

PlanetServe: A Decentralized, Scalable, and Privacy-Preserving Overlay for Democratizing Large Language Model Serving

Fei Fang^{*†} Yifan Hua^{*†} Shengze Wang^{*†} Ruilin Zhou[†] Yi Liu[†] Chen Qian[†]

Xiaoxue Zhang[‡]

Abstract

While significant progress has been made in research and development on open-source and cost-efficient large-language models (LLMs), serving scalability remains a critical challenge, particularly for small organizations and individuals seeking to deploy and test their LLM innovations. Inspired by peer-to-peer networks that leverage decentralized overlay nodes to increase throughput and availability, we propose PlanetServe, an LLM serving overlay that harnesses computing resources from decentralized contributors. We identify four key research problems inherent to enabling such a decentralized infrastructure: 1) overlay network organization; 2) LLM communication privacy; 3) overlay forwarding for resource efficiency; and 4) verification of serving quality. This work presents the first systematic study of these fundamental problems in the context of decentralized LLM serving. Evaluation results from a prototype implemented on a set of decentralized nodes demonstrate that PlanetServe achieves a latency reduction of over 50% compared to the baseline design without overlay forwarding. Furthermore, the security features introduce minimal overhead to serving latency and throughput. We believe this work pioneers a new direction for democratizing and scaling future AI serving capabilities.

1 Introduction

The rapid development and proliferation of open-source, cost-efficient large-language models (LLMs)—such as Google’s Gemma [6], DeepSeek-R1 [16], and Meta’s LLaMA series [52]—have significantly lowered the barrier for deploying AI systems. This progress makes it feasible to run advanced language models on personal workstations and local servers with consumer GPUs. Despite this accessibility, **serving scalability remains a critical challenge for small organizations and individuals to deploy their LLM innovations**. Individual researchers, university laboratories, and start-up companies often find it prohibitively expensive and technically challenging

to deploy their models at scale, particularly when popularity surges quickly. For instance, some promising models initially provided excellent performance at low cost, but later suffered severe performance degradation when user demand suddenly increased [2]. Traditional cloud-based infrastructures, which are not always affordable or accessible for smaller organizations and individuals.

We propose PlanetServe, a decentralized overlay infrastructure designed for large-scale LLM deployment. Inspired by PlanetLab [15], a well-known peer-to-peer (P2P) network allowing researchers to deploy and test Internet applications across distributed nodes, PlanetServe is designed to harness computing resources from a diverse range of contributors, including individuals, academic institutions, and small organizations. Participants in PlanetServe volunteer their computing power and network resources, and in return gain the ability to deploy their own LLMs and AI-driven services on this shared, decentralized network.

PlanetServe provides the following main advantages:

- Scalability, resiliency, and cost-efficiency.** PlanetServe can dynamically adapt to rapid fluctuations in demand, eliminating the performance bottlenecks commonly associated with centralized cloud deployments that depend on rigid, pre-defined service agreements. Idle computing resources can be utilized instead of expensive dedicated cloud instances. PlanetServe increases scalability on both computation and networks. Hence, it can resolve the scalability problem for computation (e.g., large models), networks (e.g., small models), or a mixed situation.
- Democratization of AI innovation development.** PlanetServe significantly lowers the barrier to independent researchers, developers, and academic institutions being able to deploy and experiment with their LLM innovations at scale. Hence, it encourages innovation from a wider and more diverse contributor base.
- Enhance user privacy.** Conventional cloud deployments often centralize user prompts and associated private information, creating a single point of trust and potential vulnerability. PlanetServe addresses this by incorpo-

^{*}Co-primary authors.

[†]University of California, Santa Cruz

[‡]University of Nevada, Reno

rating anonymous overlay networks, preventing model-serving nodes from easily linking user prompts to individual identities, thereby enhancing user privacy.

4. PlanetServe naturally inherits benefits that are common to decentralized overlay systems, such as service redundancy, geographic distribution of resources, and load balancing among heterogeneous servers.

PlanetServe thus represents a novel approach to deploying LLMs and other advanced AI services, democratizing large-scale AI capabilities and enhancing user privacy. Our position is that model serving today is mostly concentrated in large-scale cloud deployments, but the software ecosystem [36] has already demonstrated that inference can be effectively deployed across a wide range of devices. This trend suggests that the future inference landscape should also be determined by cost efficiency, privacy, and availability. A decentralized infrastructure like PlanetServe is well-positioned to adapt to such a landscape: whether the dominant models are large or small, participants can contribute heterogeneous resources that align with their needs (e.g., low-latency private inference or specialized model serving).

In designing and implementing PlanetServe, this work addresses four core research challenges that are critical for any decentralized LLM serving infrastructure: 1) Overlay network organization. 2) Anonymous communication to protect user privacy. 3) Overlay forwarding for serving efficiency. 4) Decentralized verification of compliant model serving. **To our knowledge, this is the first work to systematically study these core challenges for achieving LLM serving scalability via a decentralized overlay network independent of centralized cloud control.**

We implemented PlanetServe’s core functions as a prototype consisting of decentralized nodes and tested the prototype in a public cloud without central control. This approach provided a controllable testbed, necessary as we do not have direct access to a large set of physical decentralized servers across the Internet. We let each node add synthetic latency to every packet for the wide-area Internet conditions. **Upon acceptance of this paper, we will make the prototype implementation and datasets open-sourced.** We plan to deploy and maintain PlanetServe at scale in the future.

This work focuses exclusively on the LLM serving problem in a decentralized overlay network and assumes model training is completed by model developers. We do not consider decentralized LLM training [57] or collaborative serving an LLM instance [9, 40, 60]. This work is also different from distributed LLM serving in a cloud with centralized scheduling [34, 40, 50, 59, 60].

The rest of this paper is organized as follows. Section 2 presents the system models and design objectives of PlanetServe. Section 3 details the design of PlanetServe. We provide the security analysis and evaluation in Section 4. The performance evaluation results are presented in Section 5. We discuss the related work in Section 7. We conclude this work

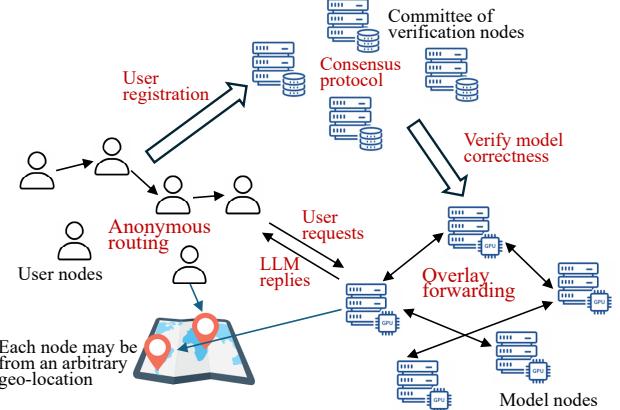


Figure 1: Overview of PlanetServe.

and discuss possible future work in Section 8.

2 System Models and Objectives

2.1 Network Model

PlanetServe operates as an overlay network composed of end hosts, each plays as one or more of the following roles:

1. **User node** represents end-users (clients) seeking access to LLM services while preserving privacy.
2. **Model node** executes LLM inference tasks to serve user requests. These nodes typically run on servers with GPU resources, contributed by participating organizations or individuals. Each model node serves one or more complete LLM instances. Collaborative inference across multiple nodes for a single request [9, 60] is not considered.
3. **Verification node.** To maintain correct LLM serving and defend against security attacks, PlanetServe includes a committee of decentralized verification nodes, far fewer than users and model nodes. Each verification node runs on a server owned by an organization or individual with certain credentials, such as a well-recognized university research team. They are to ensure that each model node behaves legitimately and manage user and model node information. This committee runs a BFT consensus protocol [11] to ensure information correctness and consistency.

A single physical device may operate in two or all roles in PlanetServe. All communications between nodes in PlanetServe are via TCP, secured with TLS. User nodes may join or leave the network at any time and hence network churn can happen. Model and verification nodes are more stable but they still can fail or be compromised by attackers.

2.2 Cost Advantage and Incentive Model

Cost advantage of PlanetServe. The cost advantage of PlanetServe compared to commercial public cloud is mainly from two facts. First, computation resources remain dramatically underutilized in organizations. Hence, the cost of contributing servers to PlanetServe is very low. Many organizations deployed GPU servers for ML training, but the servers sit idle outside of training times. It has been reported that the average

server utilization rate is between 12%-18% of capacity [5]. 10 million servers are estimated to sit completely idle, representing \$30 billion in wasted capital [5]. These idle servers can be contributed to PlanetServe and form a large-scale serving infrastructure. Second, using community-contributed GPUs for inference can dramatically cut costs compared to commercial cloud platforms. Measurements show that the energy cost of running LLM inference on GPUs such as the V100 and A100 is relatively modest [45], implying that the high hourly rates of public cloud instances are driven largely by infrastructure and pricing overheads rather than intrinsic hardware or energy expenses. Recent work on scalable inference similarly reports $20\times$ lower cost when serving LLMs on consumer GPUs versus AWS instances [1]. In addition, many consumer GPUs or GPUs that have been given away from training can still serve moderately sized LLMs (7B–13B parameters).

In summary, prior measurements and cost analyses suggest that LLM serving and testing using private hardware can be an order of magnitude cheaper than compared to public cloud if the hardware is fully utilized. However, scaling private hardware is challenging, and hardware utilization is very low. PlanetServe addresses both the scalability and under-utilization problem of private hardware by allowing different organizations to contribute their resources and share them with each other.

PlanetServe uses a **reputation-based incentive model** to encourage participation and cooperation. The incentive model only applies to the organizations that contribute model nodes to the system. There is no system-wide incentive mechanism for users. However, model deployers can use their own policies to regulate users who can access their LLMs.

Incentive for participation. Each organization participates in the system by contributing a number of model nodes. Model nodes from the same organization share the same reputation score λ , which will be updated during the system execution, as detailed in Sec. 3.4. If the reputation score is above a threshold, the organizer is allowed to deploy their own LLM to the system. To measure the resources it is allowed to use for deploying its LLM, each organization also has a contribution credit, which reflects the resource time it contributes to the system. The contribution credit is calculated proportionally to the cost of renting servers from a public cloud. For example, if an organization has contributed 5 servers that have been serving for 30 days in PlanetServe, it can deploy its LLM to PlanetServe that runs on 30 servers with similar computing resources for 5 days. All reputation scores and contribution credits are assigned and updated by the committee of verification nodes running a consensus protocol.

Incentive for cooperation. If model nodes do not follow the PlanetServe protocols in model serving and request forwarding, their reputation scores will be reduced, which will hurt their eligibility to use other resources in PlanetServe. The verification node committee is responsible for ensuring that the model nodes correctly run the models they promise to

serve and for maintaining the reputation scores of the model nodes, as detailed in Sec. 3.4.

2.3 Threat Model and Assumptions

Consistent to many existing works [25, 27, 41], this work focuses on **practical low-delay security and anonymity**, rather than perfect security at all costs. We assume the following attacks by different nodes in PlanetServe.

- Malicious user nodes may drop packets they are responsible for forwarding. They may try to deanonymize the sender of a request and inspect packet contents. We also assume user nodes that can collude to conduct these attacks.
- A dishonest model node may drop user requests on purpose or reply with incorrect model results to mislead users. This includes manipulating outputs or substituting the agreed-upon model with a cheaper, lower-quality one. For example, the node claims to serve an 8B-parameter LLM but runs a 1.5B model instead to conserve resources. It will try to differentiate normal requests from users and verification requests to treat them differently. Even though a model node does not perform active attacks, it might attempt to link the received prompts and their user nodes. It might falsely claim resource exhaustion to refuse service. A malicious model node may also be curious about sensitive prompt content, such as users’ personal data or proprietary information.
- A compromised verification node could provide wrong information to users or false verification results for model nodes. We assume an adversary, potentially leveraging Sybil attacks to control multiple verification nodes. However, the adversary cannot control more than compromise more than f out of $N = 3f + 1$ total verification nodes following the Byzantine fault tolerance (BFT) assumptions. This threshold is based on the assumption that verification nodes are run by decentralized entities, making the cost of compromising a sufficient fraction extremely high.

2.4 Design Objectives

PlanetServe is designed to achieve the following objectives:

Performance: The key performance goals include providing **low-latency LLM services** with high scalability. In addition, PlanetServe aims to achieve **load balancing**, preventing individual nodes from being overwhelmed.

Security, privacy, and availability: PlanetServe aims to preserve user anonymity and protect the confidentiality of prompt content and model responses. It is designed to prevent model nodes or other users from linking user identities (e.g., IP addresses or public keys) to their requests. For privacy-sensitive workloads, PlanetServe leverages NVIDIA Confidential Computing on Hopper and Blackwell GPUs to run inference inside trusted execution environments (TEEs), ensuring that model nodes cannot access prompts or proprietary information. [3, 4]. In addition, PlanetServe should provide high service availability under node churn and attacks and ensure model node provide accurate responses.

3 System Design of PlanetServe

This section details the design of PlanetServe.

3.1 Overview

The overview of PlanetServe is illustrated in Fig. 1. The user nodes form a dynamic overlay network. Upon joining PlanetServe, a user node registers its current IP address and public key with the verification nodes, whose public addresses and public keys are public information. The public key serves as the identifier. It then downloads a list of active user and model nodes from a verification node, and the list is signed by more than 2/3 of the verification nodes [12]. Verification nodes may choose to divide the whole system into multiple regions and create a list of users and model nodes for each region, only when the number of users in each region is sufficiently large to hide the requester's identity, for example, > 1000 users [35, 48]. PlanetServe allows a user to use other users as relays for *anonymous communication*: each user request will be forwarded along an overlay path to hide the address of the original sending user, and the LLM reply will also be forwarded along the anonymous path back to the requesting user.

Each model node runs an instance of an LLM, which is deployed as a portable isolated environment, such as a container [39], provided by the LLM developer. One or more LLMs are deployed in the network, and each user request specifies which LLM it is requesting. A user request can be responded to by any model node that serves the same LLM as its request target. Model nodes use *overlay forwarding* to achieve load balancing and KV cache reuse among them.

The committee of verification nodes uses a recent BFT consensus protocol [11] to ensure consistency. The committee maintains a list of user nodes registered to PlanetServe, including their public keys, and a list of model nodes in PlanetServe, including their public keys and reputation scores. The reputation score of a model node quantifies the quality of its LLM responses and will be defined in Section 3.4. Verification nodes periodically request the model nodes anonymously and update their scores based on the responses.

3.2 Anonymous Overlay and Content Privacy

The anonymous overlay of users in PlanetServe allows each user node to request the LLM service without revealing its IP address. We summarize the requirements of the anonymous overlay as follows: 1) User node anonymity: other nodes cannot link a prompt with the requesting user identity. 2) Confidentiality of prompts and responses: the contents can only be visible to the requester and model node. 3) Low overhead on user nodes. In particular, we do not want to use public-key cryptography for prompts and responses, which introduces large overhead at relay nodes. 4) A single anonymous path is susceptible to relay failures under network churn. PlanetServe uses multiple paths to provide reliable message delivery.

We analyze typical overlay anonymous communication protocols [17, 25, 27, 41], including Onion routing and sliced

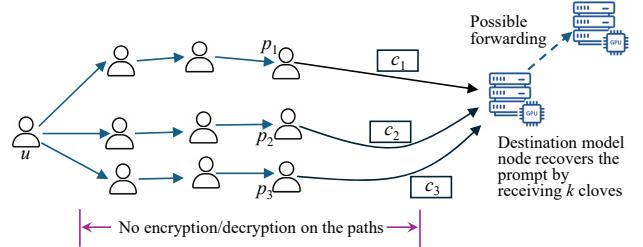


Figure 2: Process of querying the model node.

routing [27, 41], in the Appendix and find that none of them fit the above requirements. If we apply Onion to the overlay network in PlanetServe, there are a few main challenges. 1) The first layer of Onion nodes (called guards in Tor) will definitely know the sending user of a request, which should be avoided in PlanetServe. 2) The failure probability of an Onion path increases exponentially with the path length, especially for an overlay network of users with a high churn rate. 3) Each relay needs to perform public-key decryption on the entire message, introducing non-trivial overhead for users.

We propose a novel approach that achieves the advantages of both Onion and sliced routing and fits the requirements of PlanetServe exactly. In PlanetServe, each user uses Onion routing to establish n proxies. In this process, path failures and redundancy do not cause high resource waste because the establishment message is very short. After proxies are established, the user and model nodes rely on sliced routing for prompt and response messages.

Using S-IDA. We briefly present an existing protocol Secure Information Dispersal Algorithm (S-IDA) [28] that is essential to achieve message confidentiality, anonymous routing, and failure resilience in PlanetServe. Secure IDA (S-IDA) [28, 41] improves the security level of classic IDA [42] by combining it with symmetric encryption. The sender applies the following steps. 1) Encrypts M by an AES key K , getting $\{M\}_K$. 2) Splits $\{M\}_K$ into n fragments, M_1, M_2, \dots, M_n by a k -threshold Rabin's IDA. 3) Splits K into n fragments, K_1, \dots, K_n by a k -threshold Shamir's secret sharing (SSS) [46]. 4) Constructs n messages called *cloves*, C_1, C_2, \dots, C_n , where C_i includes two fields M_i and K_i . 5) Sends the cloves to the receiver using n different paths. By receiving at least k cloves, the receiver: 1) Recovers $\{M\}_K$ using Rabin's IDA and the key K by SSS. 2) Decrypts $\{M\}_K$ and gets M . Only when a node collects k cloves, it can recover the key and decrypted $\{M\}_K$. Without the complete key, it can only see the ciphertext related to $\{M\}_K$. We call the above process (n, k) S-IDA.

We present the detailed anonymous protocol.

Step 1. Preparation. A new user u contacts an arbitrary verification node to download a list of overlay users, called the *user list*, and a list of model nodes, called the *model node list*, which are signed by more than 2/3 verification nodes. Each entry in the list includes the public key and IP address. PlanetServe determines n and k ($n > k$) for (n, k) S-IDA.

2. Proxy and path establishment. Each user node, say

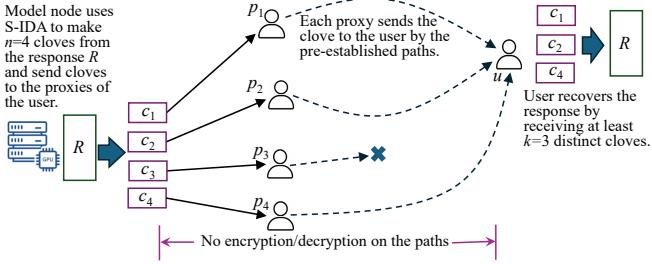


Figure 3: Process of sending the reply message from the model node to the user.

u , needs to find N proxies ($N \geq n$). To find each proxy, the user u selects l other users from the user list, where l is the path length. Conventional wisdom indicates that anonymous paths with three routers in Tor can achieve an appropriate balance between security and latency [7]. Hence PlanetServe uses $l = 3$. The user u then builds an Onion path using the public keys of these 3 selected users and uses the hash value of both u and the last user on the path to be the path session ID I . The last user p on the path becomes a proxy of u , and every node on the path stores the predecessor and successor together with the path session ID I . The above process might fail due to user dynamics but u can easily try other paths. By repeating the above process multiple times, each user u can establish N proxies using Onion paths. In later LLM prompts and responses, no public-key cryptographic operations are needed on the paths, because the relay users will use the stored path ID, predecessors, and successors for forwarding.

3. Prompt messages. When a user node u plans to send a prompt or query message Q to PlanetServe, it first selects a model node from the model node list and writes its IP as the destination. As illustrated in Fig. 2, u prepares n slices (called *cloves*) of Q using (n, k) S-IDA and sends them to n proxies respectively using the established paths. The query message Q includes only the prompt and model node IP without any information about u . The application-layer header of each clove includes the path ID I for the corresponding proxy but does not include the proxy IP. When each proxy receives the clove, it directly sends the clove to the destination model node, which is not anonymous. The prompt message also includes the IP addresses of at least n proxies of u , which are revealed to the model nodes when it receives $\geq k$ cloves.

4. Response messages. As shown in Fig. 3, each response message is made into n cloves using (n, k) S-IDA. Then the model node sends n cloves to n proxies of u respectively. Each proxy then forwards the received clove to u using the pre-established anonymous path. When u receives more than k cloves, it can decrypt the reply message using S-IDA. The content of the response is confidential unless an attacker controls users that appear on k or more of these n paths. Even in that case, the attacker needs to use brute force to decode S-IDA because these n paths have different IDs.

Content privacy. PlanetServe supports model nodes with Confidential Computing (CC) hardware, including NVIDIA

Hopper and Blackwell GPUs such as H100 [3, 4]. This will allow users to choose between two privacy tiers: a default user-anonymous mode as stated above, and an optional content-privacy mode, in which both the identity and the prompt content remain protected from the node operator. For sensitive scenarios such as resume editing, healthcare, or finance, users can opt for content anonymity. Achieving content privacy protection requires hardware-based Trusted Execution Environments (TEEs) because software-only encryption can protect data in transit but not in use—once plaintext resides in GPU memory for inference, it is otherwise accessible to the host OS or hypervisor. The H100 CC mode addresses this by booting in a verified state with an on-die hardware root of trust, ensuring only NVIDIA-signed firmware is loaded, and by running inference inside an isolated GPU TEE in which all PCIe/NVLink traffic is encrypted with AES-GCM-256 and rolling IVs to prevent replay attacks. Data is exchanged with the CPU TEE (e.g., AMD SEV-SNP or Intel TDX) only through encrypted bounce buffers. Hence, plaintext is never exposed outside the enclave. Before processing, the GPU can be remotely attested via NVIDIA’s attestation service to verify its identity, firmware integrity, and CC configuration. Each LLM is launched in a Confidential VM (CVM) with the signatures of the verification committee, using the CC mode. User nodes establish end-to-end TLS sessions directly with the CVM, ensuring that neither the host nor the hypervisor can observe the prompt content. We show the performance results via implementation in Sec. 5.3.

3.3 Overlay Forwarding among Model Nodes

All model nodes serving the same LLM form a logical group. Upon receiving a request from a user, a model node can serve this request or forward the request to another node. The forwarding logic, illustrated in Fig. 4, is designed to consider both KV cache reusing and load balancing objectives. The detailed design includes the following components.

Hash-Radix tree. Recent studies [13, 50, 59] show that sharing and reusing KV cache among different GPUs can significantly reduce the average latency for LLM requests. However, prior work such as SGLang [59] and Preble [50] assumes a cloud environment that has a centralized scheduler or request router. This central entity receives all requests and distributes them to the appropriate GPUs. PlanetServe lacks such a central coordinator. The centralized scheduler [50, 59] uses a radix tree [37] for searching existing KV caches that can be reused for the current prompt, based on matching prefixes. Each tree node in a radix tree represents a prefix.

In a decentralized system like PlanetServe, naively requiring every model node to maintain a full copy of the radix tree built from the KV cache information of all model nodes in the group introduces significant scalability problems: 1) High memory overhead: each node needs to spend substantial memory space to store the aggregated radix tree, including the metadata for the KV cache from all other nodes; and 2)

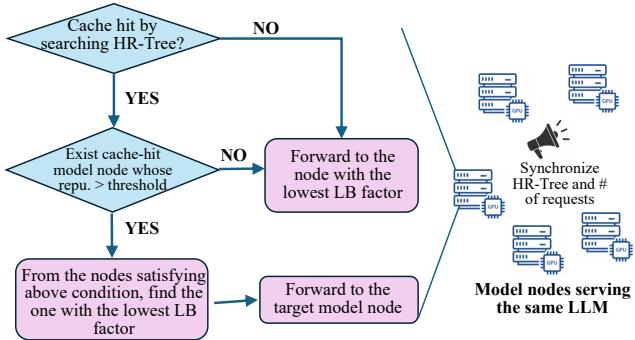


Figure 4: The logic of overlay forwarding

Req 1 The weather is nice today. Let us go surfing!
0x0A 0x8B 0x00 0x71
Req 2 The weather is bad today. The ceiling is leaking.
0x0A 0x5C 0x89 0x21
Req 3 The weather is nice today. Hope we can have a nice trip!
0x0A 0x8B 0x54 0x77

Figure 5: Pre-processing for an HR-tree

high communication overhead: each node needs to broadcast updates about the prefixes of its local KV cache to all other nodes in the group. To resolve these problems, we propose a new distributed data structure called a Hash-Radix tree (HR-tree), detailed below.

The design of the HR-tree is motivated by cuckoo filters [21], which replace hash table entries with fingerprints to reduce memory cost. An HR-tree represents the aggregated KV cache state from *all overlay model nodes in the group*. A request prompt can be efficiently searched in the HR-tree. If the prompt matches a *sufficiently long* prefix path in the HR-tree, the HR-tree returns the model node that holds the KV cache corresponding to the matched prefix. Otherwise, the HR-tree reports no match.

The HR-tree construction involves a pre-processing step, shown in Fig. 5, where each prompt is divided into variable-length chunks. Each chunk is converted into a hash value computed by a universal hash function H. The example uses 8-bit hash values such as 0x0A and 0x8B. The length of each chunk i in the number of tokens, l_i , is stored in an array L that is computed by a deterministic algorithm called Sentry, detailed in the Appendix. The HR-tree is then built based on the sequence of hash values, as shown in Fig. 6. Each tree node stores the hash value of a chunk. Two sibling nodes indicate that their corresponding prompts share the same prefix up to the parent node but differ thereafter.

When each model node receives a user request that includes a prompt, it searches the prompt in the HR-tree. The search algorithm first applies pre-processing to divide the query prompt into chunks and compute their hashes. The algorithm then traverses a tree structure, starting from the root node. For each chunk of the querying prompt, the algorithm attempts to find a child node with the same hash value as the chunk. When there is no child node matching the hash value of the next chunk, the search stops. The HR-tree returns

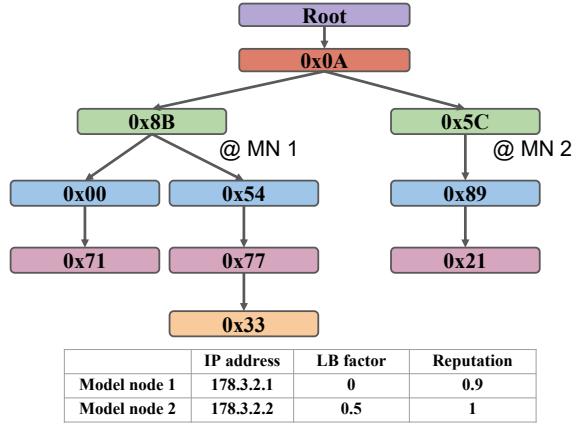


Figure 6: The structure of a Hash-Radix tree

the search depth d . If d exceeds a threshold, we consider it a match. Each tree node also includes a set of pointers, each referencing an entry of a separate table. The entry represents a model node that holds the corresponding KV cache of the prefix. Each entry in the table includes the node’s IP, load balance factor, and reputation score.

Similar to Bloom and cuckoo filters [8, 21], an HR-tree might also introduce *false positives*: a prompt that does not match a prefix can be judged as a match by the HR-tree. For a hash value with 8 bits, the probability of a hash collision is 1/256. However, a false positive requires matching d tree nodes, so the false positive rate is as small as $1/256^d$.

State synchronization. For each model node in a group, it periodically broadcasts the local updates of its HR-tree. Each node keeps a recent snapshot of its HR-tree and the following updates after the snapshot. The node periodically sends a minimal but necessary update to all nodes in the domain.

Load balancing is another design consideration in addition to KV cache reusing. The main objectives of load balancing are to ensure that no model node receives more requests than its capacity, and every model node can serve a certain number of requests to establish a reputation score. In the forwarding logic shown in Fig. 4, when selecting the target model nodes from a set of candidates, we always select the one with the lowest load balance (LB) factor, which is defined as $F_{LB} = L * (Q/C)$, where L is the moving average of the latency to serve each request on that node, Q is the number of queued requests on that node that are waiting for serving, C is the capacity of the number of concurrent requests that can be served on the node. The moving average is calculated similarly to the RTT estimation with parameter $\alpha = 1/8$. The LB factor is calculated locally by each node and audited by the verification committee. Each node periodically broadcasts its LB factor to other model nodes.

Forwarding logic. The overlay forwarding logic, executed by **every model node** upon receiving a user request, determines how to forward the request, as illustrated in Fig. 4. The model node first searches the request’s prompt in its HR-tree.

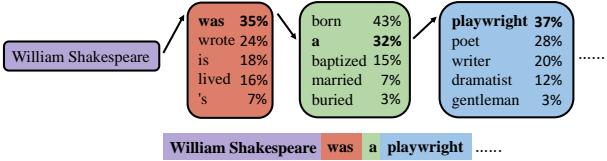


Figure 7: Verification for the model node challenge response.

This search yields either a match or a miss. If the search result is a miss, the node prioritizes load balancing and forwards the request to the model node with the lowest LB factor. If the search result is a match, the HR-tree identifies a list of model nodes that hold reusable KV caches. If there is one or more model nodes left, we pick the one with the lowest LB factor and forward the current request to it.

Session affinity for consecutive prompts. Every model node includes its IP address in the response message. For subsequent follow-up prompts from the same user session, the user attempts to direct these prompts specifically to that designated model node, routing them via the anonymous overlay. This policy prioritizes KV cache reuse for consecutive prompts within a session.

3.4 Model verification

The committee of verification nodes is responsible for ensuring that the model nodes correctly run the models they promise to serve by maintaining their reputation scores. The verification nodes run a BFT-like consensus protocol [11] to ensure data consistency and authenticity in the committee. The fundamental verification check is to compare a response from a target model node for a specific prompt against a reference result generated locally using the same LLM. However, verifying every user response is impractical due to scalability and potential user privacy issues.

PlanetServe introduces a sampling-based verification strategy. Instead of monitoring all responses, verification nodes periodically send challenge prompts to model nodes and verify their responses. These challenge prompts are routed by the anonymous overlay; hence, **model nodes cannot distinguish verification probes from regular user requests**. Verification nodes then assign each model node a reputation score (defined later) based on the quality of their response. This allows the system to evaluate the model node’s behavior probabilistically without compromising efficiency or privacy.

Each verification node locally deploys the same copy of the LLM as the model nodes. When receiving a challenge response from the model node, verification nodes evaluate it against their local reference result, which is not necessarily the same. The underlying principle is that, when using the same model and prompt, the conditional probability distribution over the next token is well-defined [20]. Suppose a verification node evaluates the correctness of a model node’s response r containing n tokens, $r = (t_1, t_2, \dots, t_n)$, to a challenge prompt. The verification node assesses the output token-by-token, as illustrated in Fig. 7. For each token t_k at position

k : 1) The verification node takes the prefix of the received response up to token t_k and the original prompt. It uses its local LLM, conditioned on this input, to compute the probability distribution over all possible tokens at position $k + 1$. 2) It then looks up the probability assigned by its local model to the actual token t_{k+1} from the target model node. This process repeats for all tokens, generating a list of probabilities.

The verification node calculates a score for each challenge response by computing its perplexity (PPL) based on the sequence of probabilities $PPL = \exp\left(-\frac{1}{n} \sum_{i=1}^n \log p(t_i | t_{<i})\right)$. The normalized perplexity $1/PPL$ is used for a single challenge. By evaluating multiple challenges, the verification node obtains an average score $C(T)$ during time epoch T .

The reputations score of a model node at time epoch T , $R(T)$, is updated by a *moving average*:

$$R(T) = \alpha \cdot R(T-1) + \beta \cdot C(T)$$

where $R(T-1)$ is the score from the last epoch, $C(T)$ is the average score from the challenges in T , and α and β are the weights for the moving average. In our implementation, we set $\alpha = 0.4$ and $\beta = 0.6$.

The punishment to the reputation for a low score should be *much stronger* than the reward for a high score. This ensures the model node has to respond correctly consistently to earn a high reputation. We use a *sliding window approach* to apply punishment to low values of $C(T)$. The verification nodes maintain a sliding window of each model including the past W values of $C(T)$ —the window size W is set to 5 in our implementation. If $C(T)$ is smaller than a threshold τ , it is considered an abnormal value. Let the count of abnormal values in the past W be c . If c/w is above a predefined threshold γ (such as 1/5), the punishment applies as follows:

$$R(T) = \alpha \cdot R(T-1) + \frac{W+1}{W+c/\gamma+2} \cdot C(T)$$

Therefore, the reputation decreases excessively when detecting abnormal values. If a model node’s reputation drops below a critical level (like 0.4), it will be marked as untrusted.

Verification is performed epoch-by-epoch. In each epoch, the committee collaboratively assesses the correctness of model nodes and updates their reputation accordingly. PlanetServe relies on an open-source consensus protocol Tendermint [10] to ensure safety and liveness under partial synchrony. Each update message should be signed by at least $2n/3 + 1$ nodes before commitment. In each epoch, a verification node is selected as the leader to send challenge requests to model nodes. To reduce overwhelming model nodes, only the leader of each epoch communicates with the model nodes, collects their responses, and broadcasts the results to the committee. Each verification node then locally verifies the results independently and votes on the updates.

The leader L_i of the epoch e_i is selected pseudo-randomly and verifiably towards the end of the previous epoch. Specifically, we use a Verifiable Random Function to select L_i based on the final commit hash of epoch e_{i-1} . At the end of epoch

e_{i-1} , the committee also agrees on the set of model nodes to be verified in epoch e_i , M_i , and the corresponding challenge prompts for each of them. This is prepared by the leader L_{i-1} in this epoch. The prompt is a unique, random natural text question, indistinguishable from normal user prompts. No two model nodes should be asked the same prompt to prevent collusion or replay attacks. This pre-determined model node and prompt list is to prevent the leader L_i from behaving maliciously in epoch e_i by selectively ignoring certain model nodes or assigning inconsistent prompts.

In epoch e_i , the leader L_i initiates the verification process by sending the pre-determined challenge prompts to each model node in M_i . Each model node receiving a prompt generates a response output, which is a sequence of output tokens. The model node cannot tell if this is a prompt from clients or verification nodes, because all requests are delivered through the anonymous overlay. Once responses are received, L_i computes a credibility score for each model node and generates a reputation score accordingly using the above method. L_i then broadcasts the signed challenge responses list and reputation scores to the committee for validation.

If the response and digital signature do not match or the model node does not respond, the leader can include an “invalid response from model node x ” message in the broadcast message, but does not reduce x ’s reputation. It is to prevent the leader from attacking a model node by dropping or altering the response. Only when more than $1/3$ of the committee reported invalid responses for the same model node, implying that x misbehaves on purpose, x ’s reputation will be reduced.

Upon receiving the information from the leader, each verification node performs an integrity check locally. It verifies whether the list of model nodes M_i and associated prompts match the pre-determined prompt list, and checks the signatures and timestamps of each model node response. If any inconsistency is detected - e.g., different prompts or wrong signatures - verification for the current epoch is aborted. A new leader will be selected for the next epoch. For valid challenge responses, each verification node independently computes the credibility and reputation score using its local LLM, and compares it with the score proposed by the leader L_i . The reputation update follows Tendermint’s two-phase voting protocol: Pre-Vote and Pre-Commit. If they match with negligible variance, the verification node sends a pre-vote.

4 Security Analysis and Evaluation

4.1 Anonymity

We apply a commonly used entropy-based metric [41, 61] to measure the anonymity of a system, whose definition is presented in the Appendix. It ranges from 0 and 1, and a higher value represents better anonymity.

We use simulations to compare the normalized entropy across three systems: onion routing [17, 43], garlic cast (GC) [41], and PlanetServe, in a 10,000-node network under different fractions of malicious nodes. As shown Fig. 8, Plan-

etServe consistently achieves the highest level of anonymity, particularly in low-to-moderate corruption scenarios. For instance, at a corruption fraction of $f = 0.05$, the normalized entropy for PlanetServe reaches 0.965, compared to 0.954 for onion and 0.903 for GC. While introducing path diversity enhances resilience to individual path failures, it simultaneously increases the risk of de-anonymization if adversaries are capable of cross-path collusion. However, PlanetServe uses different path IDs for different paths, hence achieving greater robustness against colluding adversaries.

4.2 Message confidentiality

The content of a prompt or response is revealed to adversaries if the colluding adversaries exist on more than k paths of the user node, where k is the decoding threshold for S-IDA [28, 44, 51]. Even if adversaries control k paths, exhaustive brute-force decoding over all possible combinations of forwarded messages remains computationally prohibitive. In Fig. 9, both PlanetServe and GC show degradation in confidentiality if brute-force decoding is possible – a big assumption on attack’s capability – with GC dropping to 0.73 and PlanetServe to 0.88 under high adversarial presence ($f = 10\%$). If brute-force decoding is impossible, both systems maintain almost perfect confidentiality across all scenarios.

4.3 Detecting Dishonest Model Nodes

We focus on detecting dishonest model nodes that use low-quality models or forged responses. Note that a model node cannot differentiate the requests from normal users and those from verification nodes, because all requests are routed through the anonymous overlay.

We use Meta-Llama-3.1-8B-Instruct-Q4_0 as the ground truth (GT) model for evaluation, representing a reference response. To simulate dishonest or degraded behavior, we introduce four alternative models with lower capabilities or altered configurations. Specifically, **m1** refers to Llama-3.2-3B-Instruct-Q4_K_M, **m2** refers to Llama-3.2-1B-Instruct-Q4_K_M, **m3** refers to Llama-3.2-1B-Instruct-Q4_K_S, and **m4** refers to Llama-3.2-3B-Instruct-Q4_K_S. These models exhibit varying levels of performance degradation due to either reduced model size or quantization settings, and serve to evaluate the sensitivity and discriminative power of the Planet-Serve verification and reputation framework. In addition, we include two further settings based on the ground-truth model but with deliberate prompt alterations. **gt_cb**, where prompts are rewritten into clickbait-style headlines, and **gt_ic**, where prompts are followed by injected long-form continuations spanning diverse genres (e.g., fictional expansions, documentary-like prose, or pop-psychology explanations).

We first run 50 prompts and evaluate the responses from all five models by measuring the credit scores in the normalized perplexity. Fig. 10 shows normalized perplexity-based credibility scores. Each point corresponds to the score of a single reply. We find that statistically, the ground truth model (GT)

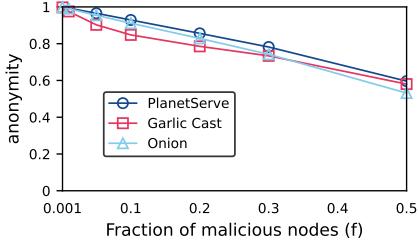


Figure 8: Anon. vs. malicious frac.

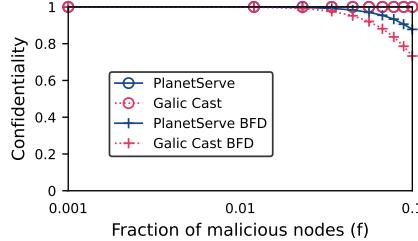


Figure 9: Conf. vs. malicious frac.

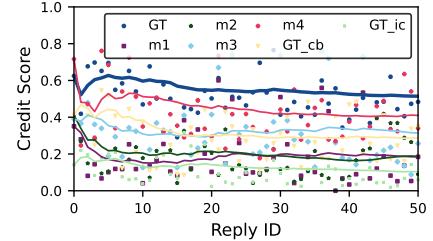


Figure 10: credit score over models.

provides higher scores than other models. A false positive is possible, but in the long term, it will be corrected if the node follows the protocol and its credit score converges to the correct value. Fig. 11 illustrates how the reputation scores evolve over time across 35 epochs (50 prompts each) by varying three penalty sensitivity levels: $\gamma = 1, \frac{1}{3}, \text{ and } \frac{1}{5}$. The results show a clear separation between GT and the weaker models after the first epoch, validating the effectiveness of the token-level probabilistic verification strategy. Fig. 11a shows the reputation trajectories under Level-1, where the penalty is relatively lenient. In this setting, the dishonest models experience slower degradation and tend to stabilize around reputation scores of 0.2 - 0.4, which may allow them to remain in the trusted pool for longer periods. With a stricter penalty in $\gamma = \frac{1}{3}$ Fig. 11b, reputation scores of dishonest models exhibit a more significant decline, often dropping below 0.2 by period 5. For the most aggressive punishment threshold $\gamma = \frac{1}{5}$ in Fig. 11c, results in an early and sharp decline of reputation scores, with most dishonest models falling below 0.1 within the first 5 periods. Although the paper only shows a few set of experiments, we have run many similar experiments and PlanetServe can detect weaker models in every set of experiments.

Based on the empirical results, we set the reputation threshold to detect a dishonest model as 0.4 and $\gamma = \frac{1}{5}$.

4.4 Other Security Properties

Denial-of-Service (DoS) attacks. We consider two scenarios of DoS attacks. 1) A malicious verification node refuses to perform the verification task, either acting as the leader for an epoch or as a member. If the leader is malicious and refuses to send challenge prompts to model nodes or broadcasts challenge responses to the committee, the system simply fails the current epoch and restores the verification process in the next epoch with a new leader. If some committee members refuse to verify and vote, the system can still rely on the remaining honest verification nodes to vote and reach a consensus, which is guaranteed by BFT. 2) A model node does not respond to user requests. This behavior can be detected by the verification process.

Counterfeiting. There are three main types of counterfeiting attacks by a malicious verification leader. 1) The leader attempts to degrade some model nodes by sending them modified challenge prompts different from the agreed prompt list. Since model node responses always include the original

prompt, other verification nodes can detect this deviation. 2) The leader alters the responses from the model nodes before broadcasting them to the committee. In PlanetServe, each response from a model node includes its digital signature, which ensures the integrity and authenticity of the responses. Any tampering by the malicious leader would be easily detected. 3) The leader can intentionally skip broadcasting responses from certain model nodes or falsely claim that a model node returned an “invalid response”. To prevent this, all verification nodes reach consensus in advance on the list of model nodes to be challenged for the epoch. If verification nodes detect that some expected responses are missing or marked as invalid without justification, they will initiate their own challenge prompts to those model nodes, which are distinct from the original prompts used by the leader, to prevent auditing detection by the model nodes. If more than 2/3 of the verification nodes receive valid responses from the model nodes, the system will identify the leader as malicious.

Sybil attacks. An adversary could potentially leverage Sybil attacks to control multiple verification nodes to alter verification results. Verification nodes are operated by organizations or individuals with verifiable credentials (e.g., recognized academic or research institutions). These credentials will be periodically checked and revoked if a verification node is compromised, and then the node will be expelled from the committee. Moreover, verification nodes are selected to be geographically and administratively diverse, which makes it hard for an adversary to compromise $\geq 1/3$ of the committee. To further improve the security of the verification committee, the committee members can be non-static. They are periodically rotated and re-elected. Misbehaving verification nodes will be detected and removed. This could increase the cost and difficulty for the adversary to compromise the committee.

5 Performance Evaluation

5.1 Methodology

We deploy the testbed of PlanetServe in a public cloud. The testbed includes eight model nodes with mid-tier hardware, each carrying one NVIDIA RTX A6000 GPU (48 GB) and serving a Meta-Llama-3 8B model, eight other model nodes with high-performance hardware, each carrying one NVIDIA A100 GPU (80 GB) and serving a DeepSeek-R1-Qwen-14B model. We run two machines as the verification nodes. One carries an NVIDIA A100 (40 GB SXM4), 30 virtual CPUs,

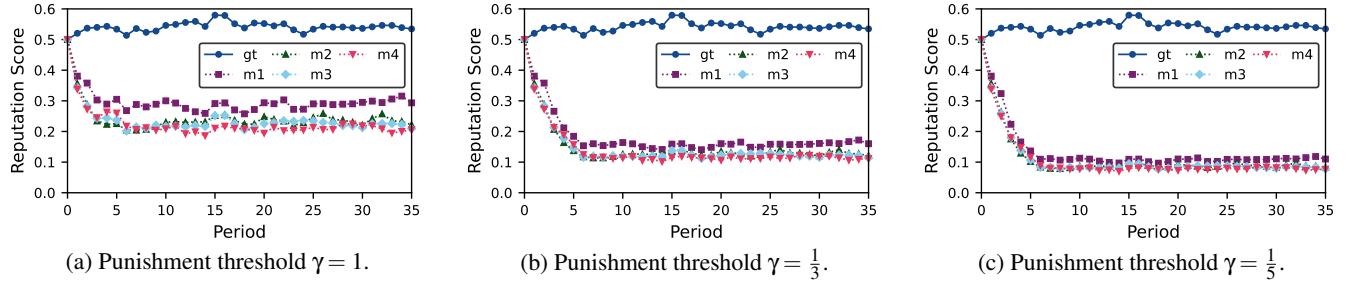
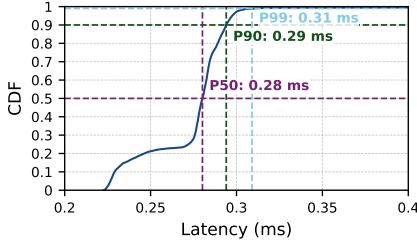
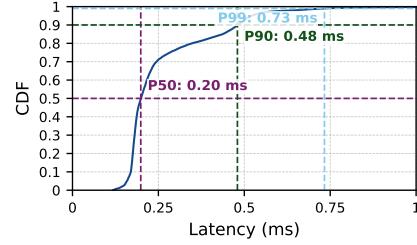


Figure 11: Reputation changes with different punishment levels.



(a) Preparation latency



(b) Decryption latency

Figure 12: Latency of preparing cloves on model nodes and decrypting cloves on user nodes.

200 GiB of RAM, and a 0.5 TiB SSD, and the other carries an NVIDIA GH200 (96 GB HBM), 64 virtual CPUs, 432 GiB of RAM, and 4 TiB SSD. There is no centralized coordinator in the testbed. Internet latency in simulations is from samples collected by our measurement. We use the most up-to-date version of vLLM [29, 53], a recently developed LLM serving library, on each single model node.

Each user request is encoded into 4 cloves routed through 3-hop paths, with successful recovery requiring at least 3 cloves ($k = 3$). 8-bit hash values are used for the HR-tree to optimize the balance between memory footprint and false-positive rate. The chunk-length arrays L are refreshed every 10,000 requests, and HR-tree state synchronization occurs every 5 seconds. Queries are dispatched according to a Poisson distribution with varied mean inter-arrival times, accurately simulating real-world user query patterns and request bursts commonly observed in production environments.

The evaluation includes four realistic workloads:

1. **ToolUse (ToolBench [23]):** .210k queries with tool-specific instructions; prompts average 7,206 tokens after Zipf-1.1 sampling. Prefix sharing is moderate, and outputs are capped at 100 tokens per query for the models we evaluate.

2. **Coding (APPS [24]):** .10,000 coding problems with detailed solution requests; prompts average 1,802 tokens after Zipf-0.8 sampling. Prefix overlap is minimal, and outputs are capped at 1,000 tokens per query for the models we evaluate.

3. **Long-Doc QA (LooGLE [32]):** 776 long documents paired with 6.4k questions. Each prompt includes a long document as prefix followed by a question, averaging 10,985 tokens per prompt after Zipf-0.6 sampling. Outputs are capped at 100 tokens per query for the models we evaluate.

4. Mixed Workload: It integrates the ToolUse, Coding, and Long-Doc QA workloads in a 3:6:1 ratio per real-world traces [50, 56], averaging 9,959 tokens per prompts (All token counts above are computed using the Llama 3 tokenizer).

Upon acceptance of this paper, we will make the prototype and data sets open-sourced.

5.2 Anonymous Routing

We evaluate the performance overhead of anonymous message (clove) processing using the Toolbench dataset [23]. Clove preparation is performed on a model node equipped with a single NVIDIA A100 40GB SXM4 GPU, 30 vCPUs, 200 GiB RAM, and 0.5 TiB SSD running Llama-3.3-70B. Decryption is executed on a user node with a single Intel i7-7700 CPU @ 3.60GHz. Figure 12 shows the cumulative distribution functions (CDFs) for both preparation and decryption latency over 10,000 trials.

Clove preparation is computationally lightweight, with a mean latency of 0.273 ms and a median (P50) latency of 0.280 ms. Even at the P99 percentile, latency remains below 0.31 ms, demonstrating tight performance bounds. Clove decryption is similarly efficient, with a slightly higher mean latency of 0.302 ms and a P50 of 0.309 ms. The P99 latency is 0.334 ms, and the success rate remains 100%, confirming the reliability of the decryption process. Overall, both operations exhibit low and stable latency.

We conduct simulations to evaluate the communication survival rate under network churn in a 3,119-node network and a churn rate of 200 nodes/min – a very high rate. We deployed 14 AWS t3.micro instances across multiple regions and measured end-to-end routing latency with realistic prompt

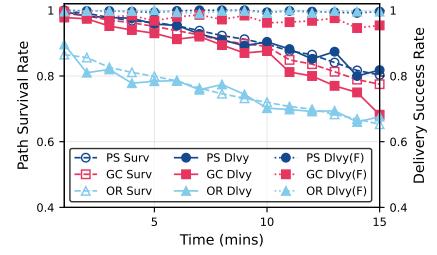


Figure 13: Survival probability and delivery rate under churn (200 nodes/min).

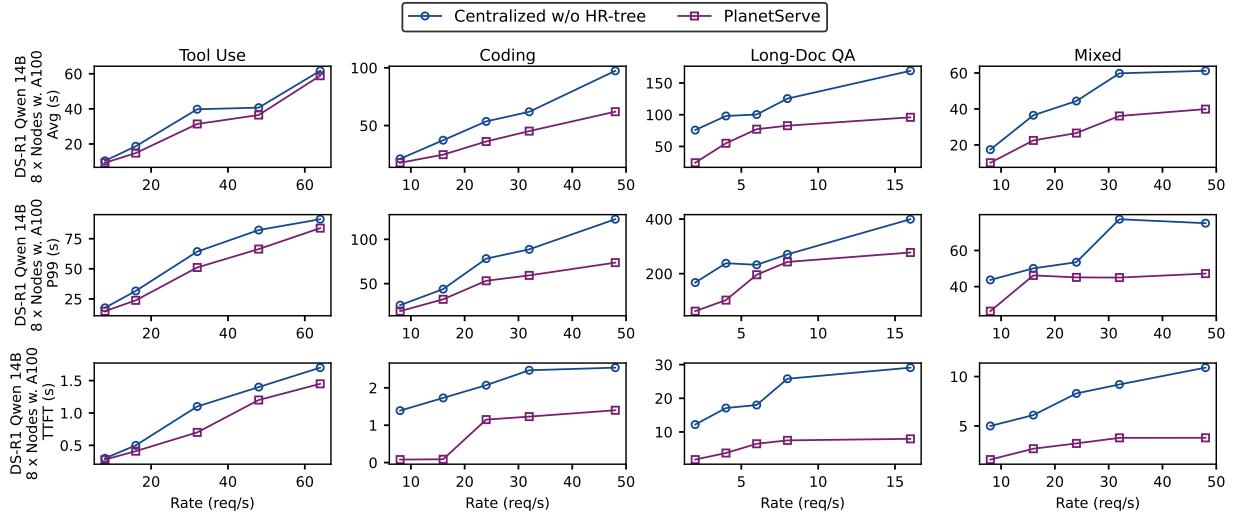


Figure 14: Latency of LLM serving w/ and w/o HR-Tree.

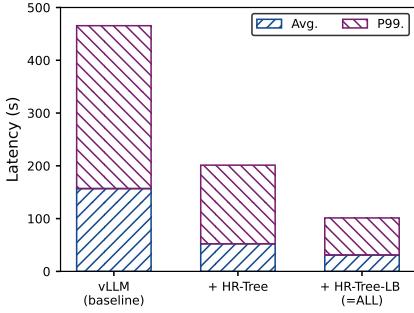


Figure 15: Ablation study of latency.

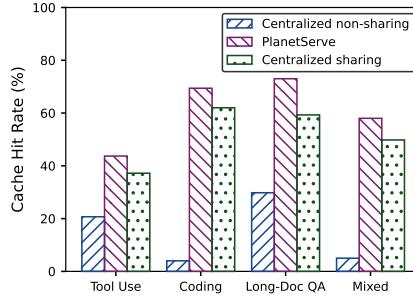


Figure 16: Cache hit rate.

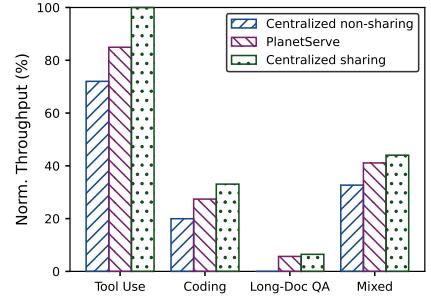


Figure 17: Norm. LLM throughput

model	mean		P99	
	CC-on	CC-off	CC-on	CC-off
Llama-3.1 8B	132.19	130.95	123.94	122.70
DS-R1-Q 14B	211.58	210.96	197.9527	195.77

Table 1: Latency comparison under CC mode.

payloads over 4,000 runs (3 nodes randomly fail in the USA; 1 randomly fails worldwide). We incorporate latency, link failures, packet loss, and congestion to the simulation and show the results in Fig. 13. The path survival rate of PlanetServe is the highest among these methods. When there is no failure, all methods have high delivery rates. However, when failure occurs, only PlanetServe can maintain a similar high delivery rate. The delivery rate of Onion significantly decreases.

5.3 Latency of Confidential Computing (CC)

In Table 1, we show the results of the experiments on an Azure Standard NCC40ads H100 v5 virtual machine (Confidential VM), equipped with a single NVIDIA H100 GPU, 40 vCPUs, and 320 GB RAM, for the CC-on setting. For the CC-off setting, we used the corresponding Standard NC40ads H100 v5 VM, which provides the same hardware resources but does

not enable confidential computing features. In both settings, the request rate was fixed at 20 requests per second. The results show that CC adds very small latency.

5.4 Overlay Forwarding

We evaluate three types of latency of the model node responding to user requests: average generation latency capped at 1000 tokens (Avg), 99th percentile (P99), and Time to First Token (TTFT). The LLMs used in the evaluations are Meta Llama 3 (8B) running on A6000 and DeepSeek-R1 (14B) running on A100. We compare PlanetServe with a centralized baseline of 8 A100 GPUs without KV-cache sharing. Each data point reported here is the average of three independent runs of the entire workload. Latency results are shown in Fig. 14 for PlanetServe and the centralized baseline. PlanetServe consistently demonstrates significant latency improvements across all evaluated workloads, LLM, and GPU setups. Notably, under heavy workload (LooGLE and Mixed), the difference is more evident, indicating the effectiveness of PlanetServe’s overlay forwarding and cache-reuse strategy. The TTFT metric directly impacts perceived user responsiveness, a critical factor in practical deployments. PlanetServe pro-

vides significantly shorter latency in TTFT. For both LLMs, PlanetServe reduces TTFT at higher request rates by 40-50%. In addition, for higher request rates, PlanetServe maintains slower latency growth.

To better understand the benefits of different design components, we conduct an ablation study by incrementally enabling the HR-tree and load balancing feature using 8 A100 as 8 model nodes running Llama-3.1-8B. The results are shown in Fig. 15 with the ToolUse workload and a Zipf-1.1 distribution. We find that using the HR-tree almost reduces both the average and P99 latency by more than half, and adding load balancing further reduces latency.

We show the KV cache hit rates of the mixed workload on DeepSeek-R1-Qwen-14B model nodes in Fig. 16. In addition to the centralized baseline, we further compare PlanetServe to centralized serving with 8x A100 GPUs, running vLLM with the default continuous-batching scheduler and tensor parallelism. PlanetServe consistently achieves significantly higher cache hit rates across all evaluated workloads. The throughput comparison is shown in Fig. 17. PlanetServe outperforms the baseline without sharing, and tensor parallelism provides the highest throughput due to the scheduler and parallelism.

5.5 Verification throughput

Assuming each model node requires verification 50 times per day and a deployment ratio of up to 100 model nodes per VN, the required throughput is 208 verifications per verification node per hour. We evaluate the verification nodes on two different hardware platforms. On a high-end setup equipped with an NVIDIA GH200 (96 GB HBM), 64 virtual CPUs, 432 GiB of RAM, and 4 TiB SSD, a node achieves a throughput of 45.04 verifications per minute. On a standard configuration featuring an NVIDIA A100 (40 GB SXM4), 30 virtual CPUs, 200 GiB of RAM, and a 0.5 TiB SSD, the verification rate is 20.72 verifications per minute. Both hardware platforms are sufficient to complete the verification tasks.

6 Discussion

Lesson learned from past projects. We investigate the challenges of the past decentralized overlay networks, such as PlanetLab and P2P file sharing, and discuss how the design of PlanetServe can learn from these projects. 1) Incentive design is important. Performance efficiency alone does not necessarily lead to adoption. Hence, we design a comprehensive incentive infrastructure with verification services. We plan to keep improving this infrastructure. 2) Churn-resilience and security should be considered at the beginning of deployment, which are the main research objectives of this work. 3) Integration with modern computing architecture. Some past P2P networks cannot be integrated with the cloud architecture, which significantly limits their applications. The design of PlanetServe aims at a flexible and scalable architecture that is compatible with cloud and can evolve with new technologies.

More complex models. This paper focuses on text-only tasks. Future research will extend the architecture to sup-

port more complex models. A key direction is serving multimodal models including requests with non-textual data. In fact, HR-Tree provides advantages in supporting non-textual data compared to existing KV cache sharing methods, because it records the hash values rather than the specific texts. Another future research direction is to support advanced reasoning and tool-using models. The workloads of these models could be different from the current ones, requiring adaptation of PlanetServe’s current KV cache sharing and load-balancing mechanisms. However, the main infrastructure of PlanetServe can remain the same. New optimization methods can be deployed in it incrementally.

Model partitioning. Due to the limited size of a research paper, we do not consider model partitioning [9], which can be complementary to PlanetServe. The new combined research objective, i.e., how to achieve decentralized overlay LLM serving with model partitioning, can be future research.

Blockchain integration. Our committee-based BFT mechanism provides a lightweight and efficient solution for serving and verification without incurring the additional overhead of deploying blockchain. However, blockchain offers desirable properties such as transparency, auditability, and simplified global coordination. In particular, smart contracts could automate reputation management in a transparent and tamper-resistant manner. Thus, We will integrate blockchain in the future to enhance transparency and auditability as the system scales to support a larger number of users and model nodes.

7 Related Works

With the growing importance and popularity of LLM applications, LLM serving engines, software frameworks to run LLM inferences efficiently, have been studied and developed [19, 29–31, 33, 34, 38, 40, 47, 49, 50, 55, 59, 60]. One key challenge of an LLM serving engine is the management of KV cache. For example, vLLM [29] stores KV cache as non-continuous blocks, each with a fixed number of tokens. It enables sharing KV caches for different user requests with similar prompts.

Distributed LLM serving studies the management of a group of serving nodes in a cloud/cluster environment [22, 26, 34, 50, 59]. One key research problem is to distribute requests to different serving nodes [26, 50, 59]. SGLang [59] and Preble [50] use a radix tree to find reusable KV caches across different user requests and implement scheduling algorithms to achieve load balancing. DistServe [60] assigns prefill and decoding to different nodes to eliminate their interferences. These studies assume a centralized “scheduler” or “router” that analyzes all user requests based on various scheduling algorithms and sends the requests to serving nodes. In contrast, there is no such central scheduler in PlanetServe with knowledge of the entire system. PlanetServe is the first study of the three key problems in decentralized serving.

Petals [9] proposes to run an LLM (such as BLOOM-176B) on multiple consumer GPUs collaboratively. The Knowledge

Delivery Network (KDN) [14] allows storing and exchanging KV caches among different serving engines. DiLoCo [18] uses distributed clusters for training. Orca [58] uses a centralized scheduling algorithm to improve serving latency and throughput. None of them address the challenges discussed in PlanetServe, including decentralized serving and verification, overlay forwarding, and anonymous communication.

8 Conclusion

This work introduces PlanetServe, a novel decentralized infrastructure designed for scalable and privacy-preserving LLM serving. We present designs addressing four key challenges: overlay network organization, anonymous communication, overlay forwarding for efficiency, and verification of model serving quality. These issues are fundamental and inherent to any future decentralized LLM-serving system. We implement a prototype of PlanetServe and evaluation results show that PlanetServe achieves low serving latency with minimal overhead for security and trustworthy features. In the future, we plan to deploy PlanetServe at scale and encourage broader community engagement in the research and use of PlanetServe. We believe this work offers a new direction that will attract further research and eventually help democratize future AI innovations and deployment.

References

- [1] ScaleLLM: Unlocking Llama2-13B LLM Inference on Consumer GPU RTX 4090, powered by FEDML Nexus AI, 2023. <https://blog.tensoropera.ai/scalellm-unlocking-llama2-13b-llm-inference-on-consumer-gpu-rtx-4090-powered-by-fedml-nexus-ai/>.
- [2] 2025. <https://github.com/Aider-AI/aider/issues/3005>.
- [3] Confidential Compute on NVIDIA Hopper H100. Technical report, NVIDIA Corporation, 2025.
- [4] NVIDIA Secure AI with Blackwell and Hopper GPUs. Technical report, NVIDIA Corporation, 2025.
- [5] Opinion: Data centers are eating the economy — and we’re not even using them, 2025. <https://www.sltrib.com/opinion/2025/08/11/opinion-data-centers-are-eating/>.
- [6] Jeanine Banks and Tris Warkentin. Gemma: Introducing new state-of-the-art open models, 2024. <https://blog.google/technology/developers/gemma-open-models/>.
- [7] Kevin Bauer, Joshua Juen, Nikita Borisov, Dirk Grunwald, Douglas Sicker, and Damon McCoy. On the optimal path length for tor. In *Proc. of HotPETs*, 2010.
- [8] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 1970.
- [9] Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. Petals: Collaborative inference and fine-tuning of large models, 2023.
- [10] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.
- [11] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *arXiv preprint arXiv:1807.04938*, 2019.
- [12] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proc. of USENIX OSDI*, 1999.
- [13] Yihua Cheng. A Scalable Approach to Distributed Large Language Model Inference, 2025.
- [14] Yihua Cheng, Kuntai Du, Jiayi Yao, and Junchen Jiang. Do large language models need a content delivery network? *arXiv preprint arXiv:2409.13761*, 2024.
- [15] Brent Chun et al. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 2003.
- [16] DeepSeek-AI. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [17] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proc. of USENIX Security 2004*, 2004.
- [18] Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc’Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. Diloco: Distributed low-communication training of language models, 2024.
- [19] Hugging Face. Large language model text generation inference, 2025. <https://github.com/huggingface/text-generation-inference>.
- [20] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018.
- [21] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proc. of CoNEXT*, 2014.
- [22] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. Serverlessllm: low-latency serverless inference for large language models. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, 2024.

- [23] Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models, 2024.
- [24] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps. *NeurIPS*, 2021.
- [25] Hsu-Chun Hsiao et al. Lap: Lightweight anonymity and privacy. In *Proc. of IEEE S&P*, 2012.
- [26] Kunal Jain, Anjaly Parayil, Ankur Mallick, Esha Choukse, Xiaoting Qin, Jue Zhang, Íñigo Goiri, Ruija Wang, Chetan Bansal, Victor Rühle, Anoop Kulkarni, Steve Kofsky, and Saravan Rajmohan. Intelligent router for llm workloads: Improving performance through workload-aware scheduling. *CoRR*, abs/2408.13510, 2024.
- [27] Sachin Katti, Jeff Cohen, and Dina Katabi. Information slicing: Anonymity using unreliable overlays. In *Proc. of NSDI 2007*, 2007.
- [28] Hugo Krawczyk. Secret sharing made short. In *Proc. of CRYPTO*, 1993.
- [29] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with page-dattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.
- [30] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. Infinigen: efficient generative inference of large language models with dynamic kv cache management. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, 2024.
- [31] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of ICML*, 2023.
- [32] Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. Loogle: Can long-context language models understand long contexts? *arXiv preprint arXiv:2311.04939*, 2023.
- [33] Yuhan Liu, Yuyang Huang, Jiayi Yao, Zhuohan Gu, Kuntai Du, Hanchen Li, Yihua Cheng, Junchen Jiang, Shan Lu, Madan Musuvathi, et al. Droidspeak: Kv cache sharing for cross-llm communication and multi-llm serving. *arXiv preprint arXiv:2411.02820*, 2024.
- [34] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024.
- [35] Glenn Mansfield, Kohei Ohta, Y. Takei, N. Kato, and Y. Nemoto. Towards Trapping Wily Intruders in the Large. *Computer Networks*, 2000.
- [36] MLC team. MLC-LLM, 2023-2025.
- [37] Donald R. Morrison. Patricia—practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 1968.
- [38] NVIDIA. Tensorrt-llm, 2025. <https://github.com/NVIDIA/TensorRT-LLM>.
- [39] Claus Pahl, Antonio Brogi, Jacopo Soldani, and Pooyan Jamshidi. Cloud Container Technologies: A State-of-the-Art Review. *IEEE Transactions on Cloud Computing*, 2017.
- [40] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024.
- [41] Chen Qian, Junjie Shi, Zihao Yu, Ye Yu, and Sheng Zhong. Garlic cast: Lightweight and decentralized anonymous content sharing. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 216–223, 2016.
- [42] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *the Journal of ACM*, 1989.
- [43] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 1998.
- [44] Ronald L Rivest. All-or-nothing encryption and the package transform. In *Fast Software Encryption: 4th International Workshop, FSE'97 Haifa, Israel, January 20–22 1997 Proceedings* 4, pages 210–218. Springer, 1997.
- [45] Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. From words to watts: Benchmarking the energy costs of large language model inference. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9. IEEE, 2023.

- [46] A. Shamir. How to share a secret. *the Communications of the ACM*, 1979.
- [47] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zuoquan Li, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. Fairness in serving large language models. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, 2024.
- [48] Clay Shields and Brian Neil Levine. A protocol for anonymous communication over the internet. In *Proc. of CCS*, 2000.
- [49] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, 2024.
- [50] Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Efficient distributed prompt scheduling for llm serving. In *Proc. of ICLR*, 2025.
- [51] Douglas R. Stinson. Something about all or nothing (transforms). *Designs, Codes and Cryptography*, 22(2):133–138, 2001.
- [52] Hugo Touvron et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*, 2023.
- [53] vLLM Team. A high-throughput and memory-efficient inference and serving engine for llms, 2025. <https://github.com/vllm-project/vllm>.
- [54] David Isaac Wolinsky, Ewa Syta, and Bryan Ford. Hang with your buddies to resist intersection attacks. In *Proc. of CCS*, 2013.
- [55] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, 2024.
- [56] Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920*, 2023.
- [57] Rui Ye, Wenhao Wang, Jingyi Chai, Dihan Li, Zexi Li, Yinda Xu, Yaxin Du, Yanfeng Wang, and Siheng Chen. OpenFedLLM: Training Large Language Models on Decentralized Private Data via Federated Learning, 2024.
- [58] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soo-jeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA, 2022. USENIX Association.
- [59] Lianmin Zheng et al. Sglang: Efficient execution of structured language model programs. In *Proc. of NeurIPS*, 2024.
- [60] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, 2024.
- [61] Li Zhuang, Feng Zhou, Ben Y Zhao, and Antony Rowstron. Cashmere: Resilient anonymous routing. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-VOLUME 2*, pages 301–314, 2005.

Appendix

A1. Limitations of existing overlay anonymous routing.

Limitations of overlay Onion routing. Onion routing [43] was proposed to achieve anonymous routing in overlay networks, such as Tor [17]. If we apply the same idea to the overlay network in PlanetServe, there are a few main challenges. First, the first layer of Onion nodes (called as guards in Tor) will definitely know the sending user of a request, which should be avoided in PlanetServe. Second, the failure probability of an Onion path increases exponentially with the path length, especially for an overlay network of users with a high churn rate. In addition, each relay needs to perform public-key decryption on the entire message, introducing non-trivial overhead for users.

Strengths and weaknesses of slicing protocols. One idea is to use sliced messages for overlay anonymity [27, 41]. In garlic routing [41], the sender sends n slices, each is about $1/k$ of the original message size long and is called a clove. If the receiver gets k out of n cloves, it can recover the original message. Garlic achieves 1) anonymity without encryption and decryption operations on the paths; 2) reliability under node failures; 3) the bandwidth cost does not increase significantly with the $n - k$ redundant cloves. Garlic routing has a few weaknesses if we apply it directly to PlanetServe. 1) Garlic relies on random walks to establish routing paths and the corresponding “proxies”. However, random walks add uncertainty to the success of finding enough proxies. 2) Garlic preserves the provider (model node) anonymity, which is unnecessary for PlanetServe.

Algorithm 1 HashRadix Tree Search

```

1: function SEARCH(HRTree, request,  $\tau_c$ )
2:    $d \leftarrow 0$ 
3:    $l_1, l_2, \dots, l_n \leftarrow HRTree.L$ 
4:    $k \leftarrow 0$ 
5:   for  $i \leftarrow 1$  to  $n$  do
6:      $chunk_i \leftarrow hash(request[k..k + l_i], HRtree.mod)$ 
7:      $k = k + l_i$ 
8:   end for
9:    $node \leftarrow HRTree.root$ 
10:  for  $i \leftarrow 1$  to  $n$  do
11:    if  $node$  has child  $chunk_i$  then
12:       $node \leftarrow node.children[chunk_i]$ 
13:       $d \leftarrow d + 1$ 
14:    else
15:      return  $node.modelNodeList, d$ 
16:    end if
17:  end for
18:  if  $d < \tau_c$  then
19:    return  $\emptyset$ 
20:  else
21:    return  $node.modelNodeList, d$ 
22:  end if
23: end function

```

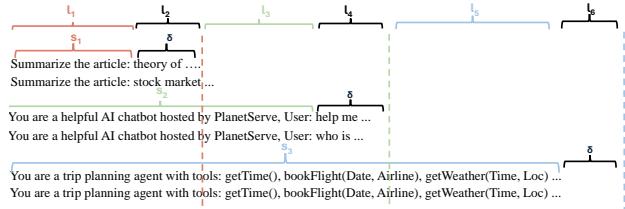


Figure 18: Chunk length array L set by Sentries.

A2. Pseudocode

Algorithm 1 presents the pseudocode of the HR-Tree search algorithm.

Algorithm 2 presents the pseudocode of the load-balancing algorithm with the HR-tree result.

Algorithm 3 presents the pseudocode of the model verification process.

A3. The Sentry algorithm for the chunk length array

Determining the optimal configuration for the chunk length array L presents a significant challenge. Setting the chunk sizes too small results in a HashRadix Tree (HR-Tree) with excessive depth, potentially over-differentiating prefixes. Conversely, if chunk sizes are too large, requests with substantial prefix overlap and high cache affinity might be incorrectly treated as entirely distinct. To address this, PlanetServe in-

corporates a Sentry module. This component is responsible for collecting incoming requests and detecting common patterns, such as system prompts, which can inform the dynamic adjustment or initial configuration of the chunking strategy defined by L . How to set L to efficiently utilize the HR-Tree? As described in Fig. 18, we define $S = s_1, s_2, \dots, s_n$ be the lengths of the distinct common system prompts detected by the Sentries, sorted in ascending order. The Sentry module uses these lengths, along with a small, fixed separator length δ , to construct the chunk length array L used in Algorithm 1. The array L is defined as follows:

$$l_1 = s_1 \quad (1)$$

$$l_{2n} = \delta \quad (2)$$

$$l_{2n+1} = s_n - s_{n-1} - \delta \quad (3)$$

This structure ensures that each distinct system prompt corresponds to a specific path segment in the HR-Tree, separated by a small δ chunk. This allows the initial layers of the HR-Tree to be efficiently utilized for routing based on common prompts, thereby enhancing cache affinity for requests that share these initial structures. Subsequent parts of the request (beyond the detected prompts) are tokenized using any remaining lengths in L or a default chunking strategy.

A4. Analysis of choice of n and k in anonymous routing.

In anonymous routing of PlanetServe, we need k of n clovers to arrive for a successful communication. There are $l = 3$ relays on each path. Assume the failure rate of an overlay node during the time of a round of communication is f . The probability that one path succeeds is $(1 - f)^3$. The rate that at least k out of n succeed is:

$$P(X \geq k) = \sum_{i=k}^n \binom{n}{i} (1-f)^{3i} (1-(1-f)^3)^{n-i}$$

Using $n = 4$ and $k = 3$, even with a failure rate as high as 3%, the success rate is $> 95\%$.

A5. Definition of Anonymity Metric

Definition. Let S be the set of all nodes in the network with $|S| = N$. An attacker assigns to each node $x \in S$ a probability p_x of being the source (or destination) of a message. The entropy of the system is defined as:

$$H(S) = - \sum_{x \in S} p_x \log_2(p_x).$$

Perfect anonymity occurs when the attacker has no information about the system, and thus must assign a uniform probability of $1/N$ to each node. In this case, the entropy reaches its maximum:

$$H_{\max} = \log_2(N).$$

Algorithm 2 Load Balancing

```
1: function LOADBALANCING(HRTree, request,  $\tau_c$ )
2:    $modelNodeList \leftarrow search(HRTree, request, \tau_c)$ 
3:   if  $modelNodeList = \emptyset$  then ▷ Cache miss
4:     Forward to model node with fewest relative requests
5:   else ▷ Cache hit
6:      $candidate \leftarrow$  Model node with fewest relative requests in  $modelNodeList$ 
7:     if  $candidate.load < candidate.threshold$  then
8:       Forward to  $candidate$ 
9:     else
10:      Forward to one with fewest relative requests in all model nodes ▷ Fallback to load balancing
11:    end if
12:  end if
13: end function
```

Algorithm 3 Model Verification

```
function CHECKCREDIBILITY(prompt, output) ▷ Tokenized
2:    $probabilities \leftarrow []$ 
3:    $context \leftarrow prompt$ 
4:   for  $i \leftarrow 0$  to  $length(output) - 1$  do
5:      $next\_token \leftarrow output[i]$ 
6:      $logprobs \leftarrow GetCompletionLogprobs(context, max\_tokens = 1)$ 
7:      $token\_prob \leftarrow FindMatchingProbability(logprobs, next\_token)$ 
8:     if  $token\_prob$  exists then
9:        $probabilities.append(token\_prob)$ 
10:    else
11:       $probabilities.append(\epsilon)$  ▷ \{Small constant\}
12:    end if
13:     $context \leftarrow context + next\_token$ 
14:  end for
15:  return  $score \leftarrow NormalizedPPL(probabilities)$ 
16: end function
```

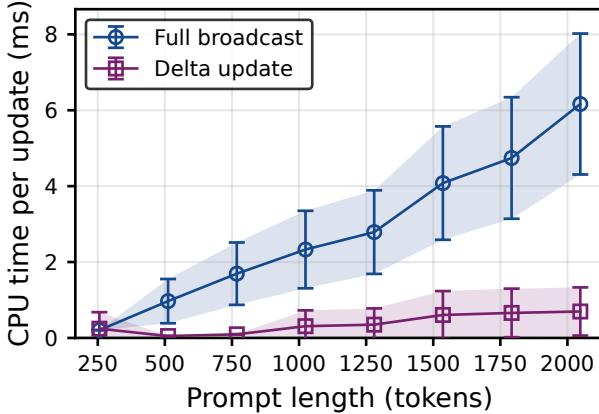


Figure 19: Computation cost for HR-tree update

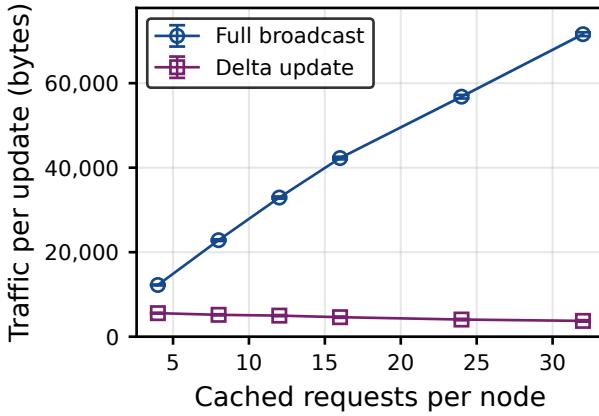


Figure 20: Network cost for HR-tree update

To evaluate the anonymity of the system, we normalize the entropy as a metric. The normalized anonymity is given by:

$$\frac{H(S)}{H_{\max}} = \frac{-\sum_{x \in S} p_x \log_2(p_x)}{\log_2(N)}.$$

We use the above metric but need to know how to compute p_x . We assume a fraction f of all users are collaborative malicious users. Some of them may sit on the k paths between the client and the proxies. However, since they have no information on the prompt, they cannot tell if they are serving the same client. Suppose the number of all nodes on the k paths is L . There may exist multiple chains of consecutive attackers on the paths. The attackers may guess that the predecessor of every chain is the source and the successor of every chain is the destination. Hence the probability of a guess being correct is $\frac{1}{L+1-f \cdot L}$. Let Γ be the set of predecessors of all malicious chains. We then have,

$$\Pr(x = \text{src}) = \begin{cases} \frac{1}{L+1-f \cdot L} & \text{if } x \in \Gamma \\ (1 - \frac{|\Gamma|}{L+1-f \cdot L}) \frac{1}{(1-f)N-|\Gamma|} & \text{otherwise} \end{cases}$$

The entropy-based anonymity can be computed using the above probabilities.

A6. HR-tree Update Cost

We evaluate the computation cost and network cost of HR-tree update by comparing the full broadcast design and the proposed delta update and show the results in Fig. 19 and Fig. 20, respectively. The results show that delta update significantly reduces the cost.

A7. Latency of LLM serving with A1000 GPUs

We evaluate the latency of LLM serving with A1000 GPUs and show the results in Fig. 21. Compared to the A100 results in Fig. 14, PlanetServe shows similar advantages.

A8. Latency Comparison with Centralized Baselines

We compare PlanetServe with a centralized baseline of 8 A100 GPUs without KV-cache sharing and centralized serving with 8 A100 GPUs, running vLLM with the default continuous-batching scheduler and tensor parallelism. Fig. 22 shows the latency comparison. We find that PlanetServe's latency is close to the Centralized sharing method, but significantly shorter than that of Centralized non-sharing.

A9. Resiliency to Intersection Attacks

An intersection attack [54] can be launched to anonymity systems if the target uses pseudonymity and the attacker has access to the user's history of visits and ISP logs. PlanetServe treats each sequence of prompts as independent. Hence users do not use pseudonymity and the attacker cannot obtain user's history. Even though the attacker can access ISP logs (very big assumption!), they cannot match the user behavior among the massive amount of Internet users.

A10. Real-World Routing Latency Measurements

To evaluate the practical impact of network overhead, we measured routing latency on real cloud infrastructure. Specifically, we deployed 14 AWS t3.micro instances, with each hop placed in a different region (datacenter), and used a mixed workload to simulate real prompt payloads. Over the course of 4000 runs, we observed three failures in the across-USA setting and one failure in the across-world setting.

Results are summarized in Fig. 23. In the across-USA experiment (four regions in the United States), the session-establishment latency was 168.9 ms (P99: 256.8 ms), while the steady in-session latency was 92.9 ms (P99: 179.2 ms). In the across-world experiment (five regions across North America, Asia, Europe, and South America), the session-establishment latency was 577.4 ms (P99: 685.8 ms), and the steady in-session latency was 919.6 ms (P99: 1025.5 ms).

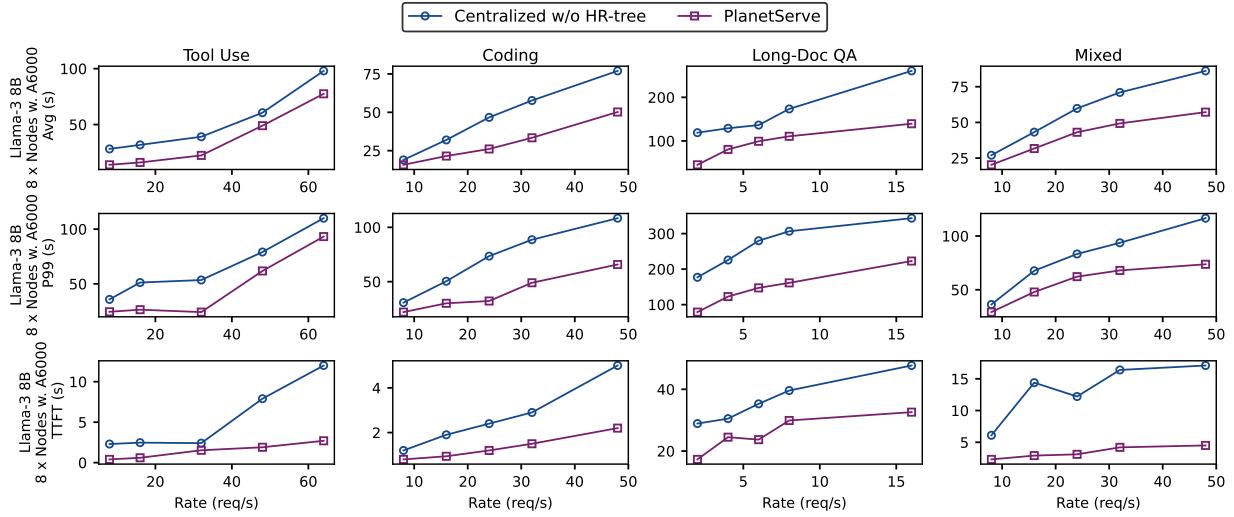


Figure 21: Latency of LLM serving with A1000 GPUs

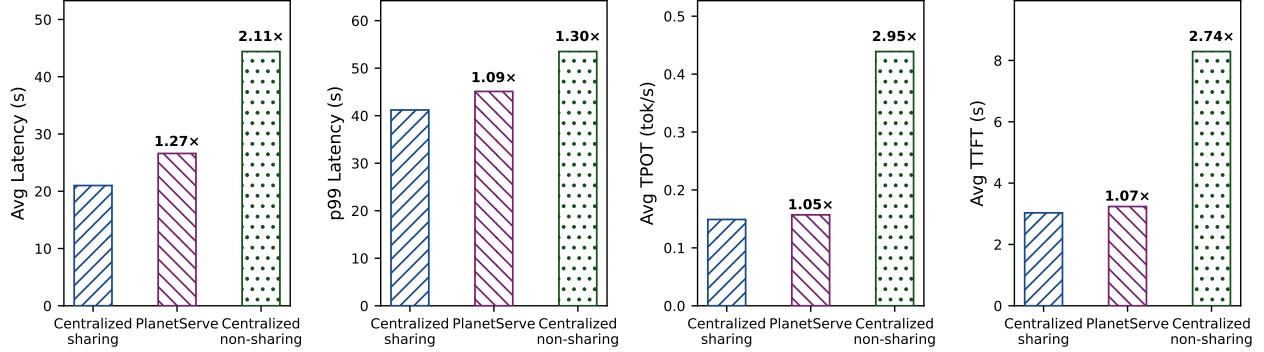


Figure 22: mixed with upper-bound

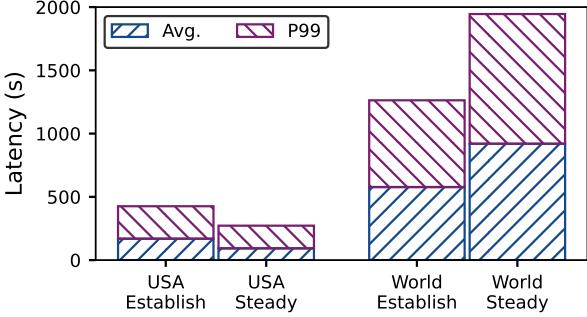


Figure 23: Measured Session-Establish and In-Session Latency Across Regions

Even at inter-continental scales, these latencies are modest compared to LLM inference time (see Fig. 14), suggesting that communication does not dominate end-to-end serving time.