Journal of Information Systems Engineering and Management
2025, 10(63s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

Consensus-Based Deployment Protocol for Distributed LLM Adapter Management

Sivaramakrishnan Vaidyanathan
University of Cincinnati, USA

ARTICLE INFO

Received: 01 Nov 2025

Revised: 05 Dec 2025

Accepted: 14 Dec 2025

ABSTRACT

Parameter-efficient fine-tuning techniques enable rapid model customization through lightweight adapter artifacts; however, the distributed deployment of these artifacts introduces critical consistency challenges. Standard orchestration strategies create temporal windows where heterogeneous model versions serve production traffic simultaneously, exposing users to unvalidated or degraded model behavior. The article presents a consensus-based deployment protocol addressing fundamental gaps in continuous fine-tuning infrastructure. The protocol partitions inference clusters into validation and serving planes, establishing a clear separation between experimental evaluation and production workloads. A quorum-gated state machine controls adapter propagation, requiring majority agreement among validation nodes before authorizing cluster-wide synchronization. The beta barrier mechanism prevents premature deployment by algorithmically blocking inference nodes from loading new adapters until validation completes successfully. Formal verification demonstrates strong safety guarantees where no production request reaches untested model versions, regardless of timing variations or partial system failures. The protocol achieves eventual convergence to a uniform cluster state while tolerating minority node failures through majority voting requirements. Theoretical analysis proves monotonic progression between stable configurations without indefinite divergence or version oscillation. The architecture complements existing serving engines by functioning as a control plane protocol governing adapter lifecycle management across distributed inference fleets.

Keywords: Distributed Consensus Protocol, Parameter-Efficient Fine-Tuning, Model Deployment Safety, Quorum-Based Validation, Adapter Lifecycle Management, Inference Cluster Consistency

Introduction

The emergence of low-rank adaptation and related parameter-efficient fine-tuning methods has fundamentally altered the operational paradigm for large language model deployment. Low-rank adaptation represents a breakthrough in model customization by decomposing weight update matrices into low-rank representations [1]. The technique enables fine-tuning through the injection of trainable rank decomposition matrices into each layer of the transformer architecture. Full fine-tuning of large language models poses significant computational challenges and storage requirements. Parameter-efficient approaches address this limitation by updating only a small subset of model parameters while freezing the pre-trained weights.

Journal of Information Systems Engineering and Management
2025, 10(63s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

The low-rank adaptation methodology introduces pairs of trainable matrices that approximate the weight updates during fine-tuning [1]. These adapter matrices maintain substantially smaller dimensions compared to the original weight matrices. The approach preserves model quality while dramatically reducing the memory footprint required for task-specific customization. Organizations can now update massive foundation models by exchanging these compact weight matrices rather than redeploying entire parameter sets. Such a change makes it possible to have very frequent update cycles that were, until now, unthinkable because of limitations in storage, bandwidth, and version management. The scenarios of continuous fine-tuning of pipelines in which adapter artifacts are updated very frequently in development, staging, and production environments raise new infrastructure issues.

Contemporary distributed serving systems optimize for inference throughput and memory efficiency through advanced batching strategies and attention mechanism optimizations. Modern serving architectures address the memory bottleneck inherent in large language model deployment. PagedAttention introduces a memory management approach inspired by virtual memory and paging techniques from operating systems [2]. The method partitions key-value cache tensors into blocks rather than storing them contiguously. Each block contains attention keys and values for fixed numbers of tokens. This block-based organization allows non-contiguous memory allocation and enables flexible sharing of cached computations across multiple requests.

The serving systems demonstrate the capability to handle thousands of concurrent inference requests while maintaining low latency through memory-efficient attention kernels and dynamic batching algorithms [2]. However, these architectures delegate deployment orchestration to external controllers that operate independently of the serving layer's consistency guarantees. Standard controllers implement rolling update strategies that gradually replace cluster nodes with new model versions. Updates typically process in sequential batches, affecting portions of the total cluster capacity. This approach creates a convergence gap where user requests are load-balanced across nodes running incompatible model logic. Different cohorts of users potentially receive responses generated by adapter versions separated by multiple training iterations.

The situation becomes critical when adapter versions contain regressions such as hallucination patterns, where models generate factually incorrect responses with high confidence. Factual inaccuracies in domain-specific knowledge or safety failures that bypass alignment constraints become visible to production users before detection mechanisms can intervene. During typical rolling update windows affecting distributed clusters, substantial user populations encounter new versions before deployment completes. This exposes users to potentially degraded model behavior. The core problem lies in the absence of a validation barrier between artifact publication and production exposure. Current architectures lack principled mechanisms to enforce that all nodes validate adapter quality before any node serves user traffic with that adapter. The validation gap is particularly acute in continuous fine-tuning scenarios where adapter updates occur at high frequencies. This work addresses the gap by introducing a distributed consensus protocol specifically designed for adapter deployment in inference clusters.

Methodology Component	Implementation Details	Evaluation Criteria	Node Participation	Asynchronous Operation	Verification Mechanism
Regression Test Suites	Perplexity measurements on reference corpora	Language modeling fluency and coherence	Independent execution per consensus node	Each node proceeds at its own pace	Statistical thresholds for quality
Reasoning Assessment	Question answering	Genuine comprehensi	Parallel evaluation	No coordination	Accuracy against

	requiring multi-hop inference	on versus pattern matching	across the consensus group	overhead	golden datasets
Safety Evaluation	Prompts testing alignment and ethical constraints	Harmful output detection and policy adherence	Distributed validation workload	Varies by computational resources	Binary pass-fail determinatio n
Vote Broadcasting	Binary decision propagation after validation	Aggregate synthesis from multiple criteria	Each consensus node casts a single vote	Completes upon local validation finish	Replicated log accumulation
Cryptographic Verification	Hash-based integrity checking during propagation	Digest matching against the state register entry	All inference nodes before loading	Sequential after download completion	Secure hash algorithm comparison
Atomic Hot-Swap	Zero-downtime adapter replacement	Seamless transition without service interruption	Individual inference node operation	Independent synchronization timing	Runtime configuration change

Table 1. Validation and Deployment Methodology, Comprehensive Testing Framework, and Synchronization Protocol [1, 2].

Related Work and Methodology

Related Work

Parameter-efficient fine-tuning has transformed model customization practices by enabling targeted weight updates without full retraining overhead. Low-rank adaptation decomposes weight matrices into trainable low-rank representations that approximate full fine-tuning results [1]. The technique reduces memory footprint and storage requirements while preserving model quality across diverse tasks. Quantized low-rank adaptation extends these benefits by applying quantization techniques to further compress adapter artifacts. These advances democratize fine-tuning for resource-constrained environments but focus primarily on training efficiency rather than deployment consistency.

Distributed consensus algorithms provide foundational mechanisms for coordinating state across replicated systems. Leader-based consensus protocols simplify coordination by centralizing decision authority in elected leader nodes [3]. Followers replicate leader decisions through append-only logs that maintain operation ordering. Quorum-based voting enables fault tolerance by requiring only majority agreement for state transitions. Byzantine fault-tolerant protocols extend these guarantees to handle arbitrary failures, including malicious behavior [4]. Replicas authenticate messages through cryptographic signatures and build agreement certificates through multi-phase communication patterns. However, existing consensus work addresses general state machine replication without considering expensive validation operations characteristic of machine learning deployments.

Large language model serving systems optimize inference throughput through memory management innovations and batching strategies. PagedAttention partitions key-value cache tensors into blocks, enabling non-contiguous memory allocation [2]. Block-based organization allows flexible cache

Journal of Information Systems Engineering and Management
2025, 10(63s)
e-ISSN: 2468-4376
https://jisem-journal.com/ **Research Article**

sharing across concurrent requests. Dynamic batching algorithms group requests to maximize GPU utilization while maintaining latency constraints. Runtime configurable architectures enable adaptation between performance and resource consumption objectives [6]. These serving engines focus on request processing efficiency but delegate deployment orchestration to external control planes lacking integration with model validation workflows.

Methodology and Contributions

The article addresses the deployment consistency gap through architectural and protocol-level innovations. The methodology establishes dual-plane infrastructure partitioning that isolates validation workloads from production serving obligations. Consensus nodes form a minority group maintaining authoritative cluster state through log-based replication. Inference nodes constitute the majority fleet processing user requests in a read-only relationship with the consensus state. This separation prevents validation risks from impacting production traffic while enabling experimental evaluation under realistic inference conditions.

The protocol introduces a quorum-gated state machine governing adapter lifecycle transitions. Three discrete states encode deployment phases: stable mode indicating validated production operation, beta testing mode signaling ongoing validation, and release commit mode authorizing cluster-wide propagation. State transitions require consensus write operations replicated across validation nodes before acknowledgment. The beta barrier mechanism implements the critical safety property by algorithmically preventing inference nodes from loading adapters until a majority validation consensus is achieved. Conditional logic gates synchronization on explicit release commit signals rather than implicit artifact availability.

Distributed validation operates asynchronously across consensus nodes executing independent test suites against candidate adapters. Validation workloads assess multiple dimensions, including reasoning capabilities [5], retrieval grounding accuracy, and safety constraint adherence. Each consensus node broadcasts binary votes upon completing evaluation. Votes accumulate in the replicated log, creating auditable consensus formation records. Quorum achievement triggers atomic state transitions that lift deployment barriers.

The convergence mechanism implements eventual consistency through polling-based synchronization. Inference nodes periodically query cluster state registers, detecting release commit transitions. Synchronization sequences download adapter artifacts, verify cryptographic digests, and execute atomic hot-swap operations [6]. The methodology ensures monotonic progression between stable configurations without indefinite divergence.

Key contributions include formal safety guarantees proving invalid adapters never reach production infrastructure, convergence properties ensuring eventual cluster uniformity, and fault tolerance characteristics tolerating minority node failures. The protocol adapts distributed consensus patterns to machine learning operations by incorporating expensive validation phases into commitment criteria rather than simple state persistence.

Dual-Plane Architecture

Infrastructure Partitioning

The proposed architecture establishes two functionally isolated node groups within the inference cluster through deliberate separation of concerns. The consensus group comprises a minority of nodes configured with odd cardinality to enable majority voting without tie scenarios. Consensus protocols fundamentally rely on leader-based coordination where a single node assumes responsibility for ordering operations [3]. The leader receives proposals from clients and ensures consistent replication across follower nodes. This model simplifies the coordination problem by centralizing decision authority while maintaining fault tolerance through leader election mechanisms.

The architecture adopts principles from distributed consensus algorithms that have evolved to balance understandability with correctness guarantees. Raft-style consensus emphasizes clear separation between leader election, log replication, and safety properties [3]. The leader maintains an authoritative log of state transitions. Followers replicate this log and apply transitions in identical order. The consensus group maintains a replicated cluster state register that serves as the single source of truth for global configuration. The register undergoes continuous synchronization across consensus members through append-only log replication.

The inference fleet constitutes the majority of cluster capacity and handles all production traffic. These nodes operate in a read-only relationship with the cluster state register. The asymmetric relationship prevents inference nodes from initiating state transitions or participating in consensus decisions. Quorum-based protocols require only a majority of replicas to agree on state changes [3]. This approach tolerates minority failures without blocking progress. The consensus group typically comprises small odd-numbered configurations. Five-node consensus groups tolerate two simultaneous failures while maintaining operational continuity. The inference fleet scales independently based on traffic demands and throughput requirements.

This partitioning fundamentally separates validation risk from serving obligations through architectural boundaries. Consensus nodes can execute potentially unstable model versions without impacting user-facing services. The isolation ensures that experimental adapter evaluation occurs in controlled environments distinct from production workloads. Inference nodes gain protection against premature adapter adoption through state-based access controls enforced by the consensus layer.

State Register Design

The cluster state register maintains linearizable consistency across consensus nodes through log-based replication mechanisms. State machine replication ensures that all non-faulty replicas execute operations in identical order and reach identical states [4]. Byzantine fault tolerance extends this model to handle arbitrary failures, including malicious behavior. Replicas must reach an agreement despite the potential presence of compromised nodes that send conflicting messages or deviate from protocol specifications.

Each register entry encapsulates three critical components that collectively define cluster configuration state. A version identifier uniquely addresses the adapter artifact using content-addressable storage principles. The identifier enables precise artifact retrieval without ambiguity across distributed storage systems. A state flag controls cluster-wide synchronization behavior through enumerated values that encode deployment phases. Cryptographic techniques provide integrity verification during artifact propagation across network boundaries [4]. Digital signatures allow replicas to authenticate messages and detect tampering. Hash chains create verifiable histories of state transitions that prevent retroactive manipulation.

The state flag implements a finite state machine with three discrete transitions governing deployment progression. The stable state indicates a validated production operation where the entire cluster serves a unified adapter version. The beta testing state signals that consensus nodes are evaluating a new version, while inference nodes must maintain their current configuration. The release commit state authorizes inference nodes to synchronize with the new version. State transitions occur atomically through consensus write operations that replicate across the consensus group before acknowledgment. Fast Byzantine protocols optimize this process through parallel communication patterns and speculative execution [4]. Replicas exchange messages in phases to build agreement certificates without waiting for sequential leader confirmations.

Node Type	Cardinality	Primary Function	State Relationship	Traffic Type
Consensus Group	Odd-numbered minority configuration	Maintains cluster state register through log-based replication	Read-write access with atomic broadcast semantics	Validation workloads only
Leader Node	Single elected member within a consensus group	Coordinates operation ordering and log replication	Authoritative log maintenance	Beta testing execution
Follower Nodes (Consensus)	Remaining consensus members	Replicate leader decisions and participate in voting	Log replication and application	Validation and voting
Inference Fleet	A large majority of cluster capacity	Process production user requests	Read-only polling of the state register	End-user production traffic
State Register	Replicated across consensus nodes	Single source of truth for global configuration	Linearizable consistency through replication	State transition coordination

Table 2. Dual-Plane Architecture Components Node Roles and Responsibilities in Distributed Inference Clusters [3, 4].

Protocol Mechanism

Artifact Ingestion and Isolation

Upon receiving a new adapter artifact, the consensus group initiates a state transition by replicating an updated register entry across its members. The replication process follows atomic broadcast semantics to ensure all consensus nodes observe identical state changes. This entry assigns a unique version identifier to the artifact through content-addressable naming schemes. The identifier combines cryptographic hashes with sequence numbers to guarantee uniqueness across the artifact lifecycle. The entry sets the state flag to beta testing mode and records the artifact's cryptographic hash using secure hash algorithms.
The beta testing flag creates an absolute barrier preventing premature deployment. Inference nodes observing this state are algorithmically prevented from loading the new adapter regardless of its availability in storage systems. The barrier implements strict read-after-write consistency where state visibility does not imply execution authorization. Model evaluation demands rigorous testing across multiple dimensions. Question answering systems must demonstrate genuine reasoning capabilities rather than pattern matching against training data [5]. Simple retrieval-based approaches often fail when confronted with questions requiring multi-hop reasoning or complex inference chains.
This isolation property ensures validation occurs in sandboxed environments, completely separated from production request processing. Consensus nodes immediately download and activate the new adapter in their inference engines. The activation loads adapter weights into GPU memory and configures the inference runtime. However, this activation affects only validation workloads running on consensus nodes. Production traffic continues flowing to inference nodes serving the previously validated version. The temporal decoupling between artifact availability and production exposure forms the protocol's foundational safety mechanism.

Distributed Validation

Each consensus node independently executes validation workloads against the newly loaded adapter without coordination overhead. These workloads consist of regression test suites encompassing multiple evaluation methodologies. Perplexity measurements on reference corpora assess language modeling capabilities. Retrieval-augmented generation accuracy checks verify correct grounding behavior. Safety evaluation prompts test adherence to ethical guidelines and content policies.

Effective evaluation requires datasets that challenge genuine comprehension abilities. Questions demanding scientific reasoning expose limitations in models that rely primarily on statistical correlations [5]. Natural language understanding benchmarks must include items requiring knowledge retrieval, common sense reasoning, and logical inference. The validation suite composition directly impacts the protocol's ability to detect problematic adapter versions before production deployment. Validation operates asynchronously across consensus nodes, where each member proceeds at its own pace. Consensus nodes may complete validation at different times depending on GPU availability and test suite complexity. Upon completing validation, each node broadcasts a binary vote indicating whether the adapter meets quality thresholds. These votes accumulate in the replicated log, creating an auditable record of consensus formation.

Quorum-Based Release and Convergence

The protocol implements majority voting requirements for state transitions following quorum-based decision principles. A new adapter version progresses from beta testing to release commit status only when more than half of the consensus nodes cast affirmative votes. This mechanism provides fault tolerance by tolerating minority node failures without blocking the deployment of valid adapters. When quorum is reached, the state flag transitions to release commit, triggering cluster-wide propagation. Inference nodes continuously poll the cluster state register to detect configuration changes. Upon detecting the release commit status, nodes initiate synchronization sequences. The node downloads the new adapter artifact and verifies cryptographic digests. Following verification, the node performs atomic hot-swap operations, replacing active adapters without service interruption. Runtime reconfiguration enables dynamic model adaptation based on operational requirements [6]. Neural network architectures can adjust computational paths during execution to balance resource consumption against performance objectives. Configuration changes modify network topology or precision settings without requiring full system restarts. This convergence mechanism provides eventual consistency guarantees where all functional inference nodes synchronize within a bounded time.

Protocol Phase	State Flag Value	Consensus Node Behavior	Inference Node Behavior	Transition Condition	Safety Property
Artifact Ingestion	BETA_TESTING	Download and load the adapter into the GPU memory	Continue serving the previously validated version	New adapter upload received	Absolute deployment barrier active
Distributed Validation	BETA_TESTING	Execute regression test suites and broadcast votes	Acknowledge the new version without loading	Validation workloads executing	Production traffic is isolated from testing
Quorum Formation	BETA_TESTING	Accumulate votes in replicated log	Maintain current adapter configuration	Votes collected from all nodes	Auditable consensus record created
Quorum Achievement	RELEASE_COMMIT	Mark validation completion in the state register	Initiate synchronization sequence	Majority affirmative votes received	Deployment barrier lifted
Cluster Convergence	RELEASE_COMMIT	Continue serving with a validated adapter	Download, verify, and hot-swap the adapter	State flag transition detected	Eventual consistency achieved

Table 3. Protocol Phase Transitions State Machine Progression and Synchronization Barriers [5, 6].

Theoretical Properties

Safety Guarantees

The protocol's primary safety property states that no user request can be served by an unvalidated model version. This guarantee follows directly from the protocol's state machine design and enforcement mechanisms. For any adapter version to be loaded by an inference node, two conditions must be satisfied simultaneously. The cluster state register must indicate release commit status as the authoritative deployment signal. Achieving a release commit requires a majority consensus vote from validation nodes operating in the consensus group.

Consider an adapter that fails quality checks during the validation phase. By definition, at most a minority of consensus nodes will vote affirmatively for deployment. Formal methods provide rigorous approaches to verify that systems behave correctly under all possible conditions [7]. Model checking explores exhaustive state spaces to identify violations of safety properties. The technique has proven effective for detecting subtle bugs in distributed protocols that traditional testing methods miss.

Without a majority consensus, the state flag cannot transition to release commit status. The state machine remains in beta testing mode indefinitely until the adapter is either rejected or replaced by a new candidate version. Inference nodes operate under strict conditional logic that gates adapter loading on release commit status. These nodes continuously poll the state register, but will never observe the enabling condition for synchronization when an adapter fails validation. Formal specification languages enable precise definition of protocol behaviors and invariants [7]. Specifications serve as executable documentation that captures system requirements unambiguously.

The safety guarantee holds even under partial system failures. Consensus nodes may crash or experience network partitions during validation. The quorum requirement ensures that the system tolerates minority failures without compromising safety properties. A failed consensus node contributes neither positive nor negative votes. The remaining functional nodes must still achieve majority agreement to authorize deployment.

Convergence Properties

For valid adapter versions, the protocol guarantees eventual convergence to a uniform cluster state across all operational nodes. When an adapter successfully passes validation criteria, a majority of consensus nodes will eventually cast affirmative votes. This triggers the state transition to release commit, which inference nodes eventually observe through their polling mechanism.

Distributed systems face fundamental trade-offs between consistency, availability, and partition tolerance [8]. Eventual consistency relaxes immediate consistency requirements to improve availability and performance. The model guarantees that all replicas converge to identical states given sufficient time without conflicting updates. Client applications must tolerate temporary inconsistencies during convergence windows.

Convergence time is bounded by three factors operating in sequence. The maximum validation duration across consensus nodes determines when the final vote arrives. The polling interval configured for inference nodes controls how quickly they detect state changes. The network transfer time for adapter artifacts depends on artifact size and available bandwidth. Consistency models exist along a spectrum from strong to weak guarantees [8]. Strong consistency provides immediate visibility of updates across all replicas. Weak consistency allows for temporary divergence along with eventual reconciliation. The protocol guarantees that the cluster state changes in a sequential manner from one stable configuration to another without any indefinite divergence. This property prevents scenarios where different cluster regions settle on incompatible versions. The state register acts as a linearization point coordinating all state transitions.

Property Type	Guarantee Description	Enforcement Mechanism	Failure Tolerance	Time Bound	Consistency Model
Safety	No unvalidated adapter serves user requests	State machine conditional logic prevents loading	Tolerates minority consensus node failures	Unbounded during validation	Strong safety under all executions
Quorum Requirement	Majority consensus required for deployment	Vote accumulation with threshold checking	Continues operation despite minority crashes	Bounded by the slowest validation node	Majority voting decision
Version Isolation	Failed adapters never reach inference nodes	Beta barrier blocks synchronization attempts	Complete isolation from production	Permanent until replacement	Invalid artifacts contained
Convergence	All functional nodes reach an identical state	Polling-based synchronization with verification	Network partition delays but preserves the guarantee	Sum of validation, polling, and transfer times	Eventual consistency
Monotonic Progression	Cluster advances without version oscillation	State register linearization point coordination	Prevents retroactive version changes	Single-direction state transitions	Sequential configuration progression

Table 4. Theoretical Guarantees and Convergence Bounds, Safety Properties and Consistency Models [7, 8].

Conclusion

The consensus-based deployment protocol addresses critical infrastructure challenges emerging from continuous fine-tuning practices in large language model operations. Traditional rolling update strategies often fail to provide adequate protection against unvalidated adapter propagation, resulting in deployment gaps where production users inadvertently become involved in model testing. The dual-plane architecture introduced in the article fundamentally restructures deployment workflows by isolating validation risks from serving obligations through explicit node role separation.

The beta barrier mechanism represents a principled solution to the temporal coupling problem inherent in conventional orchestration approaches. By requiring explicit majority consensus before lifting deployment restrictions, the protocol transforms adapter availability into a necessary but insufficient condition for production exposure. The state machine design ensures atomic transitions between deployment phases while maintaining linearizable consistency across consensus group members. Cryptographic integrity verification during artifact propagation provides additional defense against corruption or tampering during network transit.

Theoretical analysis confirms the strong safety properties of invalid adapters, which remain permanently isolated within validation boundaries. The quorum voting mechanism tolerates minority node failures without compromising safety guarantees or blocking deployment of valid adapters. Convergence properties ensure that all functional inference nodes eventually synchronize within bounded time intervals, determined by validation duration, polling frequency, and network transfer characteristics. The protocol's fault tolerance characteristics derive from majority voting requirements adapted from distributed consensus algorithms.

Future extensions could incorporate Byzantine fault tolerance to handle malicious validation nodes attempting to authorize compromised adapters. Zero-knowledge proof integration would enable cross-organizational validation without exposing proprietary test datasets. Environmental considerations could influence validation scheduling, aligning compute-intensive testing with periods of favorable grid carbon intensity. The protocol establishes foundational patterns for safe adapter deployment applicable across diverse machine learning operational contexts requiring strict version consistency and deployment authorization controls.

References

[1] Edward Hu et al., "LoRA: Low-Rank Adaptation of Large Language Models," arXiv, 2021. [Online]. Available: https://arxiv.org/pdf/2106.09685v1/1000
[2] Woosuk Kwon et al., "Efficient Memory Management for Large Language Model Serving with PagedAttention," ACM, 2023. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3600006.3613165
[3] Heidi Howard and Richard Mortier, "Paxos vs Raft: Have we reached consensus on distributed consensus?" ACM, 2020. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3380787.3393681
[4] Ittai Abraham et al., "Revisiting Fast Practical Byzantine Fault Tolerance," arXiv, 2017. [Online]. Available: https://arxiv.org/pdf/1712.01367
[5] Peter Clark et al., "Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge," arXiv, 2018. [Online]. Available: https://arxiv.org/pdf/1803.05457
[6] Hokchhay Tann et al., "Runtime Configurable Deep Neural Networks for Energy-Accuracy Trade-off," ACM, 2016. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/2968456.2968458
[7] CHRIS NEWCOMBE et al., "How Amazon Web Services Uses Formal Methods," ACM, 2015. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/2699417
[8] Werner Vogels, "Eventually Consistent," Communications of the ACM, 2009. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/1435417.1435432