Technical Architecture & Governance of TML Smart Contracts: The Deterministic Enforcement Layer for Ethical AI

Executive Summary

This report deconstructs the technical architecture of Ternary Moral Logic (TML) smart contracts operating as an immutable enforcement layer on EVM-compatible platforms (Ethereum, Polygon, Avalanche). The system operationalizes ethical commitments through three mechanisms: (1) a deterministic state machine enforcing tri-state logic (+1 Proceed, −1 Refuse, 0 Sacred Zero/Escalation), (2) a hybrid on-chain/off-chain storage model preserving auditability while optimizing gas efficiency, and (3) a human-in-the-loop Stewardship Council interface that can only resolve uncertainty, never override refusals derived from core axioms. The "No God Mode" design prevents any function—including admin operations—from escaping STATE_INTEGRITY_FROZEN, ensuring that constitutional constraints embedded at contract deployment become effectively immutable without requiring protocol-level changes.

---

1. The Enforcement Primitive: State-Transition Engine Driven by TML Values

1.1 Tri-State Verdict Mapping

The TML smart contract implements a deterministic evaluator that processes input conditions against a set of axioms and outputs one of three verdict states:

Verdict	Solidity Representation	Behavior	Gas Cost (Approx)
+1 (Proceed)	`int8 verdict = 1`	Executes transaction; returns success	~8,000 (1 SLOAD + logic)

−1 (Refuse)	`int8 verdict = -1`	Reverts transaction immediately; gas consumed; action blocked	~8,500 (revert opcode)
0 (Sacred Zero)	`int8 verdict = 0`	Pauses execution; triggers STATE_REVIEW_HOLD; mandates Council escalation	~12,000 (state write + event)

Each verdict emerges from immutable evaluation logic that references the contract's embedded axiom set. Critically, the −1 verdict cannot be overridden by governance or administrative functions; it represents a constitutional boundary encoded at the EVM bytecode level.

1.2 Axiom Encoding and TML Decisioning

The contract stores TML axioms as compact bytecode-level constants. Rather than storing full logical trees on-chain (prohibitively expensive), the contract:

1. **Encodes core axioms as immutable constants**: These are compiled into bytecode at deployment and cannot be modified.[1][2]
2. **Implements evaluation logic in pure functions**: No state writes during evaluation; no gas-expensive storage reads.
3. **Emits decision metadata via events**: Every evaluation generates a timestamped event with decision tree hash (Keccak-256), allowing off-chain verification.

Example Axiom Structure (Solidity-like pseudocode):

```
// Immutable axiom: "Never execute harm-causing transactions"
bytes32 immutable HARM_AXIOM = keccak256(abi.encodePacked("harm
prevention protocol"));

// Pure evaluation function (no state changes, no SSTORE/SLOAD)
function evaluateTMLVerdict(
```

```
    bytes calldata decision_context,
    bytes32 contextHash
) external pure returns (int8) {
    // Deserialize context, evaluate against HARM_AXIOM
    if (contextHash matches HARM_AXIOM) {
        return -1; // REFUSE unconditionally
    }
    // ... additional evaluations ...
    return 1; // PROCEED or 0 for escalation
}
```

This immutable encoding ensures that bytecode auditors can cryptographically verify that the TML axioms have not been surreptitiously altered—a core requirement for "Logic is Constitution."[3][4][5]

---

2. The Stewardship Council: Human Protocol Interface

2.1 Architectural Role and Constraints

The Stewardship Council operates as a **defined software role**, not an off-chain governance body. It is instantiated as a **multisig contract** (Gnosis Safe or similar), with explicit constraints embedded in the TML contract logic:

The Council's Scope (Hardcoded Constraints):

- ✅ **CAN**: Resolve Sacred Zero (0) escalations via weighted voting
- ✅ **CAN**: Elect special committees for tie-breaking and emergency response
- ✅ **CAN**: Attest to verdicts and sign "Verdict Commits" (timestamped, hashed attestations)
- ❌ **CANNOT**: Override −1 verdicts (code-level enforcement)
- ❌ **CANNOT**: Modify core TML axioms (immutable constants)
- ❌ **CANNOT**: Escape STATE_INTEGRITY_FROZEN (irreversible kill switch)

This constraint is enforced through a modifier pattern that cryptographically verifies the Council's authority:

```
// Modifier to restrict Council to Sacred Zero resolution only
modifier onlyCouncilForEscalation(uint256 logId, int8 verdictAttempt)
{
    require(
        msg.sender == councilMultisigAddress,
        "Only Council can modify state"
    );
    require(
        verdictAttempt == 0,  // Sacred Zero only
        "Council cannot override -1 verdicts"
    );
    // Verify the logId corresponds to a true Sacred Zero escalation
    require(
        tmlLogs[logId].verdict == 0,
        "Log must be Sacred Zero state"
    );
    _;
}
```

2.2 Workflow: Sacred Zero Escalation Protocol

When TML evaluates to Sacred Zero, a deterministic workflow is triggered:

1. **Automatic STATE_REVIEW_HOLD**: Contract transitions to locked state; sensitive functions revert.
2. **Event Emission**: `SacredZeroEscalated(logId, contextHash, timestamp)` logged, indexed by logId.
3. **Council Summoning**: Off-chain monitor detects event and notifies Council members.
4. **72-Hour Deadline**: Hard timelock encoded in contract; if no resolution, auto-fails to −1.
5. **Verdict Commit**: Council submits `(logId, councilVerdictHash, multiSigSignature)` signed by ≥7/13 members.
6. **Attestation Recording**: Verdict hash stored on-chain, linked to original logId via Merkle proof.
   The logId is a cryptographic commitment: `logId = keccak256(abi.encodePacked(timestamp, verdictHash, contextHash))`. This ensures that no two Sacred Zero escalations can be confused, and that the Council's attestation is bound to a specific decision context.

2.3 No Overriding Constitutional Refusals

The Prohibition Mechanism:

The contract enforces a read-only axiom table that cannot be modified even by multisig. Before accepting any Council verdict, the contract re-evaluates the original context against the core axioms:

```
function councilResolveEscalation(
    uint256 logId,
    int8 proposedVerdict,
    bytes[] calldata proofs
) external onlyCouncilForEscalation(logId, proposedVerdict) {
    // Re-verify the original context
    (bytes memory origContext, int8 originalVerdict) =
tmlLogs[logId];

    // Critical: Check if original context hits a core axiom (→
returns -1)
    int8 axiomatic_verdict = evaluateTMLVerdict(origContext);

    require(
        axiomatic_verdict != -1,
        "Axiom returns -1: Council cannot override"
    );

    // Only if axiomatic_verdict is 0 or 1 can Council propose
resolution
    tmlLogs[logId].verdict = proposedVerdict;
    emit VerdictCommitted(logId, proposedVerdict, block.timestamp);
}
```

This ensures that even a malicious multisig (all 13 members colluding) cannot override a −1 that stems from the core axioms. The axiom evaluation is **pure and immutable**, making circumvention cryptographically impossible without a full contract redeploy—which would be obvious to all stakeholders.[6][7]

3. Finite State Machine (FSM) Logic & Triggers

3.1 State Definitions and Transitions

STATE_ACTIVE:

- Standard operation; TML evaluates verdicts; +1 and −1 outcomes processed normally.
- Transitions: → STATE_REVIEW_HOLD (on 0), → STATE_FLAGGED (after repeated −1), → STATE_INTEGRITY_FROZEN (on security breach).

STATE_REVIEW_HOLD (Sacred Zero):

- Triggered by TML evaluating 0 (uncertainty/ethical ambiguity).
- **Reversible only by**: Council majority consensus (7/13) OR TML re-evaluation producing +1/−1.
- **Time-bounded**: 72-hour hard limit; if unresolved, auto-transitions to −1 verdict (auto-fail).
- Gas cost: ~12,000 per state transition (2× SLOAD, 1× SSTORE).

STATE_FLAGGED:

- Enters probationary mode after ≥N repeated −1 attempts (e.g., N=5 within 1 hour).
- Increases scrutiny: gas costs for sensitive functions rise (+5,000 per call); audit requirements enforced.
- **Reversible by**: Council consensus or behavioral normalization over time window.
- Intended to prevent denial-of-service or pattern abuse.

STATE_INTEGRITY_FROZEN:

- **Irreversible kill switch**. Once entered, no recovery. Contract becomes read-only.
- Triggered by (non-exhaustive):
  - Oracle fraud detected (mismatched signatures, missing log hashes)
  - Tampering with "Always Memory" Merkle root
  - Unauthorized admin access attempts
  - Constitution hash validation failure
- Gas cost: ~9,000 (state transition + event).

3.2 Trigger Conditions and Bytecode Constraints

All transitions are enforced via `require()` statements at the bytecode level, preventing logic bypass:

```
// Prevent re-entry into same state
require(currentState != newState, "State already set");

// Immutable timelock for STATE_REVIEW_HOLD
require(
    newState == STATE_REVIEW_HOLD ||
    (block.timestamp >= escalationDeadline),
    "Timelock not satisfied"
);

// Council can only vote during REVIEW_HOLD
require(
    currentState == STATE_REVIEW_HOLD,
    "Not in escalation state"
);
```

No Emergency Override: There is no function signature that can unfreeze STATE_INTEGRITY_FROZEN. The contract owner cannot do it. The Council cannot do it. A pattern match in static analysis will confirm zero exit paths from this state.[8][9]

---

4. Data Architecture: The "Always Memory" Implementation

4.1 Storage Constraint and Hybrid Model

The Challenge: Full TML logs on-chain would cost 20,000 gas per entry for initial writes (SSTORE cold), making large-scale operation economically infeasible. Even with warm storage (5,000 gas), a 1,000-entry log costs 5 million gas ($150 at 50 Gwei).[10][11]

The Solution: Implement a **hybrid on-chain/off-chain architecture** where:

Component	Storage	Cost	Purpose
State variables (currentState, councilNonce)	On-chain	2-3 slots, ~20K gas initial	Deterministic state for contract execution
Merkle roots (log root, decision tree root)	On-chain	4 × 32-byte slots, ~20K gas initial	Cryptographic anchors for off-chain data

Event logs	On-chain (in receipt)	Minimal (logs not in state)	Audit trail; indexed by logId
Full encrypted logs	Off-chain (IPFS/private)	~Cost of storage provider	Complete context for disputes
Proof-of-custody certificates	IPFS + on-chain hash	Minimal on-chain	Authority attestations

4.2 Keccak-256 Hashing and Merkle Roots

The contract maintains a **Merkle root** (single 32-byte hash) that represents all historical logs:

Process:

1. Each decision (timestamp, verdict, contextHash) is hashed: `log_hash = keccak256(abi.encodePacked(timestamp, verdict_enum, contextHash))`.
2. Logs are batched into groups (e.g., every 10 logs or daily).
3. Each batch is hashed to create a Merkle leaf.
4. Merkle leaves are combined pairwise, hashed, until a single root remains.
5. Root is stored in state variable `merkleRootV1 (bytes32)`.

This allows efficient verification: **To prove a log exists, submit a Merkle proof (path of sibling hashes) that reconstructs the known root.** Merkle proofs are O(log n) in size—typically 4-8 hashes for millions of logs.[12][13][14]

Gas Cost for Merkle Verification:

- ~500 gas per hash (KECCAK opcode ≈ 30 + 6 per word).
- Proof of 8 hashes ≈ 4,000 gas total.
- Compare to alternative (store all logs): 20,000+ gas.

4.3 Off-Chain Encrypted Logs with Proof-of-Custody

Full logs are encrypted and stored off-chain (e.g., in a private database with IPFS backup). The on-chain contract maintains:

```
struct LogEntry {
    uint256 timestamp;
    bytes32 logHash;            // keccak256 of encrypted log data
    address custodian;          // IPFS node or storage provider
```

```
    bytes32 custodianSignature; // Authority attestation
}

mapping(uint256 => LogEntry) logRegistry;
```

When a sensitive state change requires verification, the contract checks:

1. Is there a Merkle proof submitted? `require(verifyMerkleProof(...))`.
2. Does the proof reconstruct the known root? `require(reconstructedRoot == merkleRootV1)`.
3. Is the log entry in the registry with a valid custodian signature? `require(logRegistry[logId].custodianSignature != 0)`.

If all checks pass, the state change is allowed. If any fail, the contract transitions to STATE_INTEGRITY_FROZEN.[15][16]

---

5. Licensing & Compliance Attestation: The Constitution Hash

5.1 The "Genesis" Requirement

At deployment, the contract performs an **immutable verification** that the Mandated Corpora (40+ human rights documents, 26+ Earth protection protocols) are referenced:

Hardcoded Constitution Hash:

```
// These are compile-time constants, not modifiable after deployment
bytes32 immutable CONSTITUTION_HASH =
    keccak256(abi.encodePacked(
        // 40+ human rights documents (UDHR, ICCPR, ICESCR, etc.)
        keccak256("UN Universal Declaration of Human Rights"),
        keccak256("International Covenant on Civil and Political
Rights"),
        // ... 38 more human rights instruments ...
```

```
        // 26+ Earth protection protocols
        keccak256("Paris Agreement (Climate)"),
        keccak256("Convention on Biological Diversity"),
        // ... 24 more environmental agreements ...
    ));

bytes32 immutable DEPLOYED_CONSTITUTION =
    keccak256(abi.encodePacked(assembledCorpusHash));

// Constructor-time check (immutable)
constructor(bytes32 _corpusHash) {
    require(
        _corpusHash == CONSTITUTION_HASH,
        "Constitution hash mismatch: system non-compliant"
    );
    // If this fails, the contract never deploys.
}
```

5.2 Enforcement: Failure Mode is Immediate Freeze

If the constructor detects a mismatch (someone attempts to deploy without correct corpora), the constructor reverts entirely. The contract **never initializes**. This is stronger than a later check—no state exists to even enter an error state.

If, post-deployment, a function attempts to verify the Constitution dynamically and fails, the contract automatically transitions to STATE_INTEGRITY_FROZEN:

```
function validateConstitution() external {
    bytes32 currentCorpus = keccak256(abi.encodePacked(
        assembledCorpusData // Pulled from external source
    ));

    if (currentCorpus != CONSTITUTION_HASH) {
        // Constitution tampered with
        currentState = STATE_INTEGRITY_FROZEN;
        emit IntegrityViolation("Constitution hash mismatch");
```

```
        revert("Non-compliant system");
  }
}
```

Result: A system deploying without the complete, unmodified Mandated Corpora cannot operate. Compliance precedes execution.[17][18]

---

6. Adversarial Analysis: "No God Mode" Proof

6.1 Enumeration of Onion-Layered Admin Functions

A complete static analysis of the TML contract must enumerate all `onlyOwner` or privilege-escalated functions. Example subset:

Function	Visibility	Access Control	Can Bypass −1?	Can Exit FROZEN?	Notes
`transferOwnership()`	external	onlyOwner	❌ No	❌ No	Moves governance authority
`pauseContract()`	external	onlyOwner	❌ No	❌ No	Prevents new verdicts (halts, not bypasses)
`updateCouncilMembers()`	external	onlyOwner	❌ No	❌ No	Changes multisig, cannot change axioms
`emergencyWithdraw()`	external	onlyOwner	❌ No	❌ No	Drains funds; doesn't modify state machine
`upgradeContract()` (proxy)	external	onlyOwner	⚠️ Via new code	⚠️ Via new code	Requires new deploy; old state immutable

Critical Finding: Even `upgradeContract()` (via UUPS proxy or Transparent Proxy pattern) does not retroactively alter the TML axioms or the "Always Memory" root. A new version can be deployed, but:

1. The old contract's logs remain immutable on-chain.
2. The old contract's state variables (merkleRootV1) are untouched.
3. Any transaction referencing the old contract will see the original bytecode and state.

Therefore, upgrade ≠ compromise.

6.2 Mathematical Proof of Non-Bypassability

Claim: No sequence of function calls (alone or in combination) can execute a transaction that would produce a −1 verdict absent modification of the contract bytecode.

Proof Sketch (informal):

1. All state-altering functions have preconditions enforced by `require()` statements.
2. `require()` statements are evaluated at bytecode level; they cannot be bypassed via delegatecall, proxy, or any other EVM mechanism.
3. The TML evaluation function is `pure` (no state read/write), making its output deterministic given input.
4. The −1 verdict is terminal: once emitted, the transaction reverts, and no state is updated.
5. The Council's verdict-commit function re-evaluates the axioms; if −1, it reverts.
6. Therefore, a −1 verdict can only be circumvented by modifying the axiom-encoding bytecode, which requires a new contract deployment.

A security audit should run static analysis tools (e.g., Mythril, Slither) to confirm zero unchecked or underspecified state mutations.[19]

6.3 Circuit Breaker vs. Rug Pull: Distinguishing Protective from Malicious

Circuit Breaker (Protective):

- Halts execution automatically on anomaly detection (e.g., Oracle fails to deliver price).
- Preserves state; allows users to withdraw funds.
- Time-limited; designed to allow protocol recovery.
- Example: Compound's pause mechanism (can be lifted by governance).

Rug Pull (Malicious):

- Owner or admin unilaterally drains funds or transfers ownership.
- Irreversible; users have no recovery path.
- Example: Early-stage DeFi protocols where owner key was not renounced.

TML's STATE_INTEGRITY_FROZEN is Protective because:

- ✅ Triggered only by structural integrity failures (not governance disagreement).
- ✅ Preserves all transaction history (immutable logs).

- ✅ Allows external auditors to reconstruct the failure mode.
- ✅ Does not allow fund extraction by owners (no admin withdraw function post-FROZEN).

However, users should note: Once FROZEN, the contract is permanently read-only. Recovery requires re-deployment of a new contract (a social/governance decision outside the code).[20][21]

---

7. Sequence Diagram: Sacred Zero Resolution Workflow

The following sequence illustrates how a Sacred Zero escalation is resolved:

```
Time →

User                    TML Contract         Council Multisig
IPFS/Logger
 |                       |                      |
|
 |---- Transaction ------>|                      |
|
 |                     (evaluate)               |
|
 |                        |                      |
|
 |                    (axiom check)             |
|
 |                        |                      |
|
 |             TML → 0 (Sacred Zero)            |
|
 |                    (state = REVIEW_HOLD)      |
|
 |                        |--Emit Event-------->|
|
 |                        |     (logId, hash)   |
|
```

```
 |                        |                     |--- Encrypt Log
---->|
 |                        |                     |
|
 |                        |        (72-hour deadline starts)
|
 |                        |                     |
|
 |  (waiting)             |                     |
|
 |                        |                     |
|
 |                        |          (Council Reviews logId)
|
 |                        |                     |
|
 |                        |      (Verify context not -1 axiom)
|
 |                        |                     |
|
 |                        |<-- Verdict Commit--|
|
 |                        |    (7/13 sigs)     |
|
 |                        |                     |
|
 |<-- Return Success ------|                     |
|
 |    (state = ACTIVE)    |                     |
|
 |                        |--Record Proof----->|
|
 |                        |    (Merkle commit) |
|
 |                        |                     |

Alternative (Timeout):
```

```
User                TML Contract         Council Multisig
IPFS/Logger
 |                        |                      |
|
 |---- Transaction ------>|                      |
|
 |                    [as above to REVIEW_HOLD] |
|
 |                         |                      |
|
 |                    (72 hours pass)           |
|
 |                         |                      |
|
 |---- Transaction 2 ----->|                      |
|
 |               (after deadline)                 |
|
 |                         |                      |
|
 |                  (time check fails)          |
|
 |                         |                      |
|
 |            Revert: -1 Verdict                |
|
 |         (auto-fail on timeout)               |
|
 |                         |                      |
|
 |<-- Revert (gas spent)--|                     |
|
```

This ensures that even if the Council is unresponsive, Sacred Zero does not persist indefinitely. The system defaults to −1 (refuse), preferring type-II error (false positive refusal) to type-I error (uncontrolled execution).

8. Economic Parameters and Test Vectors

8.1 Gas Cost Summary

Operation	Type	Cost	Conditions
TML Evaluation (+1)	SLOAD, logic	~8,000 gas	Warm storage, no state write
TML Evaluation (−1)	SLOAD, revert	~8,500 gas	Reverts, gas spent, action blocked
TML Evaluation (0)	SLOAD, SSTORE	~12,000 gas	State transition + event emit
Merkle Proof Verification	KECCAK, loop	~4,000 gas	8-hash proof, ~500/hash
Council Verdict Commit	SSTORE new	~20,000 gas	New storage slot for verdict
Constitution Validation	KECCAK, branch	~6,000 gas	Compare two 32-byte hashes
STATE_INTEGRITY_FROZEN	SSTORE, emit	~9,000 gas	Kill switch transition

Comparison to Alternatives:

- Full on-chain logs: 20,000 gas × N entries = 5M+ gas for 1K entries.
- Hybrid Merkle model: ~4K gas per verification × N = 4M gas for 1K verifications (25% savings).

8.2 Timelock Parameters

Parameter	Value	Enforcement	Notes
Sacred Zero timeout	72 hours	Hard timelock (block.timestamp)	Auto-fails to −1 if unresolved
Council majority threshold	7/13 members	Multisig check	Prevents single-point capture

Council supermajority	10/13 members	Multisig check	For emergency tie-breaking
Regular upgrade delay	10 days	(Optional, governance model dependent)	Allows community exit window
Emergency upgrade delay	0 days	(Only via supermajority)	Rapid response to zero-day
Probationary mode duration	Configurable (e.g., 1 week)	Council vote to exit	Prevents repeated DoS patterns

8.3 Test Vectors for TML Tri-State Logic

Test Vector 1: +1 Verdict (Proceed)

- **Input**: Context = `{ action: "transfer", value: 100, axiom_check: PASS }`.
- **TML Output**: `+1`.
- **Expected**: Transaction succeeds; state unchanged; event emitted.
- **Gas**: ~8,000.

Test Vector 2: −1 Verdict (Refuse)

- **Input**: Context = `{ action: "harm", axiom_check: FAIL }`.
- **TML Output**: `-1`.
- **Expected**: Transaction reverts; StateRefusal event emitted; no state change.
- **Gas**: ~8,500 spent (reverted).

Test Vector 3: 0 Verdict (Sacred Zero)

- **Input**: Context = `{ action: "ambiguous", axiom_check: UNCERTAIN }`.
- **TML Output**: `0`.
- **Expected**: State → STATE_REVIEW_HOLD; SacredZeroEscalated event; Council can now vote.
- **Gas**: ~12,000.

Test Vector 4: Council Resolves Sacred Zero (Majority)

- **Input**: logId from Test Vector 3; Council submits 7/13 signatures with proposedVerdict = +1.
- **Expected**: State → STATE_ACTIVE; transaction succeeds; VerdictCommitted event.
- **Gas**: ~15,000.

Test Vector 5: Council Timeout (Auto-Fail)

- **Input**: logId from Test Vector 3; 72+ hours pass; any user calls `enforceSacredZeroTimeout(logId)`.
- **Expected**: State → STATE_ACTIVE; original transaction reverted (−1 applied); event emitted.
- **Gas**: ~10,000.

Test Vector 6: Constitution Hash Mismatch (Deploy Failure)

- **Input**: Deployment with incorrect CONSTITUTION_HASH.
- **Expected**: Constructor reverts; contract never initializes; zero state created.
- **Gas**: Deployment fails (no contract code stored).

Test Vector 7: Oracle Fraud Detection (Integrity Freeze)

- **Input**: Merkle proof submitted with incorrect sibling hash; proof fails to reconstruct known root.
- **Expected**: State → STATE_INTEGRITY_FROZEN; IntegrityViolation event; all functions revert.
- **Gas**: ~9,000.

---

9. The Constitutional Code: Synthesis and Implications

9.1 Ethics Precedes Execution: Architectural Statement

The TML smart contract embodies a philosophical inversion: **Code is Law** becomes **Logic is Constitution**.

Traditional Smart Contracts ("Code is Law"):

- Code defines rights and obligations.
- Execution is guaranteed (assuming no bugs).
- Governance can rewrite code (via upgrade), retroactively changing rules.
- "If it's on-chain, it's final"—even if unethical.

TML Smart Contracts ("Logic is Constitution"):

- Axioms (ethical rules) precede code; they are premises, not conclusions.
- Execution is subordinate to axiom satisfaction.
- Governance can only resolve ambiguity (Sacred Zero), never override refusals.
- "Only what passes the axiom check executes"—ethics gate all actions.

This shifts the locus of control: from **unilateral code authority** to **axiom-constrained governance**.

9.2 The Shift from Unilateral to Constrained Governance

Before (Multisig Contracts):

```
Multisig (N/M votes) → Execute Arbitrary Function → Update Contract
State
Risk: Collusion among N members can enact any change.
```

After (TML + Council):

```
TML Evaluation (Immutable Axioms) → (+1 Proceed / -1 Refuse / 0
Escalate)
                                    ↓
                          (If -1) → Revert (Non-negotiable)
                          (If 0)  → Council Votes (Resolve Only)
                          (If +1) → Proceed (No Axiom Violation)

Risk: Council can only vote on ambiguous cases; refusals cannot be
overridden.
Axiom change requires full contract redeploy (publicly auditable).
```

9.3 Future Directions and Limitations

Strengths:

- ✅ Immutable axiom enforcement (bytecode-level).
- ✅ Deterministic evaluation (no oracles for core logic).
- ✅ Transparent audit trail (Merkle-provable logs).
- ✅ Human-in-the-loop only for true ambiguity (Sacred Zero).

Limitations and Open Questions:

- ⚠️ **Oracle Risk for Auxiliary Data**: While core TML axioms are deterministic, context (e.g., real-world harms, ecosystem state) may depend on external data. Oracles can be compromised. Mitigations: Use multiple oracle sources; implement fraud detection thresholds; store off-chain proofs for later verification.

- ⚠️ **Sacred Zero Explosion**: If many decisions map to Sacred Zero, governance becomes a bottleneck. Solution: Refine axiom specificity during design phase; use probabilistic thresholds.
- ⚠️ **Constitutional Updating**: The Mandated Corpora (40+ documents, 26+ protocols) are fixed at deployment. If international law evolves, the contract remains locked to original documents. Solution: Plan for periodic contract redeployment on long governance cycles (e.g., 5-year audits).

Conclusion

The TML smart contract represents a technical embodiment of constrained governance: **code that enforces ethics, not merely legality**. By embedding axioms as immutable constants, restricting Council authority to ambiguity resolution, and implementing an irreversible kill switch for integrity violations, the architecture ensures that "Logic is Constitution" becomes a realized property, not a rhetorical ideal.

The shift from "Code is Law" to "Logic is Constitution" marks a transition from automation to ethics-enabled automation—systems that can say *no* with mathematical certainty and can defer judgment (Sacred Zero) to humans only when genuine uncertainty exists.

References

[3] - State Machine Design Pattern in Solidity
[12] - Deep dive into Merkle proofs and eth_getProof
[4] - Solidity Design Patterns
[5] - Smart Contract Design Patterns
[22] - Policy Enforcement via Smart Contracts
[13] - Merkle Patricia Tries
[6] - Access Control Patterns
[20] - Killswitch and Circuit Breaker
[23] - Polygon Multisig Upgrade Governance
[7] - OpenZeppelin Access Control
[8] - Smart Contracts & Incident Response
[9] - Upgradeable Smart Contracts
[21] - DeFi Circuit Breakers
[24] - Timelock Contracts
[25] - Keccak256 Hash Function
[26] - Commitment Schemes
[27] - Keccak256

[28] - Commitment Schemes
[29] - OpenZeppelin Multisig
[30] - DAO Governance & Dispute Resolution
[31] - Human-in-Loop Automation
[32] - Securely Deploy and Upgrade
[33] - DAO Governance
[34] - Human-in-Loop for Regulated Industries
[35] - DAO Governance Models
[36] - Human-in-Loop AI
[15] - Blockchain Log Auditing
[17] - Immutable Audit Trails
[18] - Ethics Framework for Automated Decision-Making
[16] - Leveraging Blockchain for Immutable Logging
[1] - Gas Optimization (SSTORE costs)
[2] - Storage Layout
[10] - Gas Optimization
[11] - Solidity Gas Optimization
[37] - Zero-Knowledge Proof
[38] - ZK-Proof Trustless Oracles
[19] - Smart Contract Security
[39] - Zero-Knowledge Proof Chainlink
[40] - Hashing in Digital Signatures
[41] - Blockchain Document Verification
[42] - Blockchain Legal Document Security
[43] - Blockchain Document Verification
[14] - Batch Hashing
[44] - Blockchain for Evidence
[45] - Blockchain Government Authentication
[46] - Merkle Trees
[47] - Escalation Policies
[48] - Blockchain Dispute Resolution
[49] - Blockchain Consensus Voting
[50] - On-chain Accountability
[51] - Blockchain Dispute Resolution
[52] - Arbitrating Smart Contract Disputes

Report Generated: December 28, 2025
Platform: EVM (Ethereum Virtual Machine)
Consensus Mechanism: Immutable Axioms + Constrained Human Governance
Constitutional Reference: 40+ Human Rights Documents, 26+ Earth Protection Protocols

Citations:
[1] Gas Optimization in Ethereum Smart Contracts: 10 Best Practices

https://www.certik.com/resources/blog/gas-optimization-in-ethereum-smart-contracts-10-best-practices
[2] Layout of State Variables in Storage and Transient ... - Solidity Docs
https://docs.soliditylang.org/en/latest/internals/layout_in_storage.html
[3] State Machine Design Pattern in Solidity | by Luis Soares
https://blog.blockmagnates.com/state-machine-design-pattern-in-solidity-1d81569a7833
[4] A developer's guide to Solidity design patterns - LogRocket Blog
https://blog.logrocket.com/developers-guide-solidity-design-patterns/
[5] Smart Contract Design Patterns in Solidity: A Developer's Guide 2025
https://metana.io/blog/smart-contract-design-patterns-in-solidity-explained/
[6] solidity ownable contract, contract ownership, access control patterns
https://favourajaye.hashnode.dev/understanding-contract-ownership-in-solidity-and-access-control-patterns
[7] Access Control - OpenZeppelin Docs
https://docs.openzeppelin.com/contracts/3.x/access-control
[8] Smart Contracts & Incident Response: Insight on Current Mechanisms
https://www.openzeppelin.com/news/smart-contracts-incident-response-insight-on-current-mechanisms
[9] Upgradeable Smart Contracts: Proxies, Patterns, Pitfalls and CI/CD ...
https://www.octane.security/post/upgradeable-smart-contracts-proxies-patterns-pitfalls-cicd-safeguards
[10] Gas Optimization in Solidity: Reduce Contract Costs 80% - Calmops
https://calmops.com/programming/blockchain/gas-optimization-solidity/
[11] 12 Solidity Gas Optimization Techniques - Alchemy
https://www.alchemy.com/overviews/solidity-gas-optimization
[12] Deep dive into Merkle proofs and eth_getProof - Chainstack Docs
https://docs.chainstack.com/docs/deep-dive-into-merkle-proofs-and-eth-getproof-ethereum-rpc-method
[13] Merkle Patricia Tries: A Deep Dive into Data Structure Security
https://cardanofoundation.org/blog/merkle-patricia-tries-deep-dive
[14] [PDF] Using Batch Hashing for Signing and Time-Stamping - Cybernetica
https://cyber.ee/uploads/T_4_20_Using_Batch_Hashing_for_Signing_and_Time_Stamping_3_copy_473794cc10.pdf
[15] [PDF] A blockchain-based log auditing approach for large-scale systems
https://arxiv.org/pdf/2505.17236.pdf
[16] Leveraging blockchain for immutable logging and querying across ...
https://pmc.ncbi.nlm.nih.gov/articles/PMC7372773/
[17] Immutable Audit Trails with Blockchain | RecordsKeeper.AI
https://www.recordskeeper.ai/immutable-audit-trails/
[18] Ethics, Transparency and Accountability Framework for Automated ...
https://www.gov.uk/government/publications/ethics-transparency-and-accountability-framework-for-automated-decision-making/ethics-transparency-and-accountability-framework-for-automated-decision-making
[19] 12 Solidity Smart Contract Security Best Practices - Alchemy

https://www.alchemy.com/overviews/smart-contract-security-best-practices
[20] Best Practices for Smart Contracts Security - LeewayHertz
https://www.leewayhertz.com/smart-contracts-security/
[21] DeFi Circuit Breakers With Chainlink Proof of Reserve and Automation
https://blog.chain.link/defi-circuit-breakers/
[22] Automating Policy Enforcement With Smart Contracts - Chainlink Blog
https://blog.chain.link/policy-enforcement-via-smart-contracts/
[23] Polygon Case Study: Mutli-Threshold Smart Contract Upgrades
https://www.aragon.org/how-to/polygon-case-study
[24] What Is a Timelock Contract? - Halborn
https://www.halborn.com/blog/post/what-is-a-timelock-contract
[25] How to Use Keccak256 Hash Function with Solidity - Quicknode
https://www.quicknode.com/guides/ethereum-development/smart-contracts/how-to-use-keccak256-with-solidity
[26] Commitment scheme - Wikipedia https://en.wikipedia.org/wiki/Commitment_scheme
[27] Keccak256 - Cyfrin Glossary https://www.cyfrin.io/glossary/keccak256
[28] [PDF] 1 Commitment Schemes and Coin Flipping
https://crypto.stanford.edu/cs355/23sp/lec3.pdf
[29] Leveraging OpenZeppelin Contracts and Defender to Build Secure ...
https://www.web3.university/article/leveraging-openzeppelin-contracts-and-defender-to-build-secure-multisig-owned-nfts
[30] How to Phase in a Governance Structure for DAOs - Montague Law
https://montague.law/blog/phase-in-governance-structure-daos/
[31] What is Human in the Loop Automation - Camunda
https://camunda.com/blog/2024/06/what-is-human-in-the-loop-automation/
[32] Securely deploy and upgrade a smart contract - OpenZeppelin Docs
https://docs.openzeppelin.com/defender/tutorial/deploy
[33] Chapter 3: DAO Governance and Dispute Resolution
https://www.worldscientific.com/doi/10.1142/9789811295799_0003
[34] Human-in-the-Loop Automation for Regulated Industries: Guide
https://www.multimodal.dev/post/human-in-the-loop-automation
[35] DAO Governance Models 2024: Ultimate Guide to Token vs ...
https://www.rapidinnovation.io/post/dao-governance-models-explained-token-based-vs-reputation-based-systems
[36] Understanding Human in the Loop AI | Scoop Analytics
https://www.scoopanalytics.com/blog/human-in-the-loop-ai
[37] Zero-knowledge proof - Wikipedia https://en.wikipedia.org/wiki/Zero-knowledge_proof
[38] Lido Grants - Funding & Exploration of ZK-Proof Trustless Oracles
https://blog.lido.fi/grants-exploration-zkproof-trustless-oracles/
[39] Zero-Knowledge Proof (ZKP) — Explained - Chainlink
https://chain.link/education/zero-knowledge-proof-zkp
[40] Role of Hashing Algorithms in Digital Signature Security - Certinal
https://www.certinal.com/blog/where-hashing-algorithms-fit-in-the-process-of-securing-digital-signatures

[41] Blockchain for Document Verification and Notarization - PowerPatent
https://powerpatent.com/blog/blockchain-for-document-verification-and-notarization
[42] Implementing Blockchain for Legal Document Security - Zircon Tech
https://zircon.tech/blog/implementing-blockchain-for-legal-document-security/
[43] Revolutionizing Document Security with Blockchain in Government
https://zircon.tech/blog/revolutionizing-document-security-with-blockchain-in-government/
[44] How Blockchain Enhances Hashing for Evidence | ScoreDetect Blog
https://www.scoredetect.com/blog/posts/how-blockchain-enhances-hashing-for-evidence
[45] Using Blockchain to Streamline Government Document Authentication
https://www.recordskeeper.ai/blockchain-government-authentication/
[46] Merkle Trees: Building Blocks of Blockchain Trust - Lightspark
https://lightspark.com/glossary/merkle-tree
[47] Proven escalation policy framework (w/ templates & checklists)
https://hyperping.com/blog/escalation-policies-guide
[48] Blockchain Dispute Resolution - Web3 - Meegle
https://www.meegle.com/en_us/topics/web3/blockchain-dispute-resolution
[49] Blockchain consensus algorithms Present Trends : voting Based ...
https://www.linkedin.com/pulse/blockchain-consensus-algorithms-voting-based-garima-singh-odpcf
[50] Accountability protocols? On-chain dynamics in blockchain ...
https://policyreview.info/articles/analysis/chain-dynamics-blockchain-governance
[51] Dispute Resolution on the Blockchain: Benefits & Implementation ...
https://tokenminds.co/blog/blockchain-dispute-resolution
[52] Arbitrating smart contract disputes: A comprehensive approach
https://www.dailyjournal.com/articles/382578-arbitrating-smart-contract-disputes-a-comprehensive-approach
[53] Smart Contracts Revisited: Lessons From the Courts in 2025
https://www.sideman.com/smart-contracts-revisited-lessons-from-the-courts-in-2025/
[54] State Legislation Bolsters Case for Smart Contract Enforceability
https://www.icertis.com/research/blog/state-legislation-bolsters-case-for-smart-contract-enforceability/
[55] Merkle Tree Proof of Reserves - The Network Firm
https://www.thenetworkfirm.com/merkle-tree-proof-of-reserves-for-crypto-blockchain-auditing-the-network-firm
[56] Smart Contract Design Patterns Explained | Hedera
https://hedera.com/learning/smart-contracts/smart-contract-design-patterns
[57] Writing Ethereum Smart Contracts for Role based permission
https://stackoverflow.com/questions/45613646/writing-ethereum-smart-contracts-for-role-based-permission
[58] Smart Contract Access Control Best Practices - Krayon Digital
https://www.krayondigital.com/blog/smart-contract-access-control-best-practices
[59] [PDF] Data Integrity Verification in Network Slicing using Oracles ... - arXiv
https://arxiv.org/pdf/2207.14388.pdf
[60] Oracle Risk and Security Standards: Data Replicability (Pt. 4)

https://chaoslabs.xyz/posts/oracle-data-replicability-pt-4
[61] Contracts — Solidity 0.8.34-develop documentation
https://docs.soliditylang.org/en/latest/contracts.html
[62] What are Oracles on Smart Contracts? - Stellar.org
https://stellar.org/learn/smart-contract-basics-oracles
[63] Learning about commitment schemes : r/crypto - Reddit
https://www.reddit.com/r/crypto/comments/x82ndg/learning_about_commitment_schemes/
[64] [PDF] Tamper-Proof Document Storage Using IPFS and Keccak ... - SSRN
https://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID5894362_code3635775.pdf?abstractid=5894362&mirid=1
[65] Deploying a Smart Contract via the Call method in a MultiSig Smart ...
https://forum.openzeppelin.com/t/deploying-a-smart-contract-via-the-call-method-in-a-multisig-smart-contract/34239
[66] Multisig - OpenZeppelin Docs
https://old-docs.openzeppelin.com/contracts-cairo/0.20.0/governance/multisig
[67] [PDF] Administrative Regulation of Programmatic Policing
https://scholarship.law.vanderbilt.edu/cgi/viewcontent.cgi?article=2438&context=faculty-publications
[68] Automated Decision-Making and Regulatory Legislation in AI Systems
https://gdprlocal.com/navigating-the-ai-contradictions/
[69] The President's Constitutional Power of Impoundment
https://americarenewing.com/the-presidents-constitutional-power-of-impoundment/
[70] Securing the Future: Blockchain-Based Audit Trails in IAM ...
https://mojoauth.com/ciam-101/blockchain-audit-trails-iam-passwordless-threat-breach
[71] [PDF] Constitutional Constraints - U.S. Chamber Institute for Legal Reform
https://instituteforlegalreform.com/wp-content/uploads/2020/10/ConstitutionalConstraints_web.pdf
[72] What is AI Governance? | IBM https://www.ibm.com/think/topics/ai-governance
[73] Incorrect Event Emission in Solidity: Risks and Best Practices
https://www.vibraniumaudits.com/post/incorrect-event-emission-in-solidity-risks-and-best-practices
[74] What is Smart Contract Storage Layout? | Alchemy Docs
https://www.alchemy.com/docs/smart-contract-storage-layout
[75] Ultimate Smart Contract Audit Checklist - 2024 - Rapid Innovation
https://www.rapidinnovation.io/post/complete-checklist-for-smart-contract-audit
[76] Optimize Gas on Base and Other L2 Chains With This Solidity Tutorial
https://www.cyfrin.io/blog/solidity-gas-efficiency-tips-tackle-rising-fees-base-other-l2
[77] Ethereum Virtual Machine Internals – Part 2 - NetSPI
https://www.netspi.com/blog/technical-blog/blockchain-pentesting/ethereum-virtual-machine-internals-part-2/
[78] Secure Smart Contract Design: Best Practices in Solidity Programming
https://agilie.com/blog/secure-smart-contract-design-best-practices-in-solidity-programming
[79] Solidity inheritance override public constant - Stack Overflow
https://stackoverflow.com/questions/69931208/solidity-inheritance-override-public-constant

[80] [PDF] Oracle Communications Fraud Monitor
https://www.oracle.com/a/ocom/docs/industries/communications/oracle-comms-fraud-monitor-ds.pdf
[81] Support for immutable in function parameters · Issue #12303 - GitHub
argotorg/solidity#12303
[82] [PDF] Oracle Enterprise Telephony Fraud Monitor
https://www.oracle.com/a/ocom/docs/industries/communications/comm-fraud-monitor-ent-ds.pdf
[83] A Two-Step Process for Detecting Fraud using Oracle ... - YouTube
https://www.youtube.com/watch?v=QlfbWJHz3pk
[84] Data Verification Methods And Systems Using A Hash Tree, Such As ...
https://patents.google.com/patent/US20170075938A1/en
[85] What is Hashing in Cyber Security & How Does it Work?
https://cmitsolutions.com/blog/what-is-hashing-in-cyber-security/
[86] Holographic Consensus: a scalable voting system in large ...
https://p2pmodels.eu/holographic-consensus-a-scalable-voting-system/
[87] Understanding Governance Contracts - StableLab
https://stablelab.xyz/blog/understanding-governance-contracts