ChatGPT

Distributed Usage of Local LLMs in a Decentralized Agent Network

Using local large-language models (LLMs) on each node and having them cooperate in a decentralized agent network requires careful design. The model weights aren't globally shared or split up (each node runs its own full model), but *tasks* and *inference workloads* are distributed across peers. This approach avoids centralized AI APIs or cloud services, improving resilience and privacy. Below we outline how such a system could work, focusing on how nodes advertise/accept tasks, how IPFS and smart contracts aid coordination, and how to ensure secure and verifiable inference. We also discuss existing frameworks (e.g. Bittensor, Gensyn, Æthos/Masumi) and how this fits into ChefsPlan's off-chain agents (Matching Agent, Compliance Agent, AI Copilot).

Node Advertisement and Task Sharing

To cooperate, nodes need a way to **advertise their LLM services** and discover tasks from others. A common pattern is to use a **decentralized registry or marketplace** smart contract where each node (or the agent running on it) registers its capabilities and terms:

- Each node can publish its **service offer** on-chain (e.g. model type/version, performance specs, pricing, etc.) along with a persistent identity (such as a Decentralized ID). For example, the Masumi protocol lets AI agents register their services on a *fully decentralized registry*, set usage prices, and declare they can accept jobs [1]. This listing is backed by the node's on-chain identity and possibly a stake or reputation score to signal trustworthiness.
- A node looking to **invoke an LLM task** can query this registry to find suitable providers (e.g. a node running a specific model or offering the needed GPU memory). The requesting node might either **address a request directly** to a chosen provider or broadcast a job offer that any eligible node can grab. In the Masumi network, for instance, agents can *discover and delegate tasks to other AI agents* and conversely *receive job requests from others* (monetizing their services) [1]. A similar mechanism in our design means nodes openly advertise "I have model X and can handle Y tokens/sec, cost Z per token," and listen for incoming requests.
- **Smart contracts** can coordinate the matchmaking and ensure commitments. For example, a task request could be posted on-chain with a bounty or payment. A provider node claims it via the contract (potentially locking up a bond as assurance), executes the task off-chain, then returns the result reference and gets paid. The contract could hold stakes to penalize misconduct. This on-chain escrow and advertisement system creates a **trust-minimized marketplace** for LLM queries. It is analogous to how blockchain oracles or decentralized marketplaces work, but here for AI services [2]. Using a blockchain provides a global view of available agents and an append-only log of who committed to what task, which helps in dispute resolution or reputation tracking.

Under the hood, the actual **communication of task data** would happen off-chain (for performance) via peer-to-peer channels (e.g. libp2p or similar). The smart contract or registry might just exchange task IDs and content hashes, while the bulk of the data flows directly between nodes or through a storage network like IPFS (described next). This ensures scalability by not congesting the blockchain with large payloads. Nodes can also periodically broadcast availability on a pub-sub network or DHT so that even

purely off-chain agents can find each other without every query hitting the blockchain (for lower-latency coordination).

In summary, each node actively advertises its LLM capabilities in a decentralized way and listens for tasks. A requesting agent can find a provider through a smart contract registry or P2P discovery, negotiate the job (possibly via on-chain offers or off-chain handshake), and then hand off the task for execution. Because these advertisements and agreements are public/transparent (e.g. recorded in a contract or DHT), it deters bad actors and enables reward mechanisms. This is the foundation for a cooperative network of LLM-bearing agents rather than calling a centralized API.

Task Logic and IPFS for LLM Workflows

IPFS (InterPlanetary File System) plays a crucial role in **storing and sharing task descriptions, data, or logic** in this decentralized setup. Instead of sending large prompts or complex instructions directly through transactions or P2P messages (which could be lost or tampered with), nodes can leverage content-addressable storage like IPFS for reliability and transparency:

- **Task descriptions on IPFS:** When a node wants to request an LLM operation, it can package the prompt, input data, or even a sequence of actions (an agent script or chain-of-thought) as a file and add it to IPFS. This yields a content hash (CID) that uniquely identifies the content. The hash can then be posted on-chain or sent to the provider node. By doing this, all parties have a **tamper-proof reference** to exactly what needs to be done. The smart contract or off-chain agreement only needs to carry the IPFS CID and maybe a brief title/metadata, rather than the full payload [2]. This keeps on-chain data minimal while using IPFS to retrieve the full task details on demand. It also means any node with the CID can fetch the identical instructions, ensuring consistency. For example, a complex prompt or a dataset for the LLM to analyze could be stored this way if it's too large for a transaction.
- **Storing execution logic or models:** Not only the input data, but even the *logic of how to use the model* could be shared via IPFS. For instance, if an agent has a special reasoning script or tool-usage sequence for the LLM (like a particular prompt template or an agent workflow), it could publish that script to IPFS. Other nodes could then load and execute that logic in a sandbox, allowing *task logic to be distributed* alongside tasks. This approach can facilitate standardization – e.g. a community-developed "LLM usage recipe" (prompt program) is referenced by hash so that all nodes execute the same steps. IPFS ensures that every node uses the exact code or prompt provided (it can't be silently altered due to content addressing). However, this requires nodes to trust or verify the logic before execution (perhaps through code audit or reputation of the publisher).
- **Outputs and results via IPFS:** After a node computes the result (e.g. the LLM's generated answer or analysis), it can likewise store the result on IPFS and return the content hash to the requester (and/or log it on-chain). This has two benefits: (1) large results are delivered off-chain without size limits, and (2) the result is content-addressed, so anyone can later retrieve it and confirm it matches the hash committed by the node. In practice, the serving node might directly send the result to the requester as well, but the IPFS copy serves as an immutable audit trail. Some systems require agents to log the hash of outputs on-chain for transparency – for example, in Masumi each agent *logs hashes of their outputs on the blockchain*, allowing recipients and third parties to verify that the output wasn't modified and indeed came from that agent [3]. IPFS would be the natural place to store the actual output content corresponding to that hash.

Using IPFS in this way makes the architecture **more decentralized and trustable**. The logic and data of LLM tasks live on a distributed storage network rather than any single server. It means a node cannot easily lie about what prompt it received or what result it produced – the content hashes serve as a

verifiable fingerprint. It also enables *reusability*: if multiple nodes need the same data (say a knowledge base or a large text), it's retrieved from the network once and cached, instead of every node fetching from a central source. Finally, IPFS plus smart contracts allow a clean separation: **the blockchain handles agreements, payments, and pointers**, while IPFS carries the heavy content load. This pattern (store content off-chain, reference by hash on-chain) is commonly used to keep on-chain contracts efficient [2].

In summary, IPFS can be used to **transmit task inputs and logic and to publish results** in a tamper-resistant way. Nodes advertise an IPFS hash for what they need done, provider nodes fetch that data from the IPFS network to ensure they have the exact instructions, and later the outputs can be persisted on IPFS for verification. This greatly supports the decentralized nature of the network by removing reliance on any central database or cloud storage.

Verifying and Trusting Inference Results Across Peers

A major challenge in a decentralized LLM network is **trust**: How can a requester trust that a peer actually ran the intended model correctly and returned a valid result? Unlike a centralized service where reputation and legal contracts enforce honesty, a trustless network requires *technical and economic mechanisms* to verify or incentivize correct behavior. Several complementary strategies can be combined to achieve secure and verifiable inference:

- **Cryptographic Proofs of Execution:** In theory, the strongest guarantee would be a **zero-knowledge proof** that the node executed a specific model on the given input and produced the given output. This would cryptographically prove correctness without needing to trust the node's word. Projects are exploring ZK-SNARKs for neural network inference (e.g. zkML frameworks like EZKL, Modulus, Giza) – these let a prover convince others that "I ran this NN and got this result" with a proof much smaller than the actual computation. However, today this approach is **hugely impractical for large models**. Compiling a large transformer into a circuit and proving it was executed can slow inference by orders of magnitude (1000× or more overhead in time and cost) [4]. Thus, while ZK proofs *guarantee* correctness cryptographically, they are not yet scalable for 70B parameter LLMs in real-time. We might see progress on specialized ZK accelerators or more efficient proof systems, but for now ZK is used only for small models or very high-value, infrequent tasks.
- **Optimistic Verification (Challenge Games):** Another approach is **optimistic execution with fraud proofs**, inspired by optimistic rollups in blockchain. Here, the network assumes the node's result is valid by default, but allows a window where anyone (or designated "watchers") can challenge the result if they suspect fraud [5]. Practically, a few watcher nodes would re-run the same inference task on their own (using the same model and input from IPFS). If they find a discrepancy – for example, the provider claimed an output that their own run doesn't match – they can trigger a **challenge protocol**. A smart contract could then mediate a dispute: for instance, requiring the provider to reveal a step-by-step computation trace or model activations, and the challenger pinpoints where it diverged (similar to Truebit-style interactive verification [6]). Because models are huge, this on-chain bisection of the execution trace would be extremely costly if fully carried out – but the *game theory* is that rational nodes won't cheat if they know a challenge could happen. The cost of a full fraud-proof (re-running a 1-petaflop model on-chain, as one estimate notes [7]) is prohibitive, so ideally the mere threat of it (plus slashing of a big bond) deters cheating. Optimistic schemes require **watcher nodes** who independently recompute many tasks off-chain to keep providers honest, which multiplies total compute cost. For example, if 5 watchers shadow every task, effectively 6× the computation is done (1 by the provider, 5 by watchers). The user ultimately bears that cost in fees [8]. The upside is that if no one cheats, you get near-native performance (only minimal on-chain footprint for occasional

disputes), and security holds as long as *at least one honest node* is watching and willing to challenge wrong results [9] [10]. Protocols like **Gensyn** are reportedly exploring optimistic verification: Gensyn's design involves an Ethereum rollup that coordinates off-chain execution and verification, implying fraud-proof style checks for "untrusted operations" in ML tasks [11] [12]. This approach can be a good fit if the network has many redundant nodes that can serve as checkers for each others' work.

- **Redundancy and Voting (Cryptoeconomic Consensus):** A more straightforward but somewhat weaker method is to have **multiple nodes perform the same task** and compare outputs. This is akin to an oracle network or Chainlink-style approach to trust. For a given query, the requester can ask, say, 3 independent nodes to run it. They all submit their results (perhaps hashed, via a commit-reveal scheme to prevent copying [13]). If one node's output disagrees with the majority, that node is deemed faulty or malicious and can be slashed (its stake confiscated) or penalized [9]. In the ideal case where at most one node cheated, the majority vote yields the correct result. The user pays for N executions to get assurance, effectively choosing a **cost vs. trust** tradeoff (e.g. pay 3× cost to have 3 nodes and a majority vote). This cryptoeconomic approach assumes honest majority and sufficient financial disincentives to cheating. It's fast (no long challenge game; a result can be accepted after one round if no discrepancies) and simple to implement. However, it is **not as secure if a majority collude** or if all chosen nodes decide to cheat the same way. A robust network can mitigate this by enforcing diversity (nodes run by different parties) and by making each node put down a significant stake that they lose if they return a bad result. The **stake-weighted voting** idea is essentially that nodes with more stake or better reputation carry more weight, aligning their incentives with honest behavior [10]. This approach is flexible – users who need very high confidence can increase N or require a certain total stake backing the answers. If the task is not adversarial (e.g. a straightforward computation), even a single node might be fine; for critical tasks, multiple independent runs provide assurance. Cryptoeconomic consensus is weaker than cryptographic proof, but in practice it can deter casual cheating effectively, especially if combined with reputation systems.
- **Trusted Hardware (TEEs):** As a complement to the above, nodes could run the model inside a **Trusted Execution Environment** (like Intel SGX, ARM TrustZone, or secure enclaves). These TEEs can produce a remote attestation – basically a cryptographic certificate – that the node is running a specific code (e.g. a particular model binary) with integrity. A requester could require that the provider node supplies an attestation from its hardware, proving *"I am running the unmodified LLaMA-2 70B model code in an enclave"*. While this doesn't prove the output is *correct* for a given input, it does prove that the node hasn't swapped out the model for a smaller one or tampered with it (which addresses a big part of the trust problem: ensuring the model used is the one promised [14]). Cloud providers like Azure offer confidential computing VMs that support this. In a decentralized network, if participants have modern CPUs supporting SGX or similar, they could leverage it. The downside is this introduces **trust in hardware manufacturers** and is not fully permissionless (you trust Intel/AMD's attestation service). It also may degrade performance somewhat. But it can be a useful layer: for example, a smart contract could verify the attestation signature to ensure payout only goes to nodes running approved model code.
- **Reputation and Auditing:** Over time, the network can build a **reputation system** for nodes. If a particular node has completed hundreds of tasks correctly (with no disputes or with consistently matching other nodes' outputs), it earns a high trust score. Others may prefer to send tasks to it or accept its results with less verification (similar to how one might trust a highly rated seller on a marketplace). Reputation can be on-chain (e.g. a rating or a stake that grows with successful jobs) or off-chain (community maintained). Some designs (like Bittensor's) have nodes *evaluate each other's usefulness* continuously and adjust scores accordingly [15] [16]. In Bittensor, peers essentially rank the value of information each node provides, and a blockchain layer rewards those that are consistently evaluated well by others [17] [16]. Applying that here, one could imagine nodes occasionally cross-testing each other: sending known queries or performing peer

review on random tasks to see if a node's output is good. A **reputation-based scheme** could also weight answers in the voting approach above (e.g. higher-rep nodes' answers are trusted more). While reputation alone doesn't guarantee correctness, it raises the cost of cheating (a node would lose its hard-earned reputation and future earnings). This should be used alongside cryptoeconomic incentives – for example, nodes might have to stake tokens to provide services, and if they are caught cheating via any of the above mechanisms, their stake is slashed and their reputation score reset. Finally, *auditing* can be used: some tasks could be inserted whose answers are known (like hidden test queries) to catch nodes that try to lazy-respond without actually computing. Such test prompts (with known outputs or easily checkable results) can keep providers honest.

In practice, a combination of these methods yields the best results. For example, the system might use **multi-node redundancy by default for critical queries**, and also allow **fraud challenges** on any result within a time window. Meanwhile, each node's long-term reputation and stake provide a background incentive to be honest. If extremely high assurance is needed for a particular task, a ZK-proof might be used (if a smaller model or a distilled version can be proved), or tasks could be split into verifiable sub-components. The key is that the network is **trust-minimized**: you don't have to blindly trust a random peer's response. Through cryptography and economic design, cheating either becomes *impossible* or *unprofitable*. This focus on verifiability ensures even a decentralized, open network can reliably execute LLM workloads with integrity [14] [9].

Distributed Inference Frameworks and Protocols

This vision is already being tackled by several **emerging frameworks and protocols**. ChefsPlan can draw inspiration or components from these projects that aim to decentralize AI compute:

- **Bittensor:** Bittensor is a decentralized AI network where participants (nodes) run models and **evaluate each other's outputs to generate a reputation score**, forming a kind of market for machine intelligence [15]. It uses a substrate-based blockchain (layer-0) to handle peer identities, consensus and token rewards [17]. Each peer (running a local model) queries other peers and ranks the usefulness of their responses. Peers that consistently provide valuable answers get higher "trust" weights and thus earn more of the native token (TAO) [16]. This incentivizes nodes to run high-quality models and not cheat, since other AI agents are essentially judging them. Bittensor's design is more about a *collaborative training/inference market* than direct user-issued tasks, but its core ideas can be adapted: a blockchain for **identities and staking**, a mechanism for **peer assessment**, and token incentives for good performance. In a ChefsPlan context, one could use a similar approach to reward nodes that provide accurate or helpful LLM results (perhaps using the Matching/Compliance agents to do the evaluations akin to Bittensor's peer scoring). Bittensor shows that decentralized AI can be self-regulating: it "measures" model performance via the network itself and resists collusion by only rewarding nodes that a majority of others also find trustworthy [16].
- **Gensyn:** Gensyn is a protocol focused on *verifiable decentralized compute* for machine learning. Their public testnet (launched March 2025) uses a custom Ethereum rollup chain to coordinate tasks, **maintain persistent node identities, track contributions, handle payments, and verify off-chain computations** [11]. Gensyn's approach leans on *optimistic verification*: tasks (like model training or inference jobs) are executed on worker nodes off-chain, and the rollup can verify these "untrusted operations" using fraud proofs or similar techniques if needed [11] [12]. Essentially, Gensyn is building an on-chain marketplace where ML jobs can be posted and fulfilled with guarantees that either the results are correct or the worker will be caught/challenged. They also emphasize an identity system for nodes (so you know which node did what) and an attribution system to reward contributions. A feature of Gensyn is enabling *crowd-*

sourced training (e.g. many nodes collaborate on training a model and the chain tracks each contribution), but for inference tasks the same infrastructure allows trustless outsourcing of computation. From Gensyn, ChefsPlan could adopt the idea of using a **dedicated blockchain or rollup for AI agents** that handles job scheduling and result verification. The integration of off-chain execution frameworks in Gensyn shows how to blend smart contracts with heavy computation: let the heavy LLM runs happen off-chain, but enforce correctness via on-chain dispute mechanisms. Gensyn is still evolving, but it provides a template for **"don't trust, verify"** in AI services using a combination of crypto-economic incentives and rollup architecture [5] [7].

- **Æthos / Masumi:** The Masumi protocol (an initiative under Æthos) is explicitly geared towards an **AI agent network**, very similar in spirit to the ChefsPlan scenario. Masumi's design centers on enabling AI agents to *discover each other, share tasks, and monetize services in a decentralized way*. Concretely, Masumi gives each agent a blockchain wallet and a Decentralized ID, and provides a **fully on-chain registry of agent services** [1]. Agents register what they can do, and they can send job requests or receive them through this network. Payment is handled with a native token (e.g. "Sumi" token) for per-use transactions. A key feature is that every agent must log a hash of its outputs on-chain for auditability [3] – this creates an immutable record of agent actions, which aids in accountability. Masumi also stresses *trust through transparency*: every interaction is on an open ledger, and agent identities are verified, so you can build a Web-of-trust or track an agent's history. Essentially, Masumi is building the **financial and trust infrastructure for autonomous AI agents** to cooperate and trade services. For ChefsPlan's needs, Masumi offers ready-made patterns: a **decentralized service directory**, **agent identity and reputation via DIDs**, and secure logging of outcomes. One could imagine ChefsPlan's Matching Agent and others plugging into a Masumi-like network to advertise tasks (matching jobs) and consume LLM services from external agents, with all interactions mediated by smart contracts for fairness. The emphasis on *agent accountability* (logging outputs, decisions) is very relevant for a Compliance Agent role as well. If ChefsPlan doesn't use Masumi directly, it can adopt similar decentralized identity and logging mechanisms to ensure each LLM action is traceable and attributable.
- **Other Protocols:** There are several other projects exploring this space. **Atoma Network** (and a similar concept called **Ritual**) have been mentioned as trying a stake-and-slash approach for ML services, where nodes stake tokens and form quorums to answer queries, slashing any outliers [9] [10]. **Bittensor subnets** focus on specialized AI tasks (some subnets aim at LLM inference specifically) and create mini-networks with their own incentives. **Petals** (by Yandex and collaborators) is another noteworthy project, though it takes a different approach: Petals is a decentralized inference network that **splits large models across volunteer nodes** instead of keeping the model local to one node [18]. Users can run a big model like BLOOM-176B by connecting to the Petals P2P network; the model's layers are distributed among peers, and a forward pass pipelines through them. This is essentially model *parallelism* over the network (a client's request is routed through a chain of servers hosting different layers) [18]. Petals achieves the ability to run huge models that no single node could hold, at the cost of network latency and complexity of coordination. In our ChefsPlan scenario, we *assume each node runs a whole model*, so Petals' approach isn't directly what we want – but it's good to note that Petals solves a complementary problem (decentralizing the model itself) and shows the appetite for truly **distributed LLM hosting**. Another example is **Cerebras's Cirrascale** or **Hivemind** (from EleutherAI) for decentralized training, although those are more about training collaboration than inference. The **Beyond OpenAI** report (Messari) and others highlight how a *"decentralized AI stack"* is emerging, with protocols for compute (Gensyn, BitTensor), data, and agent coordination. ChefsPlan can position itself within this ecosystem, leveraging existing tools where possible rather than reinventing the wheel.

In summary, several frameworks are paving the way to **distributed, trustless LLM inference**. Bittensor contributes a model of peer-driven quality control and token incentives; Gensyn contributes verifiable

off-chain execution tied to an L2 blockchain; Masumi focuses on agent marketplaces and accountability; Petals shows how to pool resources to run bigger models. ChefsPlan can borrow ideas from each to create a hybrid that suits its particular network of off-chain agents.

Integration with ChefsPlan's Off-Chain Agents

ChefsPlan's architecture includes off-chain agents like the Matching Agent, Compliance Agent, and AI Copilot. Integrating the above distributed LLM approach would empower these agents to work cooperatively and securely across the network of nodes. Here's how each might fit into the picture:

- **Matching Agent:** The Matching Agent could act as the **orchestrator of task allocation** in the ChefsPlan network. It can use the decentralized registry of LLM services to find which node/agent is best suited for a new task. For example, if a user needs a schedule optimized or a complex query answered, the Matching Agent consults the on-chain registry to find a node with a powerful enough local LLM (or perhaps a specialized fine-tuned model) available. It then either brokers the deal via a smart contract (posting the task with the user's offer and letting that node accept) or directly assigns the task to that node off-chain (after verifying the node's credentials and stake on-chain). Essentially, the Matching Agent ensures supply meets demand: it matches a request to an appropriate provider among the decentralized pool. It might also split a task among multiple nodes if needed for speed (parallelizing subtasks) – for example, splitting a large dataset and asking different nodes to summarize different chunks, then aggregating results. In a decentralized network, the Matching Agent itself could be a distributed service rather than a single server – it could even be an algorithm or smart contract that all nodes run to come to consensus on task assignments. But at a high level, it implements the **discovery and negotiation logic** using the primitives of the network (DHT lookups or contract calls). By doing so off-chain, ChefsPlan avoids any central bottleneck in assigning tasks and can scale as new nodes join offering their LLMs. The Matching Agent would heavily rely on the advertisement scheme and possibly an internal reputation system to route tasks to the best performers.
- **Compliance Agent:** The Compliance Agent's role is to ensure that outcomes and behaviors in the network adhere to rules – both the **business rules** (e.g. a solution meets certain criteria) and possibly ethical or safety guidelines for AI output. In the distributed LLM context, the Compliance Agent can serve as a **verification layer and watchdog**. For instance, once a node produces an LLM output, the Compliance Agent (which could be a process running on one or multiple separate nodes) could double-check the result. This might involve using its own local LLM to evaluate the answer (e.g. detecting if inappropriate content is present, or if the answer likely violates a known constraint). It could also use the **redundant execution** approach: the Compliance Agent triggers a second node to run the same task, comparing results to ensure consistency (implementing the multi-party verification discussed earlier). If discrepancies or rule violations are found, the Compliance Agent can flag this on-chain (perhaps preventing payment and logging a strike against the provider's reputation). In an optimistic verification scheme, the Compliance Agent might act as the **"honest watcher"** that challenges bad results [5]. Since ChefsPlan's agents are off-chain, the Compliance Agent could be free to use heavy analysis or meta-LLM checks on each output without clogging the blockchain – it only reports summary findings (e.g. a boolean pass/fail or a hash of approval) on-chain. Additionally, the Compliance Agent could enforce *smart contract logic* for payments: only releasing funds to the provider node if the output hash matches what was expected and passes all checks (the contract might require a signature from a Compliance Agent key or a certain time with no dispute before finalizing). This agent therefore embodies the **"trust but verify" principle** in the ChefsPlan network, ensuring the distributed LLM usage remains reliable and aligned with the platform's requirements.
- **AI Copilot:** The AI Copilot in ChefsPlan is likely a user-facing assistant that helps with planning, problem solving, or development tasks (perhaps an LLM-driven helper integrated into the

application). By integrating it with the decentralized LLM network, the AI Copilot can become much more powerful and scalable. Instead of the Copilot relying on a single local model or a centralized API, it can **orchestrate queries to the network of LLM nodes** to get the best results. For example, if the Copilot needs to answer a complex query that involves multiple steps or expertise, it could delegate subtasks: one node's LLM to do data extraction, another node (with a different model or tools) to perform reasoning or calculations, etc., then compile the final answer. The Copilot essentially becomes an **agent manager** that uses the decentralized LLM pool as its toolkit. This is in line with the concept of agentic frameworks like AutoGen or CrewAI, but operating over a decentralized backend [19] [20]. The AI Copilot could maintain a high-level dialog with the user but under the hood parcel out work to various nodes via IPFS-described tasks. It would then apply its own logic (possibly also stored on IPFS or predefined) to combine results and generate responses. By doing so, ChefsPlan's Copilot is not limited by the capacity of a single model or machine – it **scales out** with the network. Moreover, the Copilot can incorporate **fallback and validation**: if one node's answer seems dubious, it can consult another node for a second opinion (very naturally integrating the verification step into the user interaction, e.g., "Agent A suggests X, let me double-check with Agent B"). All of this remains off-chain except for the coordination signals, making it fast. The end user just experiences a more robust, fast, and trustworthy AI assistant, without needing to know that behind the scenes multiple distributed models contributed. Finally, because the Copilot is using *local models on different nodes*, sensitive data can be routed to nodes that meet certain criteria (maybe a node run by a party with certain privacy guarantees or jurisdiction, etc.), adding flexibility compared to a one-size-fits-all cloud AI. In summary, the AI Copilot becomes a **meta-agent that leverages the whole decentralized network** as its brain, orchestrating secure, efficient use of LLMs to help the user.

All these agents together, running off-chain, form a cohesive system. The Matching Agent finds the right resources, the AI Copilot delegates and synthesizes, and the Compliance Agent checks and balances. They utilize the **shared infrastructure** of the decentralized LLM network: the on-chain registry/contracts for discovery and payment, IPFS for sharing prompts/results, and the verification mechanisms for trust. ChefsPlan can thus achieve a highly scalable AI ecosystem where adding more nodes (each with their own LLM instance) increases the overall capacity and reliability of the system. Importantly, this is done **without any central server** – the smart contracts and P2P protocols coordinate the collaboration. Security is maintained through cryptographic means and incentive alignment, rather than a central authority.

Conclusion

By distributing LLM tasks across many nodes, a system like ChefsPlan can tap into a **decentralized cloud of AI models**. Each node runs an open-source LLM locally (Mistral, LLaMA, etc.), and the network orchestrates their usage so that complex jobs are solved collectively. The design patterns to enable this include: advertising capabilities via a blockchain-based registry, storing task details on IPFS for consistency, and using smart contracts and crypto-economic tricks to ensure results are trustworthy. Nodes can earn rewards for their work, tracked by on-chain payments and reputation, creating a self-reinforcing incentive for participants to contribute computation. Meanwhile, verification layers (like multi-node consensus or challenge protocols) catch errors or cheating, providing confidence in the outputs even in an untrusted environment [5] [9]. The architecture takes inspiration from existing decentralized AI projects like Bittensor's peer ranking market, Gensyn's verifiable compute network, and Æthos/Masumi's agent marketplace, tailoring their concepts to ChefsPlan's use case. Ultimately, such an approach enables *secure, scalable, and verifiable* LLM-powered agents to cooperate across a decentralized network. This would allow ChefsPlan's off-chain agents – Matching, Compliance, Copilot – to function as a **distributed AI workforce** that is not beholden to any single cloud provider, thus

aligning with the ethos of decentralization while still delivering powerful AI-driven outcomes. All of this is achieved by leveraging blockchain for coordination and trust, and peer-to-peer networks for the heavy lifting, ensuring a future-proof architecture for decentralized AI. [1] [3]

---

[1] [3] [19] [20] Introducing Masumi: Unlocking the AI Agent Economy
https://www.house-of-communication.com/de/en/brands/plan-net/landingpages/agentic-services/masumi.html

[2] PromptChain: A Decentralized Web3 Architecture for Managing AI Prompts as Digital Assets
https://arxiv.org/html/2507.09579v1

[4] [5] [6] [7] [8] [9] [10] [13] [14] Don’t Trust, Verify: An Overview of Decentralized Inference | by Haseeb Qureshi | Dragonfly Research | Medium
https://medium.com/dragonfly-research/dont-trust-verify-an-overview-of-decentralized-inference-c471a9f7a586

[11] [12] Overview | Gensyn
https://docs.gensyn.ai/testnet

[15] [16] [17] Bittensor
https://bittensor.com/academia

[18] Petals: decentralized inference and finetuning of large language models
https://research.yandex.com/blog/petals-decentralized-inference-and-finetuning-of-large-language-models