performance settings, this approach assumes access to datacenter-grade accelerators (e.g., A100 or H100 GPUs), which makes it less suitable for SMEs or educational institutions relying on legacy hardware.

Complementary to this, [4] propose adaptive scheduling policies for LLM serving, aiming to maximize GPU memory and compute efficiency under varying workloads. Their work demonstrates near-optimal VRAM utilization, but evaluations have largely been conducted on homogeneous and modern GPU clusters, leaving open questions regarding deployment on heterogeneous and resource-limited systems.

Task scheduling for distributed and heterogeneous infrastructures has also been explored in the Berkeley technical report [5], which investigates decentralized strategies for allocating workloads across diverse nodes. While the framework highlights the potential of heterogeneous orchestration, it is designed under the assumption of stable high-bandwidth interconnects, which are often not available in smaller institutional settings.

More recently, research has also examined the role of LLMs in universities. The FLEXI Project [6], describes the establishment of an open LLM infrastructure. FLEXI demonstrates how locally maintained open-source models can support teaching and research while addressing issues of data protection, operating costs, and educational equity. By leveraging moderate hardware, the project enables experimentation with LLMs without relying on commercial providers, thereby giving universities greater control over privacy and compliance concerns.

Together, these efforts provide valuable foundations for building scalable and efficient LLM infrastructures. However, most existing systems either assume homogeneous high-performance hardware or focus on experimental proofs-of-concept within well-resourced institutions. In contrast, **AIvailable** specifically targets SMEs and academic environments by enabling reliable deployment on heterogeneous and legacy GPU devices, while offering a unified interface for high availability and efficient resource utilization.

3 Proposed Architecture

The proposed architecture, as depicted in Figure 1, was designed to address the fundamental challenge of deploying and managing LLMs in environments with limited and heterogeneous GPU resources. Its purpose is to provide a low-cost yet reliable alternative to datacenter-scale infrastructures, ensuring that institutions such as SMEs and universities can still benefit from modern LLM capabilities. To achieve this, the architecture balances three core objectives: *High Availability*, by mitigating single points of failure and enabling seamless failover, *Efficient Resource Utilization*, by fully exploiting the VRAM capacity of each available device; and *Ease of Access*, by exposing a unified and transparent interface to end-users regardless of the underlying hardware diversity.

To support these objectives, the system is organized into four main components: the **Client Interface**, the **Service Frontend**, the **Software-Defined AI** (SDAI) **Controller**, and the **Service Backend**. Each component plays a distinct role in ensuring that LLMs can be deployed flexibly and operated sustainably across heterogeneous infrastructures.

The **Client Interface** serves as the primary environment for end-users to interact with the deployed LLMs. It provides access to models already running on the backend nodes, allowing users to send requests and receive responses without

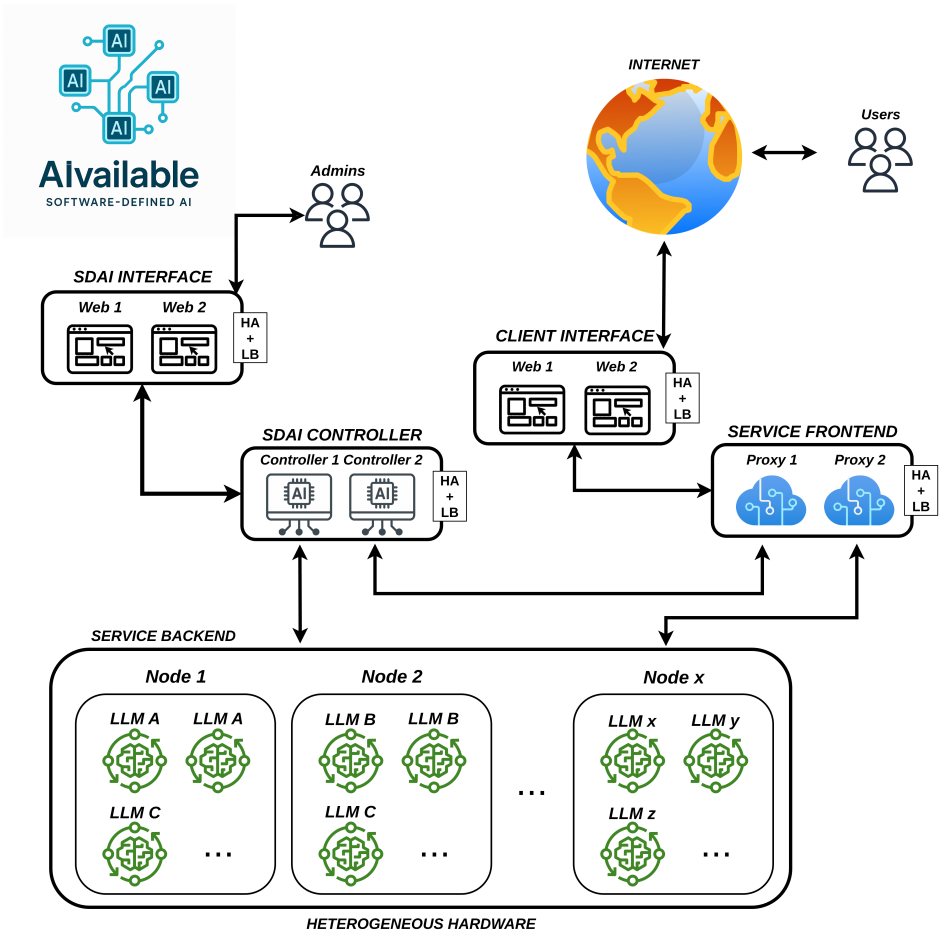

Figure 1: AIvailable Architecture

concern for the underlying routing or node selection. It does not handle model provisioning or deployment decisions, focusing solely on providing a streamlined and reliable point of interaction for inference.

The **Service Frontend** acts as the secure gateway for all client requests. It receives incoming interactions, routes them to the appropriate backend resources, and integrates HA and LB mechanisms to prevent node overload. The close integration between the **Service Frontend** and the **SDAI Controller** ensures that routing decisions are always based on the most up-to-date status of all nodes, preventing requests from being sent to unavailable or overloaded resources.

The **SDAI Controller** is the orchestration core of the system and includes its own dedicated **WebUI** for model selection and deployment management. Upon startup, it discovers and establishes communication with all backend nodes and the **Service Frontend**, registering their capabilities and current state. Through the **Controller's WebUI**, administrators can select the nodes on which specific LLMs will run, manage configurations, and initiate deployments. Once models are deployed, the **Controller** provisions access via the **Service Frontend** and continuously monitors node health, updating the UI with relevant operational data.

The **Service Backend** is composed of heterogeneous computing nodes, ranging from modern mid-tier GPUs to legacy GPUs. These nodes execute the LLM workloads assigned by the **SDAI Controller**. The system supports multiple concurrent LLM instances per node and can dynamically reallocate models to different nodes in response to workload changes or hardware constraints, ensuring efficient use of VRAM.

Operationally, the system begins with the **SDAI Controller's** discovery phase, during which it detects all available nodes, their capabilities, and any preloaded models. Administrators then use the **Controller's WebUI** to select the target nodes and LLMs for deployment. Once deployed, the models become accessible to end-users via the **Client Interface** through the **Service Frontend**. Throughout operation, the **SDAI Controller** monitors utilization, network

health, and model performance, dynamically reallocating workloads as necessary to maintain efficiency and service availability.

4 Prototype

In accordance with the proposed architecture, a prototype implementation was developed to validate the system's feasibility, performance, and operational flow. The prototype follows the same architectural pattern, with all components deployed on heterogeneous hardware to reflect realistic constraints faced by SMEs and educational institutions. Figure 2 shows the implemented prototype along with the hardware specifications of each machine used.

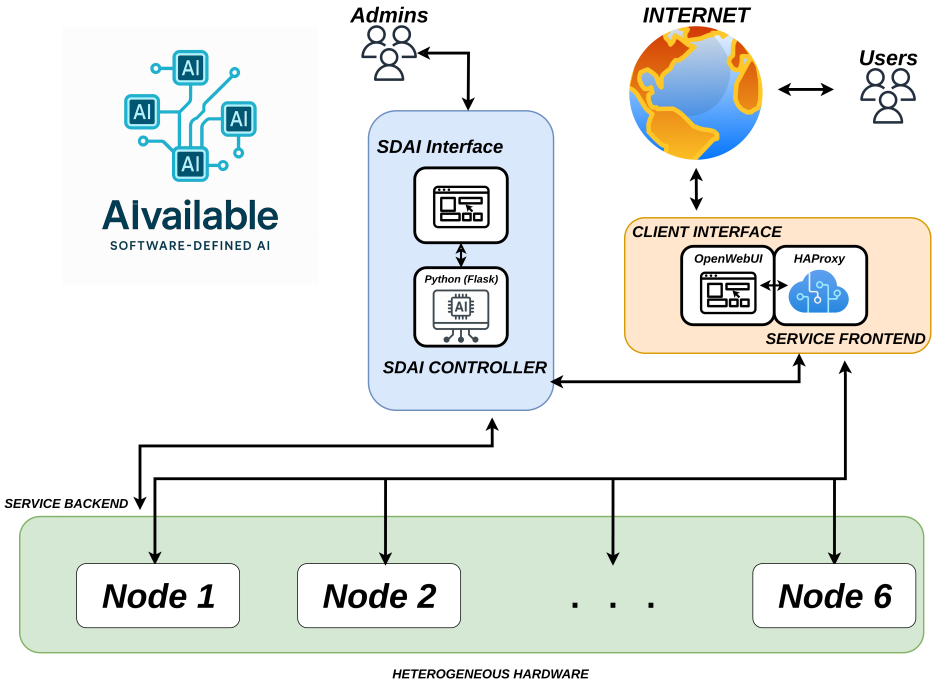

Figure 2: AIvailable Prototype

The **Client Interface** in the prototype is implemented using OpenWebUI, providing a lightweight yet feature-rich web-based environment for end-users to interact with the deployed LLMs. It is designed solely for inference usage, allowing users to send requests to models that are already running on the backend nodes. It does not handle model provisioning or deployment decisions, which are instead managed entirely through the **SDAI Controller's WebUI**.

The **Service Frontend** employs HAProxy to provide load balancing and request routing, ensuring that inference queries are efficiently directed to the appropriate backend nodes while maintaining redundancy for HA. By leveraging HAProxy's proven performance and flexibility, the system benefits from features such as health checking, connection pooling, and fine-grained traffic control. This not only maximizes throughput and fault tolerance across heterogeneous GPU backends but also simplifies scaling and operational management, making it a robust solution for production-grade LLM deployments

The **SDAI Controller**, implemented in Python using the Flask framework, serves as the orchestration core, managing node discovery, LLM allocation, and service lifecycle. Upon deployment, the controller not only provisions access through the **Service Frontend** but also sends each node a tailored HAProxy configuration based on the selected models, along with a startup script to launch the LLM instances using the Ollama framework.

The **Service Backend** in the prototype consists of six heterogeneous nodes with varying hardware capabilities, including configurations with NVIDIA RTX 3070, NVIDIA 1660 Super, AMD RX 6600 and AMD RX 6800. Each node is capable of hosting multiple LLMs matched to their available VRAM, were Models were chosen according to the computational resources available at each node, these being chosen in the SDAI Controller for each node. The list of the available LLM Models for each node, can be viewed on Table 1. Additionally, every backend node runs its own instance of HAProxy, enabling multiple replicas of the same model to run in the same node or across different nodes.

This configuration improves scalability by distributing inference workloads across replicas and enhances resilience by allowing requests to be rerouted if a particular instance fails.

Table 1: LLM Models deployed across Nodes

Nodes	Models		
1, 2, 4	deepseek-r1: 1.5b, 7b, 8b gemma3: 1b, 4b(v) llama3.2: 1b, 3b	qwen2.5vl: 3b(v) qwen3: 1.7b, 4b, 8b mxbai-embed-large	nomic-embed-text
3, 5	deepseek-r1: 1.5b, 7b gemma3: 1b	llama3.2: 1b, 3b qwen3: 1.7b, 4b	mxbai-embed-large nomic-embed-text
6	deepseek-r1: 1.5b, 7b, 8b gemma3: 1b, 4b(v) qwen2.5vl: 3b(v)	llama3.2: 1b, 3b, 11b(v) qwen3: 1.7b, 4b, 8b mxbai-embed-large	nomic-embed-text

Table 2 summarizes the hardware configuration of all backend nodes as well as the dedicated machine serving as the **Service Frontend**. These specifications highlight the heterogeneous nature of the deployment environment, ranging from modern mid-tier GPUs to legacy devices.

Table 2: Nodes Hardware Details

Node	Storage	GPU	GPU Accel. Toolkit	GPU Year
1	SSD 120GB	1x AMD RX 6600 8GB GDDR6	ROCm	2021
2	SSD 240GB	1x NVIDIA RTX 3070 8GB GDDR6	CUDA	2020
3	SSD 120GB	1x NVIDIA GTX 1660 Super 6GB GDDR6	CUDA	2019
4	SSD 240GB	1x AMD RX 6600 8GB GDDR6	ROCm	2021
5	SSD 480GB	2x NVIDIA GTX 1660 Super 6GB GDDR6	CUDA	2019
6	NVME 250GB	1x AMD RX 6800 RX 16GB GDDR6	ROCm	2020

5 SDAI Interface

This section provides an overview of the SDAI Interface, outlining its purpose, design, and core features. To enhance understanding, we include illustrative examples and images that demonstrate the interface in action, showcasing how it facilitates interaction and delivers value to end-users.

The first element presented in Figure 3 is the main dashboard of the **SDAI Interface**. This interface provides a comprehensive, real-time overview of all connected agents and their associated workloads, thereby serving as a central point for monitoring and management. The dashboard is composed of several key components.

The **Controller Overview**, located at the top bar, displays the number of connected and available agents, the last update timestamp, and the current connection status. The **Active Agents Section** presents each agent as a card, detailing hardware specifications such as GPU type, VRAM, and last communication time. It also lists the running model instances, which include both LLMs (e.g., *llama3.2*, *gemma3*, *deepseek-r1*) and embedding models (e.g., *mxbai-embed-large*, *nomic-embed-text*). The **Main Agent**, highlighted in purple, hosts the HAProxy Manager, which manages load balancing and service restarts, ensuring high availability and reliability. Finally, at the bottom of the dashboard, the **Configuration Wizard** provides tools to generate optimized HAProxy and Ollama configurations based on GPU capabilities and model requirements. Together, these features make the interface a central hub for monitoring, managing, and scaling AI deployments.

After that, we can view Figure 4, which presents the **Configuration Wizard** of the SDAI Interface. This component enables users to select agents and configure GPU instances for deployment, thereby ensuring that resources are allocated according to system requirements. The wizard follows a stepwise approach that facilitates clarity and usability. It is divided into three stages: **Select Agents**, **Configure**, and **Generate**, which together guide the user through the configuration process in a systematic manner. During the **Target Agent Selection** stage, users may choose one or more agents for GPU deployment, with the option to select all standard agents simultaneously, simplifying larger deployments. For each agent, the interface provides detailed information, including its connection status, last

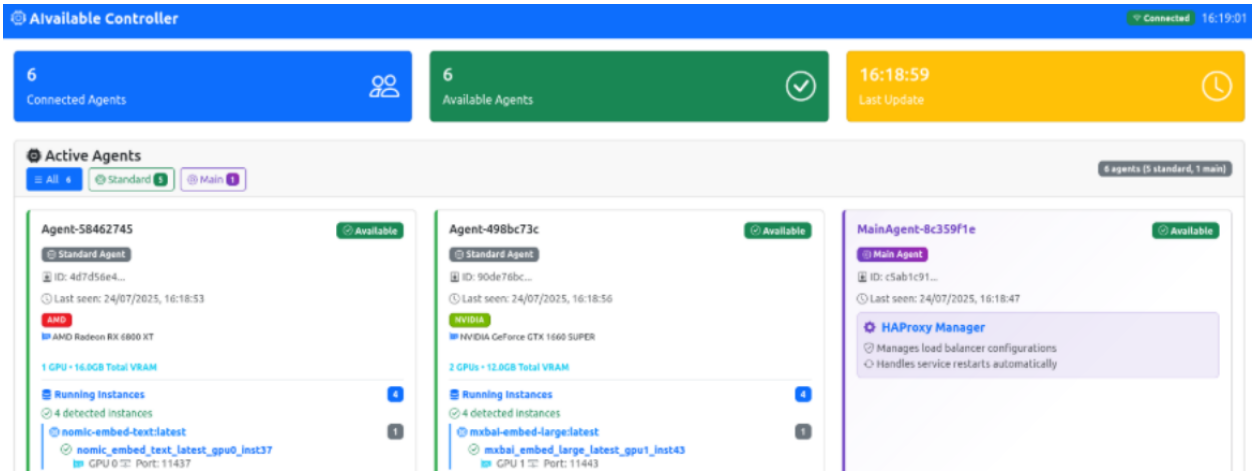

Figure 3: SDAI Interface Dashboard

communication timestamp, and GPU specifications such as vendor (e.g., AMD, NVIDIA), model, and available VRAM. In addition, users are given an explicit option to include or exclude each agent from the configuration process.

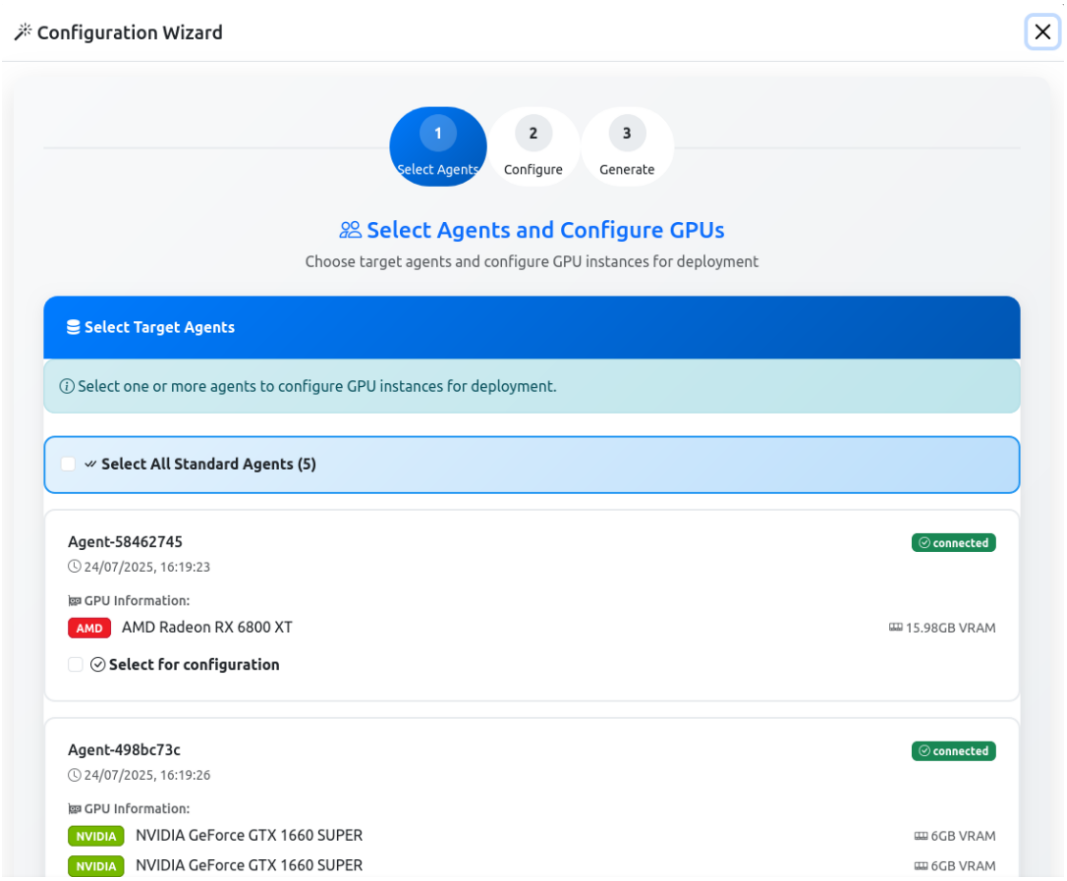

Figure 4: SDAI Configuration Wizard

5.1 Select Agents

A more detailed view of the *Select Agents* stage is provided in Figure 5. Once an agent has been selected, the user is presented with the option to enable specific **GPU instances** for configuration, showing a clear listing of available GPUs for each selected agent, model type, and associated VRAM capacity. There is also an enable/disable toggle for each GPU, allowing the user to select precisely which devices will participate in the deployment process.

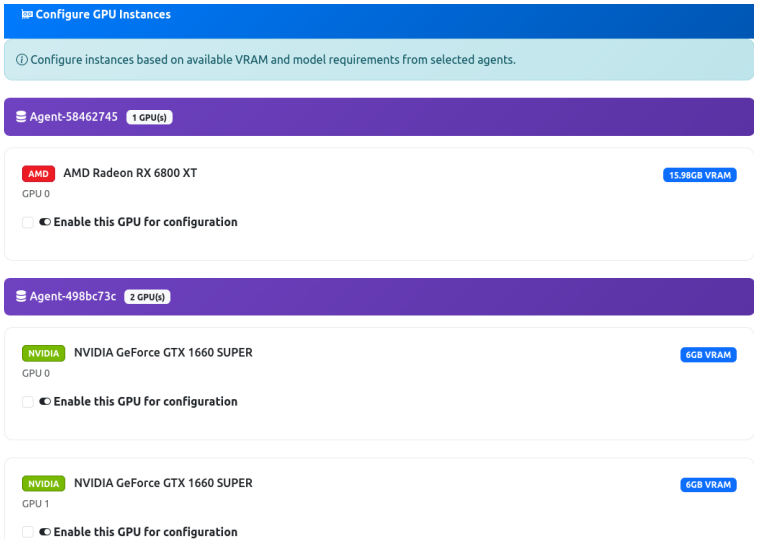

Figure 5: SDAI GPU Selection

After enabling a GPU for configuration, the user is provided with the option to select which **LLM instances** will be deployed. As shown in Figure 6, this selection is made from a predefined list of available models, which includes a variety of architectures and parameter sizes (e.g., *DeepSeek*, *Gemma*, *Llama*, *Qwen*, and embedding models). The interface also displays information about **model capacity**, including the VRAM required per instance, the available VRAM on the selected GPU, and the maximum number of instances that can be allocated.

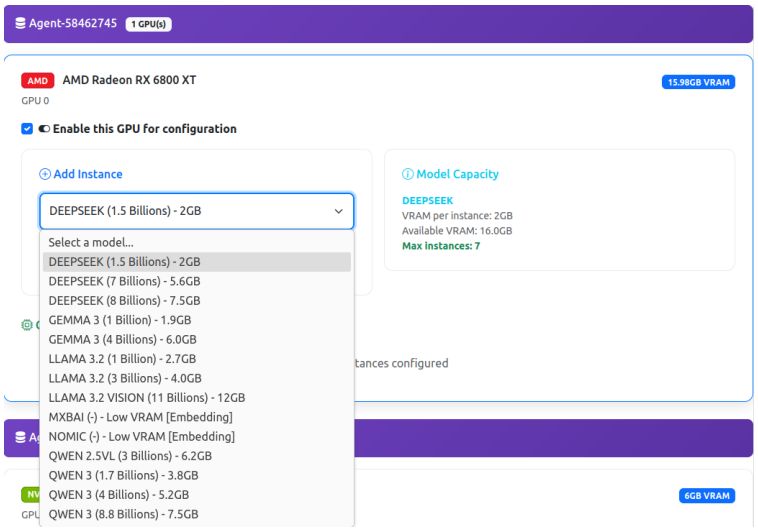

Figure 6: SDAI Model Selection

This design enables users to align model selection with hardware constraints, ensuring efficient resource utilization while maintaining flexibility in the deployment of different model families and sizes.

5.2 Configure

After selecting the agents and assigning specific model instances to each GPU, the process advances to the **Configure** stage. As illustrated in Figure 7, this stage allows the user to define the **network ports** through which each model instance will be accessed. For each deployed model (e.g., Qwen 2.5VL, DeepSeek, Qwen 3, Nomic, MXBAI), the interface provides:

- A clear indication of the number of running instances and the GPUs on which they are hosted.
- An automatically suggested default port, with the option for the user to manually adjust it as required.
- A load balancing mechanism across instances, ensuring that requests are evenly distributed for models with multiple replicas.

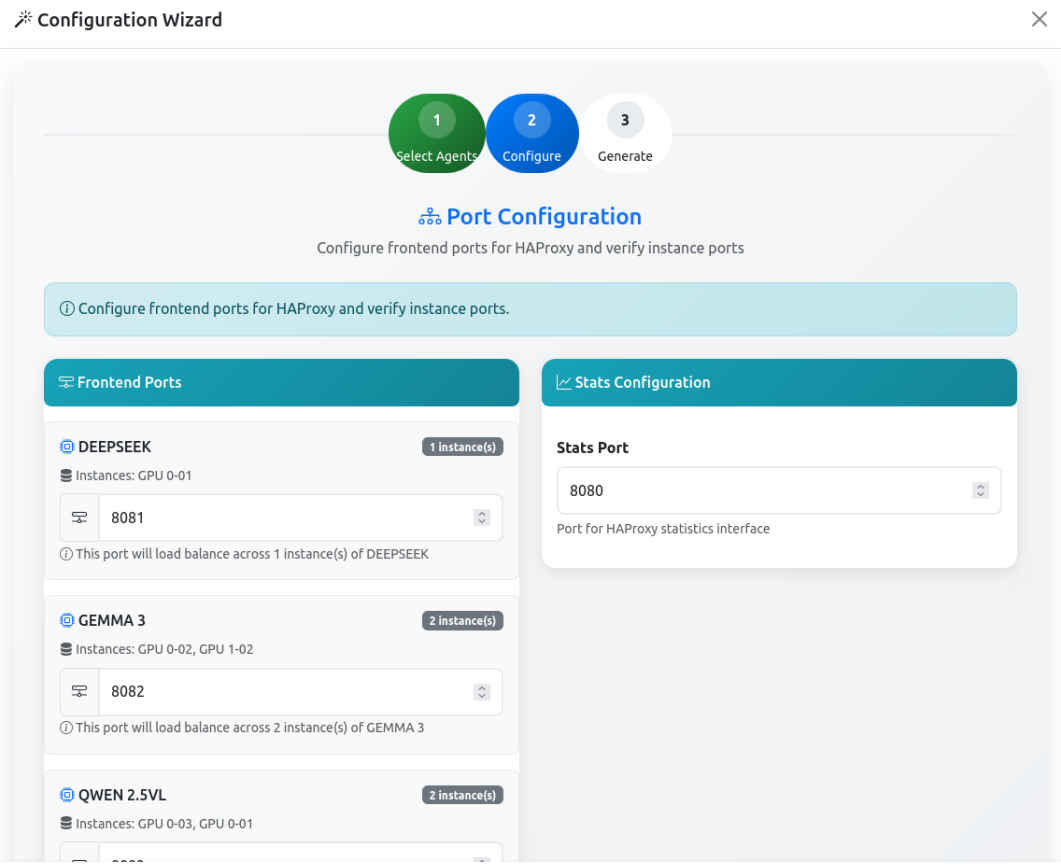

Figure 7: SDAI Port Configuration

5.3 Generate

The final stage, **Generate**, presents the **Configuration Overview**, which provides a comprehensive summary of the multi-agent deployment. As shown in Figure 8, this stage consolidates the configuration details and displays them in a structured format, enabling users to verify the allocation of resources before finalizing the setup. The overview includes:

- **System Statistics** - a summary panel indicating the total number of agents, deployed instances, distinct models, and the statistics port being used.
- **Model Distribution** - a breakdown of all deployed models, along with the number of instances allocated to each.

- **Agent Distribution** - a detailed mapping of how many instances are deployed per agent, as well as the GPU resources available to each agent.

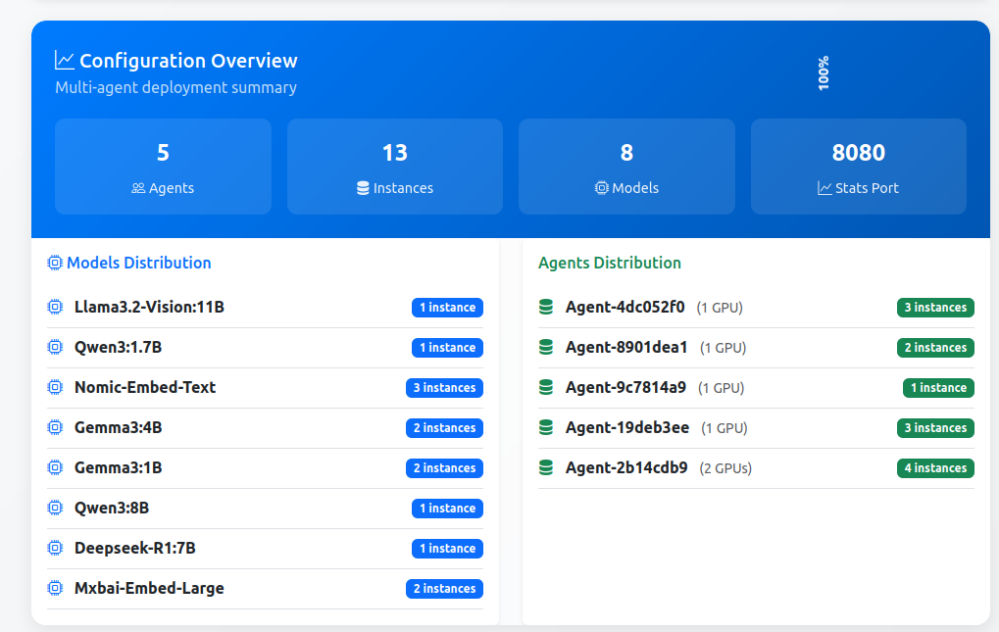

Figure 8: SDAI Configuration Overview

6 Discussion

AIvailable demonstrates that a software-defined approach can make multi-agent LLM serving feasible on heterogeneous and partially legacy GPU fleets. The SDAI Interface reduced orchestration complexity by exposing a single control surface for agent discovery, model placement, port assignment, and HA configuration. HAProxy's role simplified rollout and rollback, and its health checks provided early detection for instance drift.

Replica-level load balancing masked single-instance failures; however, it does not address *coordinated* failures (e.g., node-level outages). Observability currently focuses on liveness and capacity rather than application-level quality.

The stepwise wizard (**Select** $\rightarrow$ **Configure** $\rightarrow$ **Generate**) lowered the barrier for non-experts, but also centralized power. Furthermore, local hosting was a motivator for privacy; documenting data-handling guarantees will be important for institutional adoption.

We did not provide quantitative benchmarks in this study. As such, claims about agility and responsiveness remain qualitative. External validity is limited by the size of our testbed and by the particular set of open models considered. Finally, cost analysis (energy, amortized hardware, and ops) was out of scope.

7 Conclusion

This work introduced **AIvailable**, a software-defined architecture and working prototype for LLM-as-a-Service on heterogeneous and legacy GPU infrastructures. Our main contributions are:

1. An architecture that couples a lightweight control plane (SDAI Controller) with a standards-based data plane (HAProxy), enabling high availability and unified access across diverse nodes.
2. A practical orchestration workflow, delivered through the SDAI Interface, that streamlines agent discovery, VRAM-aware model placement, and configuration generation.
3. A demonstration on mixed ROCm/CUDA hardware showing that reliable, multi-agent LLM serving is attainable without datacenter-class GPUs.

Looking ahead, the developed architecture and prototype set the foundation for further advancements in orchestrating modern LLMs on non-uniform, GPU-driven infrastructures without reliance on high-end datacenter hardware. Future work will involve large-scale validation, including benchmarking inference latency, throughput across diverse model configurations, and assessing scalability under realistic workloads. Additional directions include exploring advanced strategies for dynamic model allocation and investigating mechanisms to further enhance fault tolerance. With these steps, we aim for **AIvailable** to contribute meaningfully to democratizing access to generative AI in academic, research, and SME environments.

References

[1] Mubashar Raza, Zarmina Jahangir, Muhammad Bilal Riaz, Muhammad Jasim Saeed, and Muhammad Awais Sattar. Industrial applications of large language models. *Scientific Reports*, 15(1):13755, April 2025.

[2] Jeremy Stephen Gabriel Yee, Pai Chet Ng, Zhengkui Wang, Ian McLoughlin, Aik Beng Ng, and Simon See. On-device llms for smes: Challenges and opportunities, 2024.

[3] Youhe Jiang, Fangcheng Fu, Xiaozhe Yao, Taiyi Wang, Bin Cui, Ana Klimovic, and Eiko Yoneki. Thunderserve: High-performance and cost-efficient llm serving in cloud environments, 2025.

[4] Siddharth Jha, Coleman Hooper, Xiaoxuan Liu, Sehoon Kim, and Kurt Keutzer. Learned best-effort llm serving, 2024.

[5] Elden Ren. Task scheduling for decentralized llm serving in heterogeneous networks. Master's thesis, EECS Department, University of California, Berkeley, May 2024.

[6] Torsten Zesch, Michael Hanses, Niels Seidel, Piush Aggarwal, Dirk Veiel, and Claudia De Witt. Flexible llm experimental infrastructure (flexi) – enabling experimentation and innovation in higher education through access to open llms. In *2024 21st International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–8, 2024.