- **Leftover resource utilization:** Enables collaborative sharing of leftover capacity across multiple devices within the enterprise.
- **Optimal block distribution:** Ensures optimal distribution of transformer blocks on an enterprise-wide scale, using OPA (optimal placement algorithm).
- **Multi-Client and Multi-model support:** Supports multiple client requests concurrently. Enables loading of blocks associated with multiple LLMs on a single resource, thus optimizing overall resource usage in the enterprise.
- **Cost optimization:**LLaMPS distributes LLM across multiple devices, utilizing leftover resources, reducing the need for expensive hosting.

LLaMPS is a tool that adopts a client-server architecture, developed in Python, and utilizes the underlying open-source Petals distributed framework. It has a web-based interface, featuring a user-friendly Streamlit-built GUI. LLaMPS is adaptable and offers flexibility to support future alternative distributed frameworks.

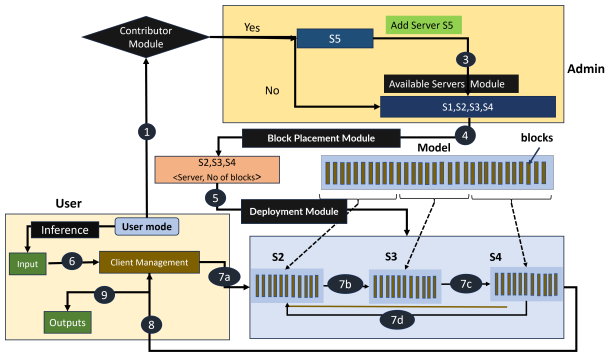

Figure 2: LLaMPS Architecture

2 LLAMPS ARCHITECTURE

We now discuss the architecture of LLaMPS depicted in Figure 2.

The user interacts with the LLaMPS tool via the user interface (Figure 1). The **Contributor Module** enables the user to *share or contribute* leftover capacity of his device with the distributed network. If the user chooses to contribute, the IP address and server information (available memory and cores) of the device are provided to the **Admin**. If not, it implies that the user only wishes to perform an *Inference* task.

Available Servers Module enables displaying a list of resources that are available on the network, along with their IP addresses, available memory, and cores on the UI. This information is then sent to the admin.

The **Block Placement Module** then kicks off the OPA (Optimal block Placement Algorithm) which takes input all info about the available servers and creates a plan for block placement. The plan outlines a list of selected servers (from the available servers) and the distribution of blocks to be loaded on each server for efficient resource utilization. Additional details of OPA can be found in our paper [1].

The **Deployment Module** deploys the blocks on the servers as per the plan. For example, Blocks (0:12) are loaded on Server 1; Blocks (12:24) are loaded on Server 2, and so on. Once the blocks are deployed, the user can start using the deployed model for the inference task. The user inputs text and initiates the **Inference process**.

In the client, a route is formed based on the sequence of blocks deployed on multiple servers, and the inference runs across these servers. The inference output from the blocks is then converted into a human-readable form and the output is then transmitted to the user interface.

3 USE CASE

Figure 1 depicts the user interface of the LLaMPS tool. The user will input the model name and the number of parameters of the model. The number of blocks corresponding to the model will be automatically displayed. The user may choose to contribute or share his device's leftover resources or perform inference without sharing any resources. Upon clicking the *Available Servers* button, the list of available servers will be displayed. The user then clicks the *Optimal Placement Algorithm* button which creates a plan for optimal distribution of blocks. The blocks are deployed by clicking the *Deploy* button. Once the deployment is complete, the user can then perform inference as displayed in the right pane.

Assume that an enterprise has the following list of available servers s1<155, 8>, s2<113.5, 8>, s3<83.5,2>, s4<83.5,4>, and s5<83.5,8>. The first value in the tuple represents the available leftover memory in GBs and the second value represents the number of available cores. The user wants to deploy the LLama2 70b model, which comprises 80 transformer blocks. The size of LLaMA2 is approximately 300GB including overheads. No single server within the enterprise can host the entire LLama2 model. By leveraging the combined memory capacities of multiple servers, the OPA algorithm of LLaMPS creates a plan for deployment. Servers s1, s2, and s5 are selected taking into account available memory and cores and will host transformer blocks 0-34, 35-60, and 61-79 respectively.

During an inference cycle, the client management(step 7a in Figure 2) tokenizes the input. The tokenized input is then relayed to server s1, where it traverses the allocated transformer blocks. The intermediate output is sequentially passed to server s2 and subsequent servers until it has traversed all the transformer blocks. The final output from the last server in the sequence is transmitted back to the client(step 8 in Figure 2), where the required output is generated.

4 CONCLUSION AND FUTURE WORK

LLaMPS is instrumental in addressing the challenges associated with deploying LLMs on limited computational resources. It achieves this by efficiently distributing the computational workload of LLMs across multiple machines, using the OPA algorithm. OPA not only optimizes the utilization of residual computing power but also manages the dynamically varying leftover memory and compute resources optimally. We plan to enhance the capability of the LLaMPS tool by extending it for optimal utilization of enterprise cloud resources when deploying LLMs.

REFERENCES

[1] Ravi Kumar Singh, Likhit Bandamudi, Shruti Kunde, Mayank Mishra, and Rekha Singhal. 2024. Leftovers for LlaMA. In *International Conference on Performance Engineering(accepted)*. ICPE.