Hypertensor: A Modular Decentralized Artificial Intelligence Platform

hypertensor.org

Abstract

This document describes the definitions and theories behind Hypertensor V2.0, including how it improves V1.0, new features, and different aspects of the implementation.

Contents

1 Introduction

Hypertensor marks a shift from an MVP (minimal viable product) to a full-fledged AI economy.

The following paper describes the key features that differentiate the MVP from the current form of an AI economy, which is split between the technology of the blockchain and the technology of the subnets, precisely:

2 Blockchain

2.1 Introduction

The blockchain is the Substrate-based base layer of the modular architecture. It makes it easy for anyone to launch a decentralized AI model within a subnet and begin receiving subnet validator incentives.

2.2 Nominated Proof of Stake (NPoS)

Hypertensor uses NPoS (Nominated Proof-of-Stake) as its mechanism for selecting the validator set. It is designed with the roles of validators and nominators, to maximize chain security. Actors who are interested in maintaining the network can run a validator node.

Validators assume the role of producing new blocks, validating blocks, and guaranteeing finality. Nominators can choose to back select validators with their stake. Nominators can approve candidates that they trust and back them with their tokens.

The following is a high-level overview with some technical explanations:

2.2.1 Hybrid Consensus

Hypertensor uses what is known as hybrid consensus. Hybrid consensus splits up the finality gadget (GRANDPA) [Foundation, n.d.] [Alistair Stewart, Eleftherios Kokoris-Kogia, 2020] from the block production mechanism (BABE) [Handan Kilinc Alper, 2020].

This is a way of getting the benefits of probabilistic finality (the ability to always produce new blocks) and provable finality (having a universal agreement on the canonical chain with no chance for reversion) in Hypertensor. It also avoids the corresponding drawbacks of each mechanism (the chance of unknowingly following the wrong fork in probabilistic finality, and a chance for "stalling" - not being able to produce new blocks - in provable finality). By combining these two mechanisms, Hypertensor allows for blocks to be rapidly produced, and the slower finality mechanism to run in a separate process to finalize blocks without risking slower transaction processing or stalling.

2.2.2 Block Production: BABE

BABE (Blind Assignment for Blockchain Extension) is the block production mechanism that runs between the validator nodes and determines the authors of new blocks. BABE is comparable as an algorithm to Ouroboros Praos, with some key differences in chain selection rule and slot time adjustments. BABE assigns block production slots to validators according to stake and using the Hypertensor randomness cycle. The chain's runtime is required to provide the BABE authority list and randomness to the host via a consensus message in the header of the first block of each epoch.

BABE execution happens in sequential non-overlapping phases known as epochs. Each epoch is divided into a predefined number of slots. All slots in each epoch are sequentially indexed starting from 0 (slot number). At the beginning of each epoch, the BABE node needs to run the Block-Production-Lottery algorithm to find out in which slots it should produce a block and gossip to the other block producers.

Validators participate in a lottery for every slot, which will inform whether or not they are the block producer candidate for that slot. Slots are discrete units of time of approximately 6 seconds in length. Because the mechanism of allocating slots to validators is based on a randomized design, multiple validators could be candidates for the same slot. Other times, a slot could be empty, resulting in inconsistent block time.

Multiple Validators per Slot When multiple validators are block producer candidates in a given slot, all will produce a block and broadcast it to the network. At that point, it's a race. The validator whose block reaches most of the network first wins. Depending on network topology and latency, both chains will continue to build in some capacity until finalization kicks in and amputates a fork.

No Validators in Slot When no validators have rolled low enough in the randomness lottery to qualify for block production, a slot can remain seemingly blockless. We avoid this by running a secondary, round-robin style validator selection algorithm in the background. The validators selected to produce blocks through this algorithm always produce blocks. Still, these secondary blocks are ignored if the same slot also produces a primary block from a VRF-selected validator. Thus, a slot can have either a primary or a secondary block, and no slots are ever skipped.

BABE

In BABE, we have sequential non-overlapping epochs $(e_1, e_2, \dots)$, each of which consists of a number of sequential block production slots $(e_i = \{sl_1^i, sl_2^i, ..., sl_t^i\})$ up to some bound t. At the beginning of an epoch, we randomly assign each block production slot to a "slot leader", often one party or no party, but sometimes more than one party. These assignments are initially secrets known only to the assigned slot leader themselves, but eventually, they publicly claim their slots when they produce a new block in one.

Each party Pj has a session key containing at least two types of secret/public key pair:

- A verifiable random function (VRF) key sk_j^v, pk_j^v, and,
- A signing key for blocks sk_j^s, pk_j^s

We favor VRF keys being relatively long-lived because new VRF keys cannot be used until well after creation and submission to the chain. Yet, parties should update their associated signing keys from time to time to provide forward security against attackers who might exploit from creating slashable equivocations.

Each party Pj keeps a local set of blockchains $C_j = \{C_1, C_2, ..., C_l\}$. All these chains have some common blocks, at least the genesis block, up until some height.

We assume that each party has a local buffer that contains a set of transactions to be added to blocks. All transactions in a block are validated with a transaction validation function before entering this buffer.

In BABE, we would like to ensure that each validator has the same chance to be selected as a block producer on a slot. Therefore, we define the probability that a validator is selected for a slot as

$$p = \phi c(\theta) = 1 - (1 - c)^{\frac{1}{n}}$$

Where $0 \leq c \leq 1$ is a constant parameter and n is the number of validators.

In order to achieve the equality of validators in BABE, we define a threshold parameter as in [2] for the slot assignment:

$$\tau = 2^{\ell_v rf} \phi_c(\theta)$$

Where $\ell_v rf$ is the length of the VRF's first output (randomness value).

BABE consists of three phases:

1st: Genesis Phase In this phase, we manually produce the unique genesis block.

The genesis block contains a random number r1 for use during the first two epochs for slot leader assignments. Session public keys of initial validators are $(pk_1^v, pk_2^v, ..., pk_n^v), (pk_1^s, pk_2^s, ..., pk_n^s)$

2nd: Normal Phase We assume that each validator divided their timeline into slots after receiving the genesis block. They determine the current slot number according to their timeline. Similarly, when a new validator joins to BABE after the genesis block, this validator divides his timeline into slots.

In normal operation, each slot leader should produce and publish a block. All other nodes attempt to update their chain by extending with new valid blocks they observe.

We suppose each validator V_j has a set of chains C_j in the current slot sl_k in the epoch e_m and has a best chain C selected in $sl_k - 1$, and the length of C is $\ell - 1$.

Each validator V_j produces a block if he is the slot leader of sl_k. If the first output (d) of the following VRF computation is less than the threshold τ then he is the slot leader.

$$vrf_{sk_j^v}(r_m||sl_k) \rightarrow (d, \pi)$$

If P_j is the slot leader, P_j generates a block to be added on C in slot sl_k. The block B_ℓ should at least contain the slot number sl_k, the hash of the previous block $H_{\ell-1}$, the VRF output d, π, transactions tx, and the signature $\sigma = Sign_{sk_j^s}(sl_k||H_{\ell-1}||d||\pi||tx))$. P_i updates C with the new block and sends B_ℓ.

In any case (being a slot leader or not being a slot leader), when V_j receives a block $B = (sl, H, d', \pi', tx, \sigma)$ produced by validator V_t, it validates a block within $Validate(B)$. $Validate(B)$ must at least check the followings in order to validate the block.

- If $Verify_{pk_t^s}(\sigma') \rightarrow valid$ (signature verification),
- if the validator is the slot leader: $Verify_{pk_t^s}(\pi', r_m||sl \rightarrow valid)$ and $d' < \tau$ (verification with the VRF's verification algorithm),
- if there exists a chain C' with the header H,
- If the transactions in B are valid.

If the validation process goes well, V_j adds B to C'. Otherwise, it ignores the block,

At the end of the slot, P_j decides the best chain with the chain selection rule.

3rd: Epoch Update Starting from the first slot, in every R slots, the new epoch starts.

Before starting a new epoch in e_m, validators should obtain the new epoch randomness and active validators set for the new epoch.

The validator set for the epoch em has to be included in the relay chain until the end of the last block of the epoch e_{m-3} so that they are able to actively participate in the block production in the epoch e_m. So, a new validator can actively join the block production at the earliest two epochs later after being included in to relay chain.

Fresh randomness for the epoch e_m is computed as in Ouroboros Praos [2]: Concatenate all the VRF outputs of blocks in the epoch e_{m-2} (let us assume the concatenation is ρ). Then the randomness in epoch e_m:

$$r_m = H(r_{m-2}||m||\rho)$$

The reason for including a validator after two epochs later is to make sure that the VRF keys of the new validators are added to the chain before the randomness of the epoch that they are going to be active is revealed.

For a more detailed explanation of BABE, see BABE references.

2.2.3 Finality Gadget: GRANDPA

GRANDPA provides block finalization. It has a known weighted authority set like BABE. However, GRANDPA does not author blocks. It just listens to gossip about blocks that have been produced by block authoring nodes. GRANDPA validators vote on chains, not blocks. GRANDPA validators vote on a block that they consider best and their votes are applied transitively to all previous blocks. After two-thirds of the GRANDPA authorities have voted for a particular block, it is considered final.

It works in a partially synchronous network model as long as 2/3 of nodes are honest and can cope with 1/5 Byzantine nodes in an asynchronous setting.

A notable distinction is that GRANDPA reaches agreements on chains rather than blocks, greatly speeding up the finalization process, even after long-term network partitioning or other networking failures.

In other words, as soon as more than 2/3 of validators attest to a chain containing a particular block, all blocks leading up to that one are finalized at once.

Probabilistic vs. Provable Finality A pure Nakamoto consensus blockchain that runs PoW is only able to achieve the notion of probabilistic finality and reach eventual consensus. Probabilistic finality means that under some assumptions about the network and participants if we see a few blocks building on a given block, we can estimate the probability that it is final. Eventual consensus means that at some point in the future, all nodes will agree on the truthfulness of one set of data. This eventual consensus may take a long time, and will not be able to determine how long it will take ahead of time. However, finality gadgets such as GRANDPA (GHOST-based Recursive ANcestor Deriving Prefix Agreement) or Ethereum's Casper FFG (the Friendly Finality Gadget) are designed to give stronger and quicker guarantees on the finality of blocks - specifically, that they can never be reverted after some process of Byzantine agreements has taken place. The notion of irreversible consensus is known as provable finality.

For a more detailed explanation of GRANDPA, see GRANDPA references.

2.3 Subnet Democracy

Each subnet must be voted in by undergoing a democratic process via a proposal mechanism. Hypertensor is building an AI economy, it needs to be a democratic economy. Humanity needs a say in what AI does to have full control of how AI operates so it never gets out of control.

2.3.1 Democratic Subnets

This facilitates the decentralized decision-making process by allowing users to vote on a subnet's induction or removal into or from the overall network. Each new subnet must undergo a democratic governance vote requiring a quorum and consensus. Subnets can be voted in and be voted out under certain circumstances.

Proposal Types:

- Subnet Activation
- Subnet Deactivation

Configuration:

- **Quorum:** 10,000 TENSOR,
- **Consensus:** >50%,
- **Maximum Activation Proposals:** 1,
  - The maximum live amount of proposals for subnet activation.
- **Maximum Deactivation Proposals:** 32,
  - The maximum live amount of proposals for subnet deactivation.
- **Minimum Proposal Stake:** 100 TENSOR,
  - The amount a proposer must stake towards a proposal.
  - The amount is the minimum versus the subnet initialization cost. If the initialization cost is less than the minimum proposal stake, the proposer must stake the minimum proposal stake and will have the difference returned on completion of the proposal.
- **Voting Period:** 9 days,
  - The length of a proposal voting period.
- **Enactment Period:** 21 days,
  - The length after a proposal has been completed to execute the proposal.
- **Verify Period:** 3 days,
  - The length after an activation proposal each subnet node must verify itself.

Query the blockchain for the most up-to-date subnet democracy configuration.

Subnet Activation The proposer must stake the minimum proposal stake or the subnet initialization cost for the length of the proposal, whichever is most, the proposal must have the minimum required subnet nodes to initiate the proposal, and each subnet node in that subnet must have the minimum amount of tokens required to stake as a subnet node in their account.

Activation requires submitting the proposal with the required parameters including the subnet model path from HuggingFace or IPFS, or any URL that can be used to download the model, and the memory requirements to host the entire subnet layer (seen in section 2.5).

Subnet Deactivation Subnets can be proposed to be deactivated.

Subnet Node Verification Each activation proposal requires submitting parameters including each subnet node that is dedicated to validating the subnet once successfully voted in. Each subnet node entered with the activation proposal must sign the proposal to ensure the proposer entered the correct accounts and isn't taking advantage of accounts with enough stake to have their proposal transaction succeed.

Casting Votes Once a proposal is initiated, subnet nodes and accounts can begin voting Yay, Nay, or Abstain by staking TENSOR to the proposal itself. Each voter must be staked under the following conditions:

Vote Eligibility:

- Blockchain Validator
- Blockchain Delegate Staker
- Subnet Validator
- Subnet Delegate Staker

A user's ability to vote is based on the sum of all staking balances on each unique proposal. For example, if multiple proposals exist and a user has 1000 tokens staked as the sum of all staking options, they can use 1000 staked TENSOR to cast a vote on each unique proposal.

Executing By the end of the proposal's designated voting period, the required quorum and consensus must be reached to activate or deactivate the subnet. Otherwise, the proposal is defeated and the subnet is not activated or deactivated.

The proposal must be executed within the enactment period or it will become expired.

Once the proposal is complete, it can be succeeded, defeated, expired, or canceled. In each case, the proposer receives the subnet initialization cost back, and all voters can manually unstake.

If the proposal succeeds by reaching the quorum and a consensus of ¡50%, then, the subnet is initialized or removed from the network. Once the execute function is called, the subnet is activated into the Network Pallet and the proposers' initial stake towards the proposal is unreserved and sent to the Network pallet and distributed to all previously initialized subnets on the epoch following.

Cancelling A proposal can be canceled by the proposer if the voting period has not ended, or by anyone, if the verify period has passed and ¡100% of the subnet nodes still need to verify themselves.

2.4 Subnet Consensus

Consensus is formed via validation and attestation of subnet nodes. Each subnet undergoes its own consensus separately.

2.4.1 Validation

One subnet validator, per subnet, is randomly selected in each epoch slot to be the epochs proposer. These validators are chosen by blockchain validators that are randomly chosen to validate the block at the start of the epoch. These chosen subnet validators submit consensus data of each subnet node in the respective subnet with details about their computational contributions towards the subnet in that epoch as a score. This proposal data then undergoes attestation.

2.4.2 Attestation

Attestation is the process through which validators confirm the accuracy of the blockchain's current state by voting and reaching a consensus, thereby ensuring the security and integrity of the chain. The purpose of the attestation is to vote in favor of the validator's view of the chain.

Once the validator submits its proposal to the blockchain, each subnet node can attest that data or not attest that data. Attestation must reach a minimum of 66% in order to achieve a consensus.

If consensus is reached for a subnet, the subnet validator, the attesting subnet nodes, and data-included subnet nodes are rewarded. If consensus isn't reached for a subnet, the validator is slashed, its penalty score incremented, and the subnet's epoch is skipped.

Subnet Validator Rewards

$$rewards+ = base\_rewards\_per\_epoch \times atestations/total\_validators$$

Subnet Validator Penalties

$$slash = effective\_balance \times slash\_percentage \times attestation\_percentage > max\_slash \rightarrow max\_slash$$

The *slash_percentage* **is**

$$slash\_percentage = 1.0 - attestation\_percentage$$

2.5 Rewards

All rewards are based on the memory requirements to validate a subnet. For example, a subnet utilizing a 400 billion parameters model will have higher compute requirements, thus requiring more nodes, and will generate greater rewards overall towards the subnet than a 100 million parameter model that requires fewer nodes to validate.

While a subnet with higher memory requirements will generate greater rewards and may incentivize users to create subnets with higher memory requirements, the higher the memory requirements result in a higher minimum nodes requirement, thus, the more TENSOR must be staked on-chain towards that subnet. If the minimum required subnet nodes are not reached during any epoch, consensus will not be accepted, rewards will not be generated, and the subnet will increment its penalty score.

Future Works Future implementations can include different categories including categories and subcategories of inference, training, model types, datasets, etc. to ensure the fine-tuning and accuracy of rewards and inflation.

2.5.1 Calculating Base Rewards

Subnet nodes and validators receive rewards when they are in consensus and the subnet is healthy. The value of the rewards in each epoch is calculated from the *based_reward*.

Each subnet's rewards mechanism is based on the default lowest expected memory per GPU (currently 16,000 MB, or 16 GiB). Each subnet must initialize with parameters including the required memory to validate the model.

The rewards for a subnet are based on the target nodes required, calculated from the subnet's memory requirements during the initialization and voting period. The target nodes are calculated from the minimum default memory per validator multiplied by the *target_subnet_nodes_multiplier*.

Minimum required subnet nodes:

$$min\_nodes = \frac{base\_min\_nodes}{subnet\_memory}$$

$$\begin{cases} \text{base\_subnet\_nodes}, & \text{if base\_min\_nodes} \\ \text{min\_subnet\_nodes}, & \text{otherwise} \end{cases}$$

Target Subnet Nodes:

$$target\_subnet\_nodes = min\_subnet\_nodes \times target\_subnet\_nodes\_multiplier + min\_subnet\_nodes$$

Subnet Rewards:

$$subnet\_rewards = reward\_per\_mb \times subnet\_mb$$

Subnet Node Reward:

$$subnet\_node\_reward = subnet\_node\_score / subnet\_scores\_sum \times subnet\_rewards$$

2.6 Subnet Delegate Staking

Users can delegate stake to subnets of their choosings in return for a portion of the subnets rewards. Users can participate in staking without the need to run their validator node. By becoming a subnet delegate staker, users can participate in governance and subnet voting without needing technical knowledge to run a validator node or subnet node.

Subnet Delegate Stake Rewards:

$$subnet\_delegate\_stake\_rewards = subnet\_rewards \times subnet\_delegate\_stake\_rewards\_percentage$$

2.6.1 Future Works

Subnet delegate staking can be used as a measurement for automating the removal of subnets that do not garner interest from the greater community, amongst other similar concepts.

2.7 Sequence Framework

Each subnet must use the sequence framework powered by blockchain validators. This sequence includes multiple steps to validate subnets, subnet nodes, and prevent attack vectors [Hypertensor, 2024]. This includes an idle state to allow for subnets to perform unique validation mechanisms such as PoI (proof of inference), PoS (proof of stake), PoT (proof of training), etc., an inclusion state for inducting new subnet nodes into the subnets consensus data, a submission state for requiring subnet nodes to begin validating and attesting consensus proposals and data, and an accountant state for subnet nodes to be enabled to create dishonesty proposals.

Each subnet can also use the provided signature validation mechanism to ensure each subnet node is staked on-chain.

The following is a framework for the blockchain and the subnets. Each step is multiple epochs and the blockchain validators update each subnet node sequence state on each new epoch.

2.7.1 Sequence

To be included as an inference-eligible node, each subnet node must stake the minimum required balance, and undergo a validation sequence, each determining its trust level.

Read more about why there is a sequence in the previous whitepaper. Each sequence step is multiple epochs in length.

1. Idle
2. Inclusion
3. Submission
4. Accountant

Idle New nodes staking to the blockchain and entering the subnets DHT are inducted in an idle state from the time they registered on-chain. In this state, the subnet node can do nothing but wait for other nodes to add them to their routing tables [Network Working Group, F. Baker, and Cisco Systems, 1995] [Network Working Group, F. Baker, P. Savola, and Cisco Systems, 2004] [Network Working Group, L. Yang, Intel Corp., R. Dantu, Univ. of North Texas, T. Anderson,Intel Corp., R. Gopal, Nokia, 2004] [Maymounkov and Maziéres, n.d.], and for blockchain validators to update their sequence step. During this step, they are not included in the consensus and do not receive rewards.

During the idle stage, subnets verify nodes by the consensus method or methods the subnet provides, such as proof of stake (PoS), proof of inference (PoI), or others, as explained in the Subnets section (seen in section 3).

Inclusion Once all nodes have validated the new node and added them to their routing tables and the required amount of epochs have gone by to include them in consensus, the subnet nodes performing validation will now include the new subnet node in consensus data to induct them into the blockchain. In this stage, the subnet node does not receive rewards.

The inclusion stage requires that each inclusion-eligible subnet node be included in consensus or will be removed if they surpass the *MaxSequentialAbsentSubnetNode*, which is the maximum count of sequential epochs any one subnet node can be out of consensus.

The length of epochs of the inclusion step is always equal to or greater than the *MaxSequentialAbsentSubnetNode*. This ensures they are never able to validate or attest to consensus if they have not been inducted via a consensus.

Submission At this stage of the sequence, they have been inducted into the blockchain via a consensus. The submission stage enables the node to begin submitting consensus data, attesting consensus data, and receiving rewards. Each node at this level is required to submit consensus data if they are chosen to be on that epoch, as well as each submission-eligible node must attest valid consensus data on each epoch.

Accountant As an Accountant, subnet nodes are enabled to initiate proposals to remove dishonest nodes (seen in section 2.8).

Conclusion This sequence is dictated by blockchain validators with each subnet node automatically pushed up in the sequence based on the epochs in each step. It's up to each subnet to verify the legitimacy of each subnet node and ensure only valid subnet nodes are included in the consensus. Each subnet must have its own validation mechanism for including subnet nodes in consensus data.

2.8 Fault Proofs

This mechanism acts as an immutable consensus state subnets can use to form a consensus on the honesty of its nodes.

2.8.1 Framework

This is designed for decentralized peer-to-peer subnets to validate and form a consensus on subnets' unique data to the blockchain. To ensure the security, integrity, and proper functioning of a subnet, each one must be able to remove validators others deem dishonest via consensus.

2.8.2 Proposals

Proposals are the core of the validation framework and are responsible for the forming of consensus to remove dishonest subnet nodes that can be performed by accountant-eligible subnet nodes.

The proposal framework is similar to fault proofs [Optimism, 2024] [ethereum-optimism, n.d.] used in Optimism, an Ethereum optimistic-rollup. The mechanism is similar whereby the plaintiff (the disputer deeming the defendant dishonest) and the defendant (the node deemed dishonest) must each stake as a monetary balance to incentivize honesty that each node can lose if found guilty. The plaintiff must submit a proposal with data proving the validity of their claim for others to attest to.

When a plaintiff proposes another subnet node as dishonest, the defendant must dispute the claim by staking or be removed. Once the defendant challenges the dispute by staking to the proposal, the proposal is activated for others to validate the submitted proposal data, vote, and form a consensus.

2.8.3 Voting

Once a defendant disputes the claim that they are dishonest, other accountant-eligible nodes can begin validating the proposal data. This validation is done off-chain within the subnet and each subnet has its own unique validation system and can have multiple of them. For example, a subnet specializing in inference may have a PoI (proof of inference) validation mechanism to validate the data. Once the subnet node has completed vetting the data, the subnet node can vote yay if they are in agreeance with the plaintiff, or nay if they are in agreeance with the defendant.

2.8.4 Completing

To complete the proposal and have either the defendant removed if the plaintiff wins, or do nothing if the defendant wins, a quorum and consensus must be met. Otherwise, nothing happens and the bids are returned to both the plaintiff and the defendant as the quorum or a consensus was unable to be reached.

2.8.5 Distribution

If the quorum is met and a consensus is reached, the winner (plaintiff or defendant) receives their initial bid back to them, and the loser will lose their bid to distribution. The subnet nodes that voted and are in consensus will be equally distributed the loser's bid. In cases where the distribution isn't equally divisible, the remaining dust is settled to the winner.

2.8.6 66% Attack

To control the voting power of a proposal a subnet node must own ¿66% of the total stake balance within the subnet the proposal is for. Each proposal to remove a subnet node requires a consensus threshold of 66%. A user can prevent other nodes they own from being removed by owning ¿33% of the subnet stake balance not including the one being proposed to be removed. A subnet node cannot vote on a proposal if they are the node being proposed to be removed. By not allowing both the proposer and the defendant to vote on the proposal, this further prevents the 66% threshold from being reached.

3 Subnets

This is an introductory explanation of the subnet-LLM repository. The blockchain itself is a modular artificial intelligence framework for distributed processing, peer-to-peer, software. This allows the blockchain to adapt to multiple AI categories such as image diffusion, AI Agents, Convolutional Neural Networks, Recurrent Neural Networks, Federated Training, etc.

3.1 Introduction

The core of the subnets-llm repository is a combination of Hivemind [team, 2020] [Ryabinin and Gusev, 2020], Petals [Borzunov et al., 2023b] [Borzunov et al., 2023a], LibP2P [libp2p, n.d.], PyTorch [pytorch, n.d.], and other Python libraries for artificial intelligence, encryption, decentralization, etc. with further adaptions for scaling decentralization. The innovations and modifications explained in this paper in both the blockchain and subnets sections include PoI (proof of inference), validation, consensus, server scoring, rewards, PoS (proof of stake), sequencing, etc.

Subnets are a global network for running decentralized AI.

3.1.1 Future Works

Due to the modular architecture and flexibility of Hypertensor, it's not restricted to only LLMs (large language models). While it may start with LLM subnets, the system is capable of integrating new subnets for different types of tasks. For example, it could incorporate subnets for computer vision models, reinforcement learning agents, image-diffusion, or even graph neural networks (GNNs) for specialized applications or autonomous systems.

Cross-subnet collaboration is also a possible evolution. As Hypertensor evolves, subnets can collaborate, forming a more powerful and interconnected AI system. For example, an LLM subnet could interact with a vision subnet to interpret and describe images, or with a reasoning subnet for decision-making tasks, unlocking new capabilities and possibilities.

The possibilities of Hypertensor are limitless.

3.2 Decentralized Inference

When generating tokens, a client stores the model's token embeddings (which typically comprise a small fraction of the total parameter count and can fit in RAM in most modern laptops, servers, and workstations) locally and relies on servers to run Transformer blocks. Each server holds several consecutive blocks, the number of which depends on the server's available GPU memory. Before each inference session, the client finds a chain of servers that collectively hold all model layers. Once the chain is formed, the client uses the local embedding layer to look up embedding vectors for prefix tokens, then sends those vectors to servers and receives new representations. Once the client obtains the outputs of the final block, it computes the next token probabilities and repeats this process. While the session is active, servers store attention keys and values from past client inputs and use them for subsequent inference steps. Clients also store past inputs to each server so that if any server fails or goes offline, another one can quickly take its place.

3.3 Client-side API

inference session iteratively takes inputs as PyTorch tensors runs them through all Transformer blocks, and returns final representations as PyTorch tensors. Under the hood, sessions form server chains, hold cache, and recover from server failures in a way that is transparent to the user.

3.4 Server Load Balancing

We ensure that servers are distributed evenly among Transformer blocks. Formally, servers maximize the total model throughput by choosing the blocks with the worst throughput and eliminating potential bottlenecks. Each server periodically announces its active blocks to a distributed hash table (Maymounkov and Mazieres, 2002). When a new server joins, it uses this information to identify an interval of blocks that contains

the most blocks with the worst throughput. This interval is always contiguous since splitting it would harm the inference latency. Once the server has selected its layers, it measures its own throughput (both network and compute) and announces it to the distributed hash table. Since peers may leave or fail at any time, all nodes periodically check if launching a rebalancing procedure would significantly improve the overall throughput. If this is the case, they switch layers until the throughput becomes near-optimal. In particular, if all peers serving certain blocks suddenly leave the system, this procedure quickly redistributes the remaining resources to close the emerged gaps.

3.5 Client-Side Routing

We want clients to be able to find a sequence of servers that run the model in the least amount of time. During generation, clients process one or few tokens at a time; in practice, the inference time is mostly sensitive to the network latency. Thus, clients have to ping nearby servers to measure latency and then find the path with minimal time via beam search. Conversely, during fine-tuning one needs to process a batch of examples in parallel. Here, clients can split their batches between multiple servers using the algorithm from Ryabinin et al. (2023). If a server fails during training or inference, a client removes it from consideration and reruns routing to find a replacement. During inference, the client sends all previous inputs to the replacement server, so that it has the same attention keys and values.

3.6 Security

Servers may turn out to be faulty and return incorrect outputs instead of the actual results of forward and backward passes. This may happen due to a malicious intent to influence other people's outputs or to earn a reward for serving layers without actually performing the calculations.

Hypertensor addresses these issues with an economically motivated approach combined with PoI (proof of inference) (seen in section 3.7), a penalization approach using a fault-proof mechanism via Subnet Democracy (seen in section 2.3), a validation and attestation framework (seen in section 2.4), and a PoS (proof of stake) consensus method (seen in section 3.8).

3.7 Proof of Inference (PoI)

In decentralized and trustless environments, ensuring the integrity of AI inferences is crucial, this can apply to training and other AI concepts. Without a mechanism like PoI, it would be difficult to trust the predictions made by a network of nodes. Proof of Inference thus provides a trust layer, allowing decentralized AI systems to operate with accountability and reliability.

Proof of Inference provides a mechanism to verify that a node in a network has correctly performed an inference or training task (e.g., a prediction or classification) using the agreed-upon model. It ensures that nodes do not return incorrect results or try to cheat

the system.

Similar to Proof of Stake or Proof of Work, PoI involves validation steps where other nodes or validators check the inference result. PoI validates the inference of the validating node and nodes are incentivized to provide accurate inferences and can be penalized or rewarded based on their behavior.

3.7.1 Mechanism

This is the first generation and proof of concept for PoI in a Hypertensor subnet. This is a high-level explanation.

On each epoch, blockchain validators choose one or multiple subnet validators using VRF (verifiable random function) to become the Accountant(s) of the epoch. Accountants are required to upload validation data of the subnet.

Accountants run inference or training validation logic to verify the accuracy of other nodes' inferences or training and put the results through a fitness function [Fitness function, n.d.] to ensure the accuracy of output tensors. A Tensor is a N-dimensional Matrix.

Each Accountant will find the minimum range of transformer blocks to validate:

$$rng = ([min..max] = min, min + 1..max - 1, max)$$

Once this range is calculated, the Accountant will run these transformer blocks individually in a sequence and cache the results locally for future use, or use previously cached results. An Accountant should cache multiple differing versions of a sequence to use at a later date or run randomized versions to ensure other nodes cannot learn the specific Accountant's expected outputs ahead of time.

Caching the sequence decreases the post-accountant sequence computations for validating other nodes by $\frac{n-1}{n}$.

Once the entire calculated range of blocks is cached, each subnet node will be validated by the validating node (Accountant) on one or multiple transformer blocks they serve. The subnet node is then injected into a new inference sequence alongside the cached results. This data is then validated across each position from the sequence, checking the absolute and relative tolerances, ensuring the maximum threshold constants are not surpassed.

Validate tensor tolerances on each position:

$$[pvals, ..., pvals] = [|input - other \leq atol + rtol \times other|, ..., |input - other \leq atol + rtol \times other|]$$

Validate validated positions:

$$valid = VTOL \leq vtol = \frac{[pvals\_true, ..., pvals\_true]}{len(pvals)}$$

3.7.2 Data Submission

This data is then submitted to the blockchain for others to verify. The data can also be used to submit a dishonesty proposal to have a dishonest node removed using a fault-proof mechanism explained in Proposals (seen in section 2.8).

3.7.3 Conclusion

Each Accountant is using its own inference data to validate other nodes. Combined with being validated and being a validator (Accountant), it is unexact to find a dishonest Accountant and or node in the network.

3.8 Proof of Stake (PoS)

Each subnet node must stake a minimum required balance to the blockchain with its PeerID and generate a signature with the encoded PeerID for other subnet nodes to validate.

Each node will validate the signature and proof of stake before adding them to their routing tables. This mechanism is always running in the background, therefor if a node removes itself or by consensus, other nodes will remove the removed node from their routing pools.

The signing algorithm used is the Ed25519 algorithm [paritytech, n.d.] [Internet Research Task Force (IRTF), S. Josefsson, SJD AB, I. Liusvaara, n.d.]. Ed25519 is an elliptic curve signing algorithm using EdDSA and Curve25519 [T. Pornin, n.d.] [National Institute of Standards and Technology, n.d.]. Ed25519 is chosen to sync with the blockchain signing algorithm. This enables the subnet signatures to be verified on-chain by other subnet nodes.

Ed25519 is the EdDSA signature scheme using SHA-512 (SHA-2) and Curve25519[2] where:

- $q = 2^255 - 19$,
- E/F_q is the twisted Edwards Curve

$$-x^2 + y^2 = 1 - 121665/121666x^2y^2,$$

- $\ell = 2^252 + 27742317777372353535851937790883648493$ and $c = 3$
- B is the unique point in $E(F_q)$ whose y coordinate is $\frac{4}{5}$ and whose x coordinate is positive. "positive" is defined in terms of bit-encoding:
  - "positive" coordinates are even coordinates (the least significant bit is cleared)
  - "negative" coordinates are odd coordinates (the least significant bit is set)
- H is SHA-512, with $b = 256$.

The curve E/F_q is birationally equivalent to the Montgomery curve known as Curve25519.

The equivalence is:

$$x = \frac{u}{v}\sqrt{-486664}, y = \frac{u-1}{u+1}$$

The signatures can be validated on-chain and off-chain against blockchain storage data to ensure the Substrate key pairs used to generate the signatures are valid [Substrate, n.d.] [Parity, n.d.].

Validate On-chain

```
pub fn validate_signature(
    data: &Vec<u8>,
    signature: &T::OffchainSignature,
    signer: &T::AccountId,
) -> DispatchResult {
    if signature.verify(&**data, &signer) {
        return Ok(())
    }


    let prefix = b"<Bytes>";
    let suffix = b"</Bytes>";
    let mut wrapped: Vec<u8> = Vec::with_capacity(data.len() +
        prefix.len() + suffix.len());
    wrapped.extend(prefix);
    wrapped.extend(data);
    wrapped.extend(suffix);
    ensure!(signature.verify(&*wrapped, &signer), Error::<T>::
       WrongSignature);

    Ok(())
}
```

3.9 Server Scoring

Each subnet node is scored accurately based on its computation and validation of the subnet. Subnet nodes are scored based on how many layers the subnet node is serving. These scores are submitted during the validation on each epoch (seen in section 2.4).

To incentivize higher computing capabilities and disincentivize a hostile takeover, subnet nodes serving a greater proportion of layers compared to the sum of all layers served will yield greater rewards. It is more profitable to run fewer nodes with higher computing servers than to run more nodes with lower computing servers. This further incentivizes a game theory for subnet nodes to compete for higher-performing servers and can theoretically create a low deviation near the most optimal measurement of computation, thus increasing subnet computation capabilities on a linear time horizon.

3.9.1 Scoring

The overall score for each subnet node is based on a percentage of computation in the subnet as:

Score:

$$score = k \times share \times share + share$$

Where *share* is:

$$\frac{node\_layers}{layers\_sum}$$

The blockchain itself is agnostic to the scoring mechanism. The sum of scores isn't required to be 100% in sum (seen in section 2.5).

In peer-to-peer networks, there are direct peers and relay peers. In summary, relay peers use a transport protocol to route traffic between two peers over a third-party "relay" peer. Relay peers are unreliable compared to direct peers, therefor to incentivize subnet nodes to become direct peers, relay peers' scores are lowered by 33%. Subnets have the ability to disallow relay peers.

Future Work For higher accuracy in scoring individual subnet nodes, a consensus state integrated directly into each subnet can be developed to gather greater detail of server information on data points such as throughput, memory contribution, availability, inference, and more.

3.10 Censorship Resistant

Control and decision-making are distributed across a network of participants rather than concentrating it on a single central authority. This structure makes it difficult for any one entity to gain control over the entire network.

3.11 Fault Tolerant

Subnets are inherently fault-tolerant because they distribute data, control, and operations across multiple nodes or participants, rather than relying on a single centralized server or point of control. This design ensures that even if some components of the system fail or are compromised, the overall system can continue functioning without interruption. As long as all of the subnet AI model layers are being served by subnet nodes, the subnet will continue to operate in a healthy state.

3.12 Permissionless

Anyone can join, participate, and contribute to the network without needing approval from a central authority or intermediary. This contrasts with permissioned networks, where participants must obtain explicit access or meet specific requirements to engage with the system. As long as a subnet node is staked, it can join other subnet node routing tables and begin running the subnet software.

3.13 Conclusion

This section introduces the first generation of the decentralized text-generation subnet for Hypertensor. Using this repository, thousands of subnets can be deployed with the thousands of open-sourced LLMs available by any platform participants.

References

Alistair Stewart, Eleftherios Kokoris-Kogia (2020). *GRANDPA: a Byzantine Finality Gadget*. URL: https://arxiv.org/pdf/2007.01560.

Borzunov, Alexander et al. (2023a). "Distributed inference and fine-tuning of large language models over the Internet". In: *Advances in Neural Information Processing Systems*. Vol. 36, pp. 12312–12331. URL: https://arxiv.org/abs/2312.08361.

Borzunov, Alexander et al. (2023b). "Petals: Collaborative Inference and Fine-tuning of Large Models". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 558–568. URL: https://arxiv.org/abs/2209.01188.

ethereum-optimism (n.d.). *OP Mainnet Security Model*. URL: %5Curl%7Bhttps://github.com/ethereum-optimism/docs/blob/main/pages/chain/security/faq.mdx%7D.

Fitness function (n.d.). *Fitness function — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Fitness_function.

Foundation, Web3 (n.d.). *GRANDPA: a Byzantine Finality Gadget*. URL: %5Curl%7Bhttps://github.com/paritytech/substrate/blob/master/primitives/core/src/ed25519.rs%20%7D.

Handan Kilinc Alper (2020). *BABE*. URL: https://research.web3.foundation/Polkadot/protocols/block-production/Babe.

Hypertensor (2024). *Hypertensor Whitepaper V1*. URL: https://hypertensor.org/whitepaper-v1.pdf.

Internet Research Task Force (IRTF), S. Josefsson, SJD AB, I. Liusvaara (n.d.). *Edwards-Curve Digital Signature Algorithm (EdDSA)*. URL: https://datatracker.ietf.org/doc/html/rfc8032.

libp2p (n.d.). *libp2p*. https://github.com/libp2p/libp2p.

Maymounkov, Peter and David Maziéres (n.d.). "Kademlia: A Peer-to-peer information system based on the XOR metric". In: *arXiv preprint arXiv:1409.0473* (). URL: https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf.

National Institute of Standards and Technology (n.d.). *Digital Signature Standard (DSS)*. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

Network Working Group, F. Baker, and Cisco Systems (1995). *Requirements for IP Version 4 Routers*. URL: https://datatracker.ietf.org/doc/html/rfc1812.

Network Working Group, F. Baker, P. Savola, and Cisco Systems (2004). *Ingress Filtering for Multihomed Networks*. URL: https://datatracker.ietf.org/doc/html/rfc3704.

Network Working Group, L. Yang, Intel Corp., R. Dantu, Univ. of North Texas, T. Anderson,Intel Corp., R. Gopal, Nokia (2004). *Forwarding and Control Element Separation (ForCES) Framework*. URL: https://www.ietf.org/rfc/rfc3746.txt.

Optimism (2024). *Fault Proofs Explainer*. URL: https://docs.optimism.io/stack/protocol/fault-proofs/explainer.

Parity (n.d.). *Pair*. URL: https://paritytech.github.io/polkadot-sdk/master/sp_core/crypto/trait.Pair.html.

paritytech (n.d.). *ed25519*. URL: %5Curl%7Bhttps://github.com/paritytech/substrate/blob/master/primitives/core/src/ed25519.rs%20%7D.

pytorch (n.d.). *pytorch*. https://github.com/pytorch/pytorch.

Ryabinin, Max and Anton Gusev (2020). "Towards Crowdsourced Training of Large Neural Networks using Decentralized Mixture-of-Experts". In: *Advances in Neural Information Processing Systems*. Vol. 33. URL: https://proceedings.neurips.cc/paper/2020/file/25ddc0f8c9d3e22e03d3076f98d83cb2-Paper.pdf.

Substrate (n.d.). *Key and Network Operations*. URL: https://docs.substrate.io/deploy/keys-and-network-operations.

T. Pornin (n.d.). *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)*. URL: https://www.rfc-editor.org/rfc/rfc6979.html.

team, Learning@home (2020). *Hivemind: a Library for Decentralized Deep Learning*. https://github.com/learning-at-home/hivemind.