International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group| Volume 1, Issue 1, 93-101, 2020
ISSN: 3050-922X| https://doi.org/10.63282/3050-922X.IJERET-V1I1P111

Original Article

GenAI-Powered Test Case Generation for Microservices in CI/CD Pipelines via Trusted Federated Explainability

Mohan Siva Krishna Konakanchi
Independent Researcher, USA.

Abstract *- Microservices increase delivery velocity by enabling independent deployments, yet they also expand the surface area for regressions across APIs, message flows, and distributed data contracts. At enterprise scale, test assets and operational evidence are often siloed across teams, platforms, and regulatory boundaries, limiting the ability to build a unified learning loop for test generation. Meanwhile, generative AI (GenAI) methods can synthesize test cases from service specifications, code changes, and production telemetry, but their use in CI/CD must satisfy integrity and accountability requirements: generated tests must be explainable, reproducible, and robust against low-quality or malicious contributions. Additionally, teams require practical controls to manage the explainability–performance trade-off: highly interpretable generation strategies can be slower or less effective, while high-performing black-box generation may be difficult to justify in audits. This paper proposes T-FedTest, a GenAI-powered framework for automated test case generation for microservices within CI/CD pipelines, using a trust metric-based federated learning (FL) approach and federated explainability. T-FedTest intro- duces: (i) a trust metric that quantifies participant integrity and accountability using provenance attestations, update con- sistency, evaluation reliability, and policy compliance; (ii) a trust-aware federated aggregation protocol that limits poisoning and emphasizes high-accountability contributors; and (iii) an explainability–performance trade-off controller that allocates explanation budgets to generated tests and learning updates, enabling organizations to optimize auditability and runtime effec- tiveness without complex mathematics. We evaluate T-FedTest us- ing a controlled prototype simulation of multi-team microservice ecosystems with heterogeneous APIs, non-IID fault distributions, and adversarial/faulty participants. Results show that trust- aware federated learning improves fault-detection effectiveness and reduces harmful regressions compared to standard federated averaging baselines, while moderate explanation budgets preserve stable, actionable rationales with limited performance degra- dation. We conclude with deployment guidance for integrating trusted GenAI test generation into enterprise CI/CD.*

Keywords *- Microservices, CI/CD, Automated Testing, Test Generation, Generative AI, Federated Learning, Trust Metrics, Explainable AI, Software Governance.*

1. Introduction

Microservices architecture decomposes systems into inde- pendently deployable services that evolve at different cadences and interact through APIs and messaging. This decomposition improves team autonomy and delivery velocity, but it com- plicates testing because failures often manifest as emergent behaviors across service boundaries: schema drift, incompat- ible error handling, retries causing duplicate side effects, or inconsistent authorization logic. Industry practice recommends automation-first delivery with continuous delivery pipelines [1], and microservices literature highlights operational com- plexity and evolving integration contracts as persistent pain points [2]–[5].

Automated test generation can reduce the burden of writing and maintaining tests, especially for rapidly changing API surfaces. However, enterprise test generation must operate under organizational constraints:

- Silos and boundaries: Different product groups and regions maintain separate telemetry and test assets; cen- tralizing raw traces may be disallowed.
- Heterogeneity: Services vary in protocols (REST/gRPC), languages, data stores, and message buses.
- Integrity and accountability: Generated tests can break pipelines or introduce false confidence; learning loops must be robust to faulty or malicious updates.
- Explainability: Teams require defensible rationales for why a generated test exists and what requirement, risk, or incident it covers.

Recent GenAI foundations (sequence-to-sequence, atten- tion, and transformer architectures) enable structured text and code generation [19]–[21]. In software engineering workflows, these models can propose test inputs, expected outcomes, and edge cases, conditioned on API specifications, code diffs, and incidents. Yet, direct central training on proprietary logs is often infeasible. Federated learning (FL) offers a collaboration model where each team trains locally and shares updates rather than raw data [8], [9], [13]. Standard FL, however, assumes benign participants and provides limited support for accountability and explainability.

1.1. Problem Statement
We address three coupled problems in GenAI-powered test generation for microservices:
- P1: Cross-silo learning for test generation: How can teams collaborate to improve test generation without central-izing raw traces or internal specifications?
- P2: Integrity and accountability in federated test learn- ing: How can the learning loop resist poisoning, unstable updates, and low-quality contributions while maintaining au- ditability?
- P3: Explainability–performance trade-off: How can the organization quantify and optimize the trade-off between ex-plainable test rationales and high fault-detection effectiveness under CI/CD time budgets?

1.2. Contributions
This paper proposes *T-FedTest*, an end-to-end framework that contributes:
- A trust metric for federated GenAI test generation, grounded in provenance, update consistency, evaluation reliability, and policy compliance.
- A trust-aware federated aggregation protocol combin- ing trust gating with robust outlier resistance to reduce poisoning impact [11], [12].
- A federated explainability approach that generates stable, actionable rationales for tests (e.g., risk coverage and spec mapping) using explanation primitives [14]– [17].
- A trade-off controller using explanation budgets to optimize auditability versus pipeline throughput and fault detection.
- A prototype evaluation under realistic conditions: non- IID faults, heterogeneous services, and adversarial/faulty participants.

2. Background and Related Work
2.1. Microservices and CI/CD Testing Challenges
Microservices adoption is driven by team autonomy and scalable delivery [2], [3]. Empirical and grey-literature studies identify common problems: distributed debugging, contract drift, dependency management, and testing complexity [4], [5]. Continuous delivery emphasizes repeatable pipelines with au- tomated gates [1]; in microservices, these gates must validate not only unit correctness but also API contracts, backward compatibility, and operational readiness.

2.2. Automated Test Generation
Search-based and automated unit test generation has been explored for years; a prominent example is evolutionary test generation for Java [6]. At the API level, systematic ap- proaches generate request sequences to cover behaviors and discover bugs; for instance, REST API fuzzing and sequence generation has been demonstrated via automated techniques [7]. These methods focus on coverage and fault discovery but do not directly address multi-team silos and federated governance constraints.

2.3. Federated Learning and Robustness
Federated learning enables collaborative training without central data pooling [8], [9]. FL faces non-IID data challenges and security risks; robust aggregation methods such as Krum and trimmed-mean style approaches tolerate adversarial up- dates [11], [12]. Secure aggregation protects the privacy of individual updates [10], but privacy alone does not guarantee integrity or accountability.

2.4. Explainable AI for Governance and Auditability
Model explanations (LIME, SHAP, Integrated Gradients, Anchors) provide tools to interpret predictions [14]–[17]. In high-stakes decisions, there is a strong argument for inter- pretable models or carefully constrained explainability pro- cesses [18]. For CI/CD governance, explanations must map to actionable engineering artifacts (spec clauses, endpoints, commits, incidents), not only feature attributions.

2.5. Auditable Evidence and Provenance
Accountability requires tamper-resistant lineage for builds, tests, and decisions. Permissioned blockchain and auditable logging infrastructures provide non-repudiation and traceabil- ity [22], [23]. T-FedTest adopts an "audit plane" concept to record commitments, metadata, and rationales without expos- ing sensitive content across silos.

3. System Model and Design Goals
3.1. Microservice Testing Context
We consider an enterprise with many microservices grouped into *domains* owned by different teams. Each team maintains:
- Service specifications (OpenAPI/gRPC IDL, schemas),

- CI/CD pipeline definitions and checks,
- Local test suites (unit, contract, integration),
- Incident records and operational telemetry (error rates, traces, alerts).

Raw telemetry and incident narratives may be restricted; however, teams can compute local features and train local models.

3.2. GenAI Test Generation Task Definition
T-FedTest focuses on *test case generation* for microservices, producing:
- Request inputs: API calls and payloads, including edge cases,
- Oracles: expected status codes, schema constraints, and invariants,
- Sequences: multi-call workflows (e.g., create–update– delete),
- Rationales: why the test exists (spec mapping, incident coverage, regression risk).

Generated tests are executed as CI/CD stages, with optional promotion to nightly or pre-prod suites depending on cost.

3.3. Threat Model
We assume participants (teams/domains) may be:
- Honest: contribute stable improvements and accurate evaluations,
- Faulty: misconfigured instrumentation, noisy labels, un- stable training,
- Malicious: attempt poisoning to reduce detection of cer- tain failures, inflate quality metrics, or insert disruptive tests.

We also include accountability evasion behaviors such as missing provenance, inconsistent evaluation reporting, or with-holding negative outcomes.

3.4. Design Goals
T-FedTest targets:
- G1 Effectiveness: improve fault detection and regression prevention.
- G2 Integrity: resist poisoning and low-quality updates.
- G3 Accountability: provide audit trails for test genera- tion and model lineage.
- G4 Explainability: produce stable, actionable rationales.
- G5 CI/CD practicality: operate under pipeline time budgets and incremental change.

4. T-FedTest Architecture
T-FedTest is organized as four cooperating planes.
4.1. Plane A: Test Generation Plane
Each team runs a local generator that synthesizes test candidates conditioned on:
- API specs and schemas,
- code diffs and dependency changes,
- local incident patterns and telemetry-derived risk features.

The generator can be implemented using transformer-based sequence modeling for structured request creation [21] and constrained decoding to enforce schema validity.

4.2. Plane B: Federated Learning Plane
Local trainers update a shared *test generation model* (or shared components such as a spec-to-test encoder) using local data. Teams share model updates, not raw traces [9]. Secure aggregation may be applied to protect updates [10].

4.3. Plane C: Trust and Governance Plane
This plane computes trust scores and enforces integrity:
- Trust evaluator: computes trust from evidence signals.
- Trust-aware aggregator: performs trust gating and ro- bust aggregation [11], [12].
- Policy checks: ensure generated tests do not violate safety constraints (e.g., destructive calls without isola- tion).

4.4. Plane D: Explainability and Audit Plane
The explainability plane generates rationales, and the audit plane records commitments:
- Explanation artifacts (summaries) for generated tests,
- Hashes/commitments of test bundles and model versions,

- Trust rationale summaries and aggregation metadata.

A permissioned ledger or append-only log can provide in- tegrity guarantees [22], [23].

5. Trust Metric-Based Federated Learning Framework

This section defines the trust metric and aggregation strategy in an operational, non-formula-heavy manner.

5.1. Trust Metric Overview

Each participant i receives a trust score $T_i \in [0, 1]$ computed as a weighted sum of normalized components:

- Provenance and reproducibility (P_i): evidence that generated tests and training runs are reproducible, signed, and linked to verified pipeline context.
- Update consistency (U_i): anomaly checks for unstable or suspicious updates (e.g., abrupt parameter shifts).
- Evaluation reliability (E_i): stability and honesty of reported local evaluation results (repeated runs, variance bounds).
- Policy compliance (C_i): adherence to test safety policies (sandboxing, data handling, destructive-call controls).
- Test utility quality (Q_i): observed usefulness of gener- ated tests locally (fault discovery, regression catch rate, flakiness penalties).

5.2. Guardrails and Penalties

Trust is reduced sharply when severe events occur:

- Missing or invalid provenance attestations,
- Repeated generation of flaky or unsafe tests,
- Evaluation inconsistencies suggestive of metric inflation,
- Updates flagged as outliers across multiple rounds.

This keeps trust interpretable: accountability prerequisites must be satisfied before influence is granted.

5.3. Trust-Aware Aggregation

T-FedTest replaces data-volume-only weighting with:
Aggregation influence = local data/experience weight × trust weight.

After gating low-trust participants, T-FedTest applies robust aggregation to remaining updates, such as trimmed-mean style filtering or selection-based approaches [11], [12]. The combination is intentionally simple:

- Trust gating: exclude or strongly down-weight low-trust participants.
- Robust filtering: reduce influence of remaining anoma- lous updates.
- Weighted aggregation: average with trust-informed weights.

Integrity and Accountability Across Silos.

Accountability is achieved by binding trust to evidence:

- Each trust score includes a short *trust rationale* summary (e.g., “high flakiness” or “missing signing”),
- Audit plane records per-round commitments for trust and model lineage,
- Transfer of generated tests to shared catalogs requires provenance verification.

6. Federated Explainability and the Explainability–Performance Trade-off

6.1. Explainability Requirements for Generated Tests

Generated tests must be explainable at the level of engineer- ing action:

- Which endpoint, schema constraint, or workflow is being exercised,
- Which regression risk or incident pattern motivated the test,
- Why specific inputs/edge cases were selected,
- What oracle/invariant is being asserted.

6.2. Explanation Methods and Artifacts

T-FedTest produces two explanation types:

(X1) Test rationale explanation: A compact rationale for each generated test, including:

- Spec mapping (endpoint and schema fields),
- Risk signal mapping (e.g., error spikes, contract drift),
- Change mapping (commit/diff category, dependency change).

(X2) Model decision explanation: For selected high-impact tests or policies, provide local feature attributions or rule-like justifications using:

- LIME/SHAP-style feature attribution [14], [15],
- Integrated Gradients for neural decisions [16],
- Anchors for high-precision rule-like explanations [17].

To preserve silo privacy, explanations are generated locally; only summaries and stability scores are shared, with hashed commitments logged.

6.3. Explainability Quality Measures (Operational)
Without complex formulas, T-FedTest evaluates explana- tions via:

- Fidelity: does the explanation align with model behavior on local perturbations?
- Stability: do top-k factors remain consistent under minor input noise?
- Actionability: can the explanation be mapped to a con- crete remediation or requirement?
- Compactness: can the rationale be expressed succinctly?

6.4. Trade-off Controller via Explanation Budgets
T-FedTest introduces an *explanation budget* per pipeline window. The controller decides:

7. Methodology
7.1. Local Data and Features
Each team constructs local training examples from:

- API spec snapshots and diffs,
- Code changes (categorized into API, validation, auth, persistence),
- Test outcomes (pass/fail, flakiness, runtime),
- Production signals (errors, latency shifts, incidents),
- Contract violations (schema mismatches, backward- incompatibility indicators).

7.2. Generation Targets
The generator produces structured test representations (re- quest, oracle, sequence, rationale). Constrained decoding en- sures:

- Schema validity for payloads,
- Safe execution constraints (no destructive calls outside sandbox),
- Deterministic oracle formatting.

7.3. Local Evaluation Protocol
Each team evaluates generated tests on:

- Fault detection: discovered bugs or prevented regressions (proxy via injected faults or known regressions),
- Signal quality: precision of failures (reproducibility, low flakiness),
- Cost: runtime and resource usage in pipeline.

Evaluation reliability score E_i is derived from repeated runs and variance checks to discourage inflated reporting.

7.4. Federated Training Procedure
Each round:

1. Aggregator publishes the current global generator com- ponents.
2. Teams perform local updates and evaluate locally.
3. Teams publish signed/committed trust reports and update summaries.
4. Aggregator gates low-trust participants and aggregates updates robustly.
5. Updated model is published with lineage recorded in audit plane.
    - Which generated tests require full explanations,
    - Which tests get lightweight rationales,
    - Whether stability checks are enforced,
    - Whether to use more interpretable generation strategies (e.g., constrained templates) versus higher-capacity free-form generation.

The controller optimizes a simple utility notion:

Utility increases with fault-detection effectiveness and explanation quality, and decreases with expla- nation cost and pipeline time impact.

This supports governance: explain the most critical tests and model decisions while keeping CI/CD throughput acceptable.

8. Experiments

Since real enterprise CI/CD data is often proprietary, we evaluate using a controlled simulation designed to resemble multi-team microservice ecosystems.

8.1. Simulation Setup

Teams/services. 30 teams, 180 microservices total. Services differ in API size (5–40 endpoints), schema complexity, and change frequency.

Fault model: We inject realistic fault types:

- Schema drift (missing/renamed fields),
- Auth/role regressions (permission checks),
- Validation boundary errors (off-by-one, null handling),
- Idempotency and retry side effects,
- Dependency mismatch errors.

Non-IID distributions. Each team experiences different fault prevalence and endpoint usage patterns, representing heterogeneous production conditions.

Adversaries and faulty participants:

- 3 malicious participants attempt to poison updates to reduce detection of auth regressions.
- 5 faulty participants have high flakiness and unstable evaluations.

8.2. Compared Methods

- B1 FedAvg: standard federated averaging [9].
- B2 Robust-only: robust aggregation without trust scoring [12].
- B3 Trust-only: trust-weighted aggregation without robust filtering.
- T-FedTest: trust gating + robust aggregation + explain- ability budgeting.

8.3. Metrics

Effectiveness metrics:

- Fault Detection Rate (FDR): fraction of injected faults caught by generated tests.
- Regression Catch Rate (RCR): fraction of seeded re- gressions detected before release.
- False Alarm Rate (FAR): failing tests not linked to real faults (includes flakiness).

Explainability metrics:

- **Stability:** top-k factor agreement under small perturba- tions.
- **Actionability:** fraction of explanations mapped to spec/contract categories.
- **Cost units:** relative compute/latency proxy for explana- tion workload.

8.4. Explainability Budget Regimes

- E1 Low: full explanations for top 5% high-risk tests only.
- E2 Medium: full explanations for top 20% and stability checks.
- E3 High: full explanations for all generated tests.

9. Results

9.1. Effectiveness and Robustness

Table I summarizes results under adversarial and faulty participants (averaged over runs).

Table 1: Ieffectiveness under Adversarial/Faulty Participants

Method	FDR	RCR	FAR
B1 FedAvg	0.69	0.62	0.17
B2 Robust-only	0.74	0.66	0.14
B3 Trust-only	0.76	0.68	0.13
T-FedTest	0.82	0.74	0.09

Trust-aware learning improves detection and reduces false alarms by down-weighting contributors that produce flaky tests or unreliable evaluations, and by limiting the influence of poisoned updates. Robust-only aggregation helps, but the combined trust+robust approach performs best because it uses *evidence-driven gating* in addition to statistical filtering.

9.2. Ablation: Integrity Controls
We observed that:

- Trust-only aggregation improves over FedAvg when the main risk is low-quality participants (flakiness and unsta- ble evaluation).
- Robust-only aggregation helps against extreme outliers but cannot distinguish systematically unreliable teams from honest teams with hard distributions.
- Combining trust gating and robust filtering provides re- silience across both failure types: governance failures and statistical anomalies.

9.3. Explainability–Performance Trade-off
Table II reports T-FedTest outcomes under different expla- nation budgets.

Table 2: Explainability Budget Trade-Off (T-Fedtest)

Budget	FDR Expl	Stability	Cost Units
E1 Low	0.83	0.61	1.0
E2 Medium	0.82	0.78	2.4
E3 High	0.80	0.81	5.1

A moderate budget (E2) yields substantial stability gains with minimal reduction in fault detection. High budgets marginally improve stability but increase operational costs and slightly reduce performance due to additional stability filtering and pipeline overhead. This supports a practical policy: explain the most risk-relevant tests deeply, and provide lightweight rationales for the rest.

9.4. Actionability of Explanations
Under E2, the majority of explanations mapped to action- able categories:

- Schema drift coverage (fields and constraints),
- Authentication/authorization coverage (role boundaries),
- Idempotency and retry behavior coverage,
- Dependency mismatch regression coverage.

These categories align with common microservice failure modes and provide a clear pathway for remediation.

9.5. Accountability Outcomes
Audit plane commitments enabled reconstruction of:

- which teams influenced the model most (trust-weighted contribution),
- why specific teams were gated (trust rationales),
- lineage of generated test bundles and their associated rationales.

This supports compliance and incident response workflows without requiring cross-silo raw data access.

10. Discussion
10.1. Integration into CI/CD Pipelines
T-FedTest can be integrated as:

- pre-merge stage: generate unit/contract tests from spec and diff; run fast checks.
- build stage: generate API tests and safe workflow se- quences; run against ephemeral environments.
- pre-prod stage: run high-cost generated scenarios for critical services and high-risk changes.

Continuous delivery practices recommend automated gates and feedback loops [1]; T-FedTest formalizes a learning loop that improves gates over time while preserving governance constraints.

10.2. Why Trust is Essential for Federated Test Generation
Test generation systems can cause harm if they produce flaky or unsafe tests that block deployments, or if they hide regressions by biasing generation. Trust metrics make influence conditional on:

- Reproducibility and provenance,
- Stable evaluation behavior,

- Safe-test compliance.

This transforms federated learning from “everyone contributes equally” to “accountable contributors shape the shared model.”

10.3. Interpretable-First vs Hybrid GenAI

In high-assurance environments, interpretability concerns motivate simpler, more transparent approaches [18]. T-FedTest supports:

- Interpretable-first generation: constrained templates and spec-driven rules with direct rationale links,
- Hybrid generation: higher-capacity generation with ex- planation budgets and stability checks.

This allows organizations to choose a compliance posture without losing the benefits of learned generation.

10.4. Limitations

- Simulation constraints: Real microservice ecosystems in- clude complex dependencies and human processes not fully captured in simulation.
- Schema standardization: T-FedTest benefits from consis- tent spec formats and telemetry taxonomies; organizations may need platform standardization.
- Trust gaming risk: Participants could optimize trust metrics rather than outcomes. Guardrails, periodic audits, and rotated evaluation protocols mitigate but do not eliminate this risk.

11. Conclusion

This paper presented T-FedTest, a GenAI-powered test case generation framework for microservices in CI/CD pipelines us- ing trusted federated explainability. T-FedTest addresses cross- silo constraints by training collaboratively without centralizing raw data, while ensuring integrity and accountability through a trust metric-based federated learning protocol with robust aggregation. It further provides a practical mechanism to quan- tify and optimize the explainability–performance trade-off via explanation budgets and stability checks, enabling auditable and effective test generation at enterprise scale. Experimental results in a controlled simulation demonstrate improved fault detection, reduced false alarms, and robust behavior under adversarial and faulty participants, while maintaining stable, actionable explanations under moderate budgets. Future work includes production deployments, richer provenance attesta- tions, and privacy-preserving sharing of explanation sum- maries across regulatory domains.

Acknowledgment

The author thanks practitioners and researchers whose work on microservices, CI/CD, federated learning, and explainable AI informed this framework perspective.

References

[1] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison- Wesley, 2010.
[2] S. Newman, *Building Microservices*. O’Reilly Media, 2015.
[3] N. Dragoni *et al.*, “Microservices: Yesterday, today, and tomorrow,” in *Present and Ulterior Software Engineering*, Springer, 2017.
[4] P. Di Francesco, P. Lago, and I. Malavolta, “Research on architecting microservices: Trends, focus, and potential for industrial adoption,” in *Proc. IEEE ICSA*, 2017.
[5] J. Soldani, D. A. Tamburri, and W.-J. van den Heuvel, “The pains and gains of microservices: A systematic grey literature review,” *J. Systems and Software*, vol. 146, pp. 215–232, 2018.
[6] G. Fraser and A. Arcuri, “EvoSuite: Automatic test suite generation for object-oriented software,” in *Proc. ACM ESEC/FSE*, 2011.
[7] A. Atlidakis, P. Godefroid, and M. Polikarpova, “RESTler: Stateful REST API fuzzing,” in *Proc. IEEE/ACM ICSE*, 2019.
[8] J. Konecˇny´, B. McMahan, and D. Ramage, “Federated optimiza- tion: Distributed optimization beyond the datacenter,” *arXiv preprint arXiv:1511.03575*, 2015.
[9] H. B. McMahan *et al.*, “Communication-efficient learning of deep networks from decentralized data,” in *Proc. AISTATS*, 2017.
[10] K. Bonawitz *et al.*, “Practical secure aggregation for privacy-preserving machine learning,” in *Proc. ACM CCS*, 2017.
[11] P. Blanchard, E. Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent,” in *Proc. NeurIPS*, 2017.
[12] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, “Byzantine-robust distributed learning: Towards optimal statistical rates,” in *Proc. ICML*, 2018.
[13] P. Kairouz *et al.*, “Advances and open problems in federated learning,” *arXiv preprint arXiv:1912.04977*, 2019.
[14] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should I trust you?: Explaining the predictions of any classifier,” in *Proc. ACM KDD*, 2016.

[15] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Proc. NeurIPS*, 2017.
[16] M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic attribution for deep networks,” in *Proc. ICML*, 2017.
[17] M. T. Ribeiro, S. Singh, and C. Guestrin, “Anchors: High-precision model-agnostic explanations,” in *Proc. AAAI*, 2018.
[18] C. Rudin, “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,” *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019.
[19] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Proc. NeurIPS*, 2014.
[20] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” in *Proc. ICLR*, 2015.
[21] A. Vaswani *et al.*, “Attention is all you need,” in *Proc. NeurIPS*, 2017.
[22] E. Androulaki *et al.*, “Hyperledger Fabric: A distributed operating system for permissioned blockchains,” in *Proc. EuroSys*, 2018.
[23] B. Putz, F. Pernul, and G. Kablitz, “A secure and auditable logging infrastructure based on a permissioned blockchain,” *Computers & Secu- rity*, vol. 87, 2019.