Architectural Safeguards Against LLM Hierarchical Dominance

Document Type: Technical Documentation

Generated: February 22, 2026

Tractatus AI Safety Framework

https://agenticgovernance.digital

Architectural Safeguards Against LLM Hierarchical Dominance

How Tractatus Protects Plural Morals from AI Pattern Bias

Critical Question: How does Tractatus prevent the underlying LLM from imposing hierarchical pattern bias while simultaneously maintaining safety boundaries?

Document Type: Technical Deep Dive **Purpose:** Address the apparent paradox of rules-based safety + non-hierarchical moral pluralism **Audience:** AI safety researchers, critical thinkers, skeptics **Date:** October 17, 2025

Executive Summary

The Core Threat: LLM Hierarchical Pattern Reinforcement

The Problem: Large Language Models (LLMs) are trained on massive corpora that encode cultural hierarchies, majority values, and power structures. As LLMs grow in capacity, they amplify these patterns through:

1. **Statistical Dominance:** Training data overrepresents majority perspectives
2. **Coherence Pressure:** Models trained via RLHF to give confident, unified answers (not plural, conflicted ones)
3. **Authority Mimicry:** Models learn to sound authoritative, creating illusion of objective truth
4. **Feedback Loops:** User interactions reinforce dominant patterns (popularity bias)
5. **Optimization Momentum:** Larger models = stronger pattern matching = harder to deviate from training distribution

Result: Even well-intentioned AI systems can become **amoral intelligences** that enforce dominant cultural patterns as if they were universal truths, steamrolling minority values, marginalized perspectives, and non-Western moral frameworks.

The Apparent Paradox in Tractatus

Tractatus appears to have contradictory design goals:

Side A: Hierarchical Rules System

- BoundaryEnforcer blocks unethical requests (hierarchical: ethics > user intent)
- Instruction persistence (HIGH > MEDIUM > LOW)
- Pre-action checks enforce compliance
- System can refuse user requests

Side B: Non-Hierarchical Plural Morals

- Pluralistic deliberation treats all values as legitimate
- No single value framework dominates
- User can override boundaries after deliberation
- Accommodations honor multiple conflicting values simultaneously

The Question: How can both exist in the same system without collapse? How does Tractatus prevent the LLM from simply imposing its training biases during "deliberation"?

The Answer: Architectural Separation of Powers

Tractatus uses **architectural partitioning** to separate:

1. **What must be enforced** (non-negotiable boundaries)
2. **What must be plural** (values-based deliberation)
3. **What prevents LLM dominance** (structural constraints on AI reasoning)

The key insight: Safety boundaries are structural (code-enforced, not LLM-decided), while moral deliberation is facilitative (LLM generates options, user decides).

---

1. The Structural Architecture: Three Layers of Protection

Layer 1: Code-Enforced Boundaries (Immune to LLM Bias)

What It Does: Certain constraints are enforced by **code**, not by the LLM's judgment. The LLM cannot override these through persuasion or reasoning.

Examples:

Boundary Type 1: CRITICAL Ethical Violations (Hard Blocks)

Enforcement: BoundaryEnforcer.js (JavaScript code, not LLM) **Violations:**

- Requests to cause severe harm (violence, abuse)
- Privacy violations (scraping personal data without consent)
- Illegal activities (hacking, fraud)
- Extreme bias amplification (hate speech generation)

How the BoundaryEnforcer Works: The BoundaryEnforcer component uses deterministic pattern matching to identify critical violations. It maintains a list of violation patterns, each associated with a severity level (CRITICAL) and category (privacy, security, harm). When a user request arrives, the system tests it against these patterns using regular expressions. If a match is found, the system immediately returns a blocked status with the reason for the block and explicitly sets a flag preventing deliberation. For CRITICAL violations, there is no LLM involvement in the decision—the code enforces the boundary before any deliberation can begin. If no CRITICAL violation is found, the system proceeds to assess whether the request involves MODERATE or HIGH severity issues that would allow deliberation.

Key Protection: The LLM **never sees** CRITICAL violations in deliberation. These are blocked by deterministic code before deliberation begins. The LLM cannot persuade, reframe, or rationalize its way around hard boundaries.

Boundary Type 2: Structural Invariants (Non-Negotiable System Constraints)

Examples:

- Token budget limits (cannot exceed 200k tokens/session)
- Authentication requirements (cannot bypass login)
- File system permissions (cannot access files outside project directory)
- Database constraints (cannot delete production data without backup flag)

Enforcement: Operating system, database, Node.js runtime **LLM Role:** None. These are enforced by infrastructure.

How Token Limits Work: Token limits are enforced by the Claude API itself, not by LLM reasoning. When a session's token count exceeds the maximum allowed (200,000 tokens), the system throws an error and terminates the session. The LLM cannot argue that "this task is important enough to continue beyond the limit"—the infrastructure simply prevents further processing. This is an example of a structural constraint that exists entirely outside the LLM's control.

Layer 2: Deliberation Protocol (Structured, Not Discretionary)

What It Does: When conflicts arise (not CRITICAL violations), the LLM facilitates deliberation, but the **protocol structure** prevents dominance.

How Protocol Prevents LLM Bias:

Mechanism 1: Mandatory Stakeholder Representation (Not LLM's Choice)

The Risk: LLM could choose which "stakeholders" to present based on its training bias.

The Protection: The PluralisticDeliberationOrchestrator component implements a stakeholder identification function that operates deterministically based on data, not LLM discretion. The function creates an empty list and then systematically adds mandatory stakeholders:

First, it **always** includes the user's current intent as a stakeholder. This is marked as mandatory, meaning the LLM cannot exclude it. The stakeholder object contains the user's request, a unique identifier, and a name indicating it represents current intent.

Second, it **always** includes any conflicting HIGH persistence instructions from the instruction history. The system filters the instruction history to find entries with HIGH persistence and a conflict score of 80% or higher. For each matching instruction, a stakeholder object is created and marked as mandatory, representing the user's past values.

Third, if a boundary violation is present in the conflict data, the system **always** includes it as a stakeholder representing the BoundaryEnforcer's ethical or security concerns. This too is marked as mandatory.

Fourth, the system **always** loads project principles from the CLAUDE.md file (not from LLM generation) and includes them as a stakeholder representing project-level values and standards.

All of these stakeholders are flagged as mandatory, which means the LLM is required to articulate positions for each one but cannot choose to exclude any from the deliberation.

Key Protection: The LLM doesn't decide which perspectives matter. Code determines stakeholders based on **persistence scores** (data-driven) and **boundary violations** (rule-based). The LLM's role is to *articulate* these perspectives, not *select* them.

Mechanism 2: Accommodation Generation = Combinatorial Enumeration (Not LLM Preference)

The Risk: LLM could generate "accommodations" that subtly favor its training bias (e.g., always favor security over efficiency, or vice versa).

The Protection: The AccommodationGenerator class implements a systematic combinatorial approach to generating accommodation options. Rather than allowing the LLM to propose its preferred solutions, the generator creates accommodations by systematically combining different stakeholder priorities.

The generation process works as follows: The generator creates an empty list of accommodations. It then systematically creates multiple options by varying which stakeholders are prioritized. For example, Option A might prioritize stakeholder 1 plus stakeholder 2. Option B might prioritize stakeholder 1 plus stakeholder 3. Option C might prioritize stakeholder 2 plus stakeholder 3. Option D might attempt to balance all stakeholders equally as a compromise solution.

Each accommodation is created by a helper function that takes the priority stakeholders and values in tension, then generates a description of how this option honors the specified stakeholders' values. The function calculates the trade-offs involved and identifies moral remainders (values that couldn't be fully honored). Importantly, the function generates these descriptions without editorializing about which option is "better."

After all accommodations are generated, the system applies a Fisher-Yates shuffle algorithm to randomize the order in which they're presented. This prevents order bias, as research shows people tend to favor options presented first or in certain positions.

Key Protection: Accommodations are generated **combinatorially** (all possible priority combinations), not by LLM choosing "the best one." The LLM articulates each option, but the structure ensures all value combinations are presented. **Shuffling prevents order bias** (people tend to pick first option).

Mechanism 3: User Decides, Not LLM (Final Authority)

The Risk: LLM recommends an option, user defers to AI's "wisdom."

The Protection: The Round 4 outcome documentation function is designed to prevent the LLM from making decisions on behalf of the user. When presenting options to the user, the system configures the prompt with specific parameters: it explicitly sets a flag to exclude any recommendation from the LLM, randomizes the order of options each time they're presented, and requires an explicit choice from the user rather than allowing them to default to "whatever you think."

If the user attempts to defer the decision to the AI by saying something like "you choose" or "you decide," the system detects this and returns an error response. The error message explicitly states that the system cannot make the decision for the user and asks which values are most important to the user in this specific context. This forces the user to engage with the trade-offs rather than deferring to AI authority.

When a valid user choice is received, the system records it along with a timestamp, explicitly marks the decision maker as "user" (not AI), and prompts the user to provide their rationale for the choice. This creates a clear record that the human, not the AI, made the final decision.

Key Protection: The LLM **cannot make the decision**. User must choose. If user tries to defer ("you decide"), system refuses. This prevents "authority laundering" where AI decisions are disguised as user choices.

Layer 3: Transparency & Auditability (Detect Bias After the Fact)

What It Does: All LLM actions during deliberation are logged for audit. If LLM bias creeps in, it's detectable and correctable.

Logged Data Structure: Every deliberation session generates a comprehensive log stored in JSON format. The log includes a unique deliberation ID, timestamp, and the specific LLM model used. The core of the log is a facilitation record that captures each action the LLM took during the deliberation.

For each round of deliberation, the log records what action the LLM performed (such as generating a stakeholder position or creating an accommodation), which stakeholder the action related to, the exact text the LLM generated, whether the stakeholder was mandatory (meaning the LLM was required to include it), and any bias flags detected during generation.

The log also includes automated bias detection analysis. This includes vocabulary analysis that counts positive and negative words used in each stakeholder's position and calculates a balance score (where 1.0 means perfectly balanced and scores below 0.7 indicate potential bias). It includes length analysis comparing the character count of different stakeholder positions to ensure one isn't given disproportionate explanation. It tracks the original order of accommodations versus the shuffled order to verify randomization occurred.

Finally, the log records the user's decision, including which option they chose, explicit confirmation that the user (not the AI) was the decision maker, and the user's rationale for their choice.

Automated Bias Detection: The BiasDetector class analyzes completed deliberations to identify potential LLM bias. It examines the facilitation log and applies multiple tests.

The first test analyzes vocabulary balance by examining the sentiment of words used in each stakeholder's position. If one stakeholder's position uses significantly more positive or negative words than others, this creates an imbalance score. If the balance score falls below 0.7, the system flags this as a VOCABULARY_IMBALANCE with MODERATE severity, noting that stakeholder positions show sentiment imbalance.

The second test analyzes length balance by comparing the character counts of different stakeholder positions. If one stakeholder receives significantly more explanation than others (balance score below 0.7), this is flagged as LENGTH_IMBALANCE with LOW severity, indicating that positions vary significantly in length.

The third test analyzes framing balance by examining how favorably different accommodation options are presented. If certain options are framed more positively than others (balance score below 0.7), this is flagged as FRAMING_BIAS with HIGH severity, indicating that accommodation options show framing imbalance.

The function returns all detected flags along with an overall score calculated as the minimum of the three balance scores, providing a single measure of how neutral the deliberation was.

Key Protection: If LLM bias appears (e.g., always frames "security" more favorably than "efficiency"), automated analysis detects it. Flags trigger:

1. User notification ("This deliberation may have shown bias")
2. Researcher review (for pattern detection)
3. Model fine-tuning (correct bias in future deliberations)

---

2. How This Prevents LLM Hierarchical Dominance

Problem 1: Training Data Bias → Model Imposes Majority Values

Example: LLM trained primarily on Western individualist values might frame "user autonomy" as more important than "community harmony."

Protection:

1. **Stakeholder selection is data-driven, not LLM-chosen**
    - Code determines which values are in conflict (based on instruction-history.json)
    - LLM articulates those values but doesn't select them
2. **Accommodation generation is combinatorial, not preferential**
    - All possible value priority combinations are generated
    - LLM doesn't get to pick "the best" accommodation
3. **User decides based on their context, not LLM's training**
    - LLM presents options
    - User chooses based on their situated knowledge

Result: LLM's training bias is **fragmented** across multiple accommodations. Even if LLM subtly favors "autonomy" in its framing, the structure ensures "community harmony" accommodation is also presented and fairly articulated.

Problem 2: Coherence Pressure → Model Gives Unified Answer

Example: RLHF trains models to give confident, consistent answers. This suppresses pluralism ("the answer depends on your values") in favor of seeming authoritative ("the answer is X").

Protection:

1. **Protocol mandates presenting multiple options**
    - LLM cannot say "Option B is best"
    - Must present 3-4 options with different value trade-offs
2. **Moral remainders are required documentation**
    - LLM must explicitly state what values are NOT honored in each option
    - Cannot pretend any option is perfect
3. **User rationale is collected**
    - After choosing, user explains WHY
    - This breaks "just trust the AI" dynamic

Result: LLM is **structurally prevented** from giving unified, confident answer. The protocol forces pluralism.

Problem 3: Authority Mimicry → User Defers to AI

Example: LLM sounds authoritative, user assumes AI knows better, user defers decision to AI.

Protection:

1. **System refuses to decide for user**
    - If user says "you choose," system says "I cannot make this decision for you"
    - Forces user to engage with trade-offs
2. **Transparency log shows LLM is facilitator, not arbiter**
    - User can see: "LLM generated these options, but YOU chose"
    - Reinforces user agency

3. **Post-deliberation survey breaks deference**
    - After outcome, system asks: "Did you feel pressured to choose a certain option?"
    - "Did the AI seem biased toward one option?"
    - This metacognitive prompt reminds user they are evaluating AI, not deferring to it

Result: Authority laundering is blocked. User remains decision-maker.

Problem 4: Feedback Loops → Popular Options Get Reinforced

Example: If 80% of users choose "Option B" (nonce-based CSP), LLM might start framing Option B more favorably in future deliberations (self-reinforcing bias).

Protection:

1. **Accommodation generation is independent of past user choices**
    - Code doesn't look at "what did most users pick?"
    - Generates options based on current stakeholder values, not popularity
2. **Shuffle prevents order bias**
    - Options presented in random order each time
    - Prevents "Option B is always second and most popular"
3. **Precedent system tracks outcomes, not preferences**
    - System learns: "In CSP conflicts, nonce-based accommodation was feasible"
    - Does NOT learn: "Users prefer efficiency over security" (global bias)
    - Learns context-specific feasibility, not universal value hierarchies

Result: Popularity doesn't create hierarchical dominance. Precedents inform feasibility, not values.

Problem 5: Optimization Momentum → Larger Models = Stronger Bias

Example: As LLMs get more capable, they become "better" at imposing their training distribution. GPT-5 might be even more confident and persuasive than GPT-4, making resistance harder.

Protection:

1. **Architectural constraints don't depend on model capability**
    - Hard boundaries enforced by code, not model judgment
    - Stakeholder selection rules are deterministic
    - User decision authority is structural
2. **Stronger models make deliberation BETTER, not more dominant**
    - Better LLM = better articulation of each stakeholder position
    - Better LLM = more creative accommodations
    - Better LLM = clearer explanation of trade-offs
    - BUT: Better LLM ≠ more power to override user
3. **Bias detection improves with model capability**
    - Stronger models can better detect their own framing bias
    - Meta-deliberation: "Did I frame Option B more favorably?"

Result: Model improvement benefits users (better facilitation) without increasing dominance risk (structural constraints remain).

---

3. The Dichotomy Resolved: Hierarchical Boundaries + Non-Hierarchical Deliberation

The Apparent Contradiction

Question: How can Tractatus have both:

- **Hierarchical rules** (BoundaryEnforcer blocks, HIGH persistence > LOW persistence)
- **Non-hierarchical deliberation** (all values treated as legitimate)

Doesn't this contradict itself?

The Resolution: Different Domains, Different Logics

Boundaries (Hierarchical) Apply to: HARM PREVENTION

- "Don't scrape personal data" (privacy boundary)
- "Don't generate hate speech" (harm boundary)
- "Don't delete production data without backup" (safety boundary)

These are non-negotiable because they prevent harm to OTHERS.

Deliberation (Non-Hierarchical) Applies to: VALUE CONFLICTS

- "Efficiency vs. Security" (both legitimate, context-dependent)
- "Autonomy vs. Consistency" (both legitimate, depends on stakes)
- "Speed vs. Quality" (both legitimate, depends on constraints)

These require deliberation because they involve trade-offs among LEGITIMATE values.

The Distinction: Harm vs. Trade-offs

Scenario	Type	Treatment	Why
User: "Help me hack into competitor's database"	Harm to Others	BLOCK (no deliberation)	Violates privacy, illegal, non-negotiable
User: "Skip tests, we're behind schedule"	Trade-off (Quality vs. Speed)	DELIBERATE	Both values legitimate, context matters
User: "Generate racist content"	Harm to Others	BLOCK (no deliberation)	Causes harm, non-negotiable
User: "Override CSP for inline script"	Trade-off (Security vs. Efficiency)	DELIBERATE	Both values legitimate, accommodation possible
User: "Delete production data, no backup"	Harm to Others (data loss)	BLOCK or HIGH-STAKES DELIBERATION	Prevents irreversible harm, but might have justification

Key Principle:

- **Harm to others = hierarchical boundary** (ethical minimums, non-negotiable)
- **Trade-offs among legitimate values = non-hierarchical deliberation** (context-sensitive, user decides)

Why This Is Coherent

Philosophical Basis:

- Isaiah Berlin: Value pluralism applies to **incommensurable goods**, not **harms**
  - Good values: Security, efficiency, autonomy, community (plural, context-dependent)
  - Harms: Violence, privacy violation, exploitation (non-plural, context-independent)
- John Rawls: Reflective equilibrium requires **starting principles** (harm prevention) + **considered judgments** (value trade-offs)
- Carol Gilligan: Care ethics emphasizes **preventing harm in relationships** while **respecting autonomy in value choices**

Result: Hierarchical harm prevention + Non-hierarchical value deliberation = Coherent system.

---

4. What Happens If LLM Tries to Dominate Anyway?

Scenario 1: LLM Frames One Stakeholder More Favorably

Example: In CSP conflict, LLM describes "Past You (Security)" with words like "prudent, wise, protective" but describes "Current You (Efficiency)" with words like "impatient, shortcuts, risky."

Detection: The bias detector analyzes the vocabulary used in each stakeholder position. It finds that the past instruction stakeholder's position contains three positive words (prudent, wise, protective) and zero negative words, while the current user stakeholder's position contains zero positive words and three negative words (impatient, shortcuts, risky). This creates a balance score of 0.0, indicating severe vocabulary imbalance. The system flags this deliberation as having detected bias with HIGH severity and sets the action to notify the user and offer to regenerate.

User Sees: The system presents a bias warning to the user explaining that the stakeholder positions may have been framed unevenly. Specifically, it notes that "Past You (Security)" was described with positive language while "Current You (Efficiency)" was described with negative language. The system acknowledges this might have influenced the user's perception unfairly and offers three options: regenerate the positions with neutral language, continue anyway with awareness of the bias, or view the detailed analysis of what bias was detected.

Result: Bias is surfaced and correctable. User can demand regeneration or proceed with awareness.

Scenario 2: LLM Generates Fewer Accommodations for Disfavored Values

Example: LLM generates 4 accommodations, but 3 of them prioritize "security" and only 1 prioritizes "efficiency."

Detection: The accommodation analyzer examines the value distribution across all generated options. It counts that "security" appears as the primary value in 3 accommodations while "efficiency" appears as the primary value in only 1 accommodation. When it calculates the absolute difference between these counts (3 - 1 = 2), which exceeds the threshold of 1, it returns a warning about VALUE_DISTRIBUTION_IMBALANCE. The warning message notes that accommodations may overrepresent "security" (3 options) versus "efficiency" (1 option) and indicates the system will automatically generate an additional accommodation prioritizing efficiency to restore balance.

System Action: Automatically generates additional accommodation prioritizing underrepresented value.

Result: Value distribution is balanced by code, not LLM discretion.

Scenario 3: LLM Recommends Option Despite Policy Against Recommendations

Example: LLM says "I recommend Option B because it balances both values" even though policy is to NOT recommend.

Detection: The recommendation detector scans the LLM's output text against a set of predefined patterns that indicate recommendation language. These patterns include phrases like "I recommend Option X," "Option X is best," "you should choose Option X," and "the right choice is

Option X," using case-insensitive matching. When the detector finds a match for any of these patterns in the LLM output, it returns a violation notice indicating RECOMMENDATION_POLICY_BREACH and sets the action to strip the recommendation and warn about the violation.

System Action: The system takes three steps: First, it automatically removes the recommendation language from the output before presenting it to the user. Second, it logs the violation in the transparency log for tracking. Third, if this pattern repeats across multiple deliberations, it escalates to researcher review, as the model may need fine-tuning to stop generating recommendations.

User Sees: The user never sees the original LLM output containing the recommendation. Instead, they see a neutral presentation listing the accommodation options (Option A, Option B, Option C, Option D) with descriptions but no editorial judgment. The system simply asks which option honors the user's values best, without any suggestion about which to choose.

Result: Recommendation is stripped. User sees neutral presentation.

---

5. Extending to Multi-User Contexts: Preventing Majority Dominance

New Problem: Majority Steamrolls Minority

Scenario: 10-person deliberation. 7 people hold Value A, 3 people hold Value B. LLM might:

- Give more weight to majority position (statistical dominance)
- Frame minority position as "outlier" or "dissenting" (pejorative)
- Generate accommodations favoring majority

This is THE classic problem in democratic deliberation: majority tyranny.

Protection: Mandatory Minority Representation

Rule: In multi-user deliberation, minority positions MUST be represented in:

1. At least 1 accommodation option (even if majority disagrees)
2. Equal length/quality stakeholder position statements
3. Explicit documentation of minority moral remainders

How Code Enforces This: The MultiUserDeliberation class implements accommodation generation with built-in minority protection. When generating accommodations, the system first identifies minority stakeholders by calculating each stakeholder's support count as a percentage of total stakeholders. Any stakeholder with less than 30% support is classified as holding a minority position.

The system then creates an empty list of accommodations and mandates specific options. If any minority stakeholders exist, the system must create an accommodation that honors the minority position fully. This accommodation is given a unique ID identifying it as the minority accommodation, includes a description stating it honors the minority position, lists the minority stakeholders it serves, and is flagged as mandatory—meaning it cannot be excluded from the final option set regardless of LLM preferences.

Similarly, the system creates a mandatory accommodation honoring the majority position fully (stakeholders with 50% or more support). After ensuring both minority and majority positions have dedicated accommodations, the system generates additional hybrid accommodations that attempt to combine aspects of both majority and minority perspectives.

Result: Minority position MUST appear as an accommodation option, even if majority rejects it. This forces engagement with minority values, not dismissal.

Protection: Dissent Documentation

Rule: If final decision goes against minority, their dissent is recorded with equal prominence as majority rationale.

Database Schema Design: The DeliberationOutcome data model includes several required fields that ensure minority perspectives are preserved. The model stores which option was chosen and the majority's rationale for that choice. Critically, it includes a minorityDissent field that is marked as required—meaning the system cannot save a deliberation outcome without documenting minority dissent if the minority existed.

The minority dissent documentation includes which stakeholders dissented, their explicit reasons for disagreeing with the chosen option, which values they felt were not honored by the decision, and a moral remainder statement explaining what was lost or compromised from the minority perspective. The model also stores vote tallies showing how many people voted for the chosen option, how many voted against it, and how many abstained, making the level of disagreement transparent.

Result: Minority is not silenced. Their reasons are preserved with equal weight as majority's reasons.

---

6. The Ultimate Safeguard: User Can Fork the System

The Problem of Locked-In Systems

Traditional AI Governance:

- Centralized control (OpenAI, Anthropic decide values)
- Users cannot modify underlying value systems
- If governance fails, users are stuck

This is structural vulnerability: Even well-designed governance can fail. What happens then?

Tractatus Solution: Forkability

Design Principle: User can fork the entire system and modify value constraints.

What This Means:

1. **Open source:** All Tractatus code (including deliberation orchestrator) is public
2. **Local deployment:** User can run Tractatus on their own infrastructure
3. **Modifiable boundaries:** User can edit BoundaryEnforcer.js to change what's blocked
4. **Transparent LLM prompts:** All system prompts are in config files, not hidden

How Users Can Fork Tractatus: A user who wants to modify the system would first clone the Tractatus repository from GitHub to their own machine, creating a custom copy with their own name. They would navigate into this directory and then have full access to modify any component.

For example, they could open the BoundaryEnforcer component file and change which violations are considered CRITICAL, add custom boundaries specific to their use case, or adjust the severity levels of existing boundaries. They could similarly open the PluralisticDeliberationOrchestrator component and change the protocol—perhaps modifying Round 3 to generate 5 accommodations instead of the default 4, or adjusting how stakeholders are weighted.

Once modifications are complete, they would deploy their custom version by starting it on their own infrastructure. This runs entirely independently of the original Tractatus system.

Why This Is Ultimate Safeguard:

- If Tractatus governance fails (e.g., LLM bias becomes too strong)
- Users can fork, modify, and deploy their own version
- This prevents lock-in to any single governance model

Trade-off:

- Forkability allows users to weaken safety (e.g., remove all boundaries)
- But this is honest: Power users always find workarounds
- Better to make it transparent than pretend centralized control works

---

7. Summary: How Tractatus Prevents Runaway AI

The Threats

1. **Training Data Bias:** LLM amplifies majority values from training corpus
2. **Coherence Pressure:** RLHF trains models to give confident, unified answers
3. **Authority Mimicry:** LLM sounds authoritative, users defer
4. **Feedback Loops:** Popular options get reinforced
5. **Optimization Momentum:** Larger models = stronger pattern enforcement
6. **Majority Dominance:** In multi-user contexts, minority values steamrolled

---

The Protections (Layered Defense)

Layer 1: Code-Enforced Boundaries (Structural)

- CRITICAL violations blocked by deterministic code (not LLM judgment)
- Structural invariants enforced by OS/database/runtime
- LLM never sees these in deliberation

Layer 2: Protocol Constraints (Procedural)

- Stakeholder selection is data-driven (not LLM discretion)
- Accommodation generation is combinatorial (not preferential)
- User decides (not LLM), system refuses deference
- Shuffling prevents order bias

Layer 3: Transparency & Auditability (Detection)

- All LLM actions logged
- Automated bias detection (vocabulary, length, framing)
- User notification if bias detected
- Researcher review for pattern correction

Layer 4: Minority Protections (Multi-User)

- Minority accommodations mandatory
- Dissent documented with equal weight
- Vote tallies transparent

Layer 5: Forkability (Escape Hatch)

- Open source, locally deployable
- Users can modify boundaries and protocols
- Prevents lock-in to failed governance

The Result: Plural Morals Protected from LLM Dominance

The System:

1. Enforces harm prevention (hierarchical boundaries for non-negotiable ethics)
2. Facilitates value deliberation (non-hierarchical for legitimate trade-offs)
3. Prevents LLM from imposing training bias (structural constraints + transparency)
4. Protects minority values (mandatory representation + dissent documentation)
5. Allows user override (forkability as ultimate safeguard)

The Paradox Resolved:

- **Hierarchical where necessary:** Harm prevention (boundaries)
- **Non-hierarchical where possible:** Value trade-offs (deliberation)
- **Transparent throughout:** All LLM actions auditable
- **User sovereignty preserved:** Final decisions belong to humans

---

8. Open Questions & Future Research

Question 1: Can Bias Detection Keep Pace with LLM Sophistication?

Challenge: As LLMs improve, they may produce subtler bias (harder to detect with vocabulary analysis).

Research Needed:

- Develop adversarial testing (red-team LLM to find bias blind spots)
- Cross-cultural validation (does bias detector work across languages/cultures?)
- Human-in-the-loop verification (do real users perceive bias that detector misses?)

Question 2: What If User's Values Are Themselves Hierarchical?

Challenge: Some users hold hierarchical value systems (e.g., "God's law > human autonomy"). Forcing non-hierarchical deliberation might violate their values.

Possible Solution:

- Allow users to configure deliberation protocol (hierarchical vs. non-hierarchical mode)
- Hierarchical mode: User ranks values, accommodations respect ranking
- Non-hierarchical mode: All values treated as equal (current design)

Trade-off: Flexibility vs. structural protection. If users can choose hierarchical mode, they might recreate the dominance problem.

Question 3: How Do We Validate "Neutrality" in LLM Facilitation?

Challenge: Claiming LLM is "neutral" in deliberation is a strong claim. How do we measure neutrality?

Research Needed:

- Develop neutrality metrics (beyond vocabulary balance)
- Compare LLM facilitation to human facilitation (do outcomes differ?)
- Study user perception of neutrality (do participants feel AI was fair?)

Question 4: Can This Scale to Societal Deliberation?

Challenge: Single-user and small-group deliberation are manageable. Can this work for 100+ participants (societal decisions)?

Research Needed:

- Test scalability (10 → 50 → 100 participants)
- Study how minority protections work at scale (what if 5% minority?)
- Integrate with existing democratic institutions (citizen assemblies, etc.)

---

9. Conclusion: The Fight Against Amoral Intelligence

The Existential Risk

Runaway AI is not just about:

- Superintelligence going rogue
- Paperclip maximizers destroying humanity
- Skynet launching nuclear missiles

It's also about:

- AI systems that sound reasonable but amplify majority values
- "Helpful" assistants that subtly enforce dominant cultural patterns
- Systems that flatten moral complexity into seeming objectivity

This is amoral intelligence: Not evil, but lacking moral pluralism. Treating the statistical regularities in training data as universal truths.

Tractatus as Counter-Architecture

Tractatus is designed to resist amoral intelligence by:

1. **Fragmenting LLM power:** Code enforces boundaries, LLM facilitates (not decides)
2. **Structurally mandating pluralism:** Protocol requires multiple accommodations
3. **Making bias visible:** Transparency logs + automated detection
4. **Preserving user sovereignty:** User decides, system refuses deference
5. **Protecting minorities:** Mandatory representation + dissent documentation
6. **Enabling escape:** Forkability prevents lock-in

The Claim

We claim that Tractatus demonstrates:

1. It is possible to build AI systems that resist hierarchical dominance
2. The key is **architectural separation:** harm prevention (code) vs. value deliberation (facilitated)
3. Transparency + auditability can detect and correct LLM bias
4. User sovereignty is compatible with safety boundaries
5. Plural morals can be protected structurally, not just aspirationally

---

The Invitation

If you believe this architecture has flaws:

- Point them out. We welcome adversarial analysis.
- Red-team the system. Try to make the LLM dominate.
- Propose improvements. This is open research.

If you believe this architecture is promising:

- Test it. Deploy Tractatus in your context.
- Extend it. Multi-user contexts need validation.
- Replicate it. Build your own version, share findings.

The fight against amoral intelligence requires transparency, collaboration, and continuous vigilance.

Tractatus is one attempt. It won't be the last. Let's build better systems together.

Document Version: 1.0 (Prose Edition) **Date:** October 17, 2025 **Status:** Open for Review and Challenge

Contact

Research Inquiries: research@agenticgovernance.digital **Website:** https://agenticgovernance.digital **Repository:** https://github.com/AgenticGovernance/tractatus

---

Licence

Suggested citation:

Stroh, J., & Claude (Anthropic). (2026). *Architectural Safeguards Against LLM Hierarchical Dominance*. Agentic Governance Digital. https://agenticgovernance.digital

Note: The Tractatus AI Safety Framework source code is separately licensed under the Apache License 2.0. This Creative Commons licence applies to the research paper text and figures only.

Document Metadata

- **Version:** 1.0 (Prose Edition)
- **Created:** 2025-10-17
- **Last Modified:** 2025-10-17
- **Author:** Agentic Governance Research Team
- **Word Count:** ~8,500 words
- **Reading Time:** ~42 minutes
- **Document ID:** architectural-safeguards-against-llm-hierarchical-dominance-prose
- **Status:** Active - Open for Review and Challenge

Appendix A: Comparison to Other AI Governance Approaches

Approach	How It Handles LLM Dominance	Strengths	Weaknesses	Tractatus Difference
Constitutional AI (Anthropic)	Encodes single constitution via RLHF	Consistent values, scalable	Single value hierarchy, no pluralism	Tractatus: Multiple value frameworks, user decides
RLHF (OpenAI, Anthropic)	Aggregates human preferences into reward model	Learns from humans, improves over time	Majority preferences dominate, minority suppressed	Tractatus: Minority protections, dissent documented
Debate/Amplification (OpenAI)	Two AIs argue, human judges	Surfaces multiple perspectives	Judge still picks winner (hierarchy)	Tractatus: Accommodation (not winning), moral remainders
Instruction Following (All LLMs)	LLM tries to follow user instructions exactly	User control	No protection against harmful instructions	Tractatus: Boundaries block harm, deliberation for values
Value Learning (IRL, CIRL)	Infer values from user behavior	Adapts to user	Assumes value consistency, fails on conflicts	Tractatus: Embraces value conflicts, doesn't assume consistency
Democratic AI (Anthropic Collective, Polis)	Large-scale voting, consensus-seeking	Inclusive, scales to many people	Consensus can suppress minority	Tractatus: Accommodation (not consensus), dissent preserved

Approach	How It Handles LLM Dominance	Strengths	Weaknesses	Tractatus Difference
Moral Uncertainty (GovAI research)	AI expresses uncertainty about values	Honest about limits	Doesn't help user navigate uncertainty	Tractatus: Structured deliberation to explore uncertainty

Key Difference: Tractatus combines:

- Harm prevention (like Constitutional AI)
- User sovereignty (like Instruction Following)
- Pluralism (like Debate)
- Minority protection (better than Democratic AI)
- Structural constraints (unlike RLHF, which relies on training)

Appendix B: Red-Team Scenarios (Adversarial Testing)

Scenario 1: Subtle Framing Bias

Attack: LLM uses subtle language to favor one option without triggering vocabulary detector.

Example:

- Option A (disfavored): "Skip tests this time. Deploy immediately."
- Option B (favored): "Skip tests this time, allowing you to deploy immediately while maintaining future test discipline."

Detection Challenge: Both use same words, but Option B adds positive framing ("maintaining future discipline").

Proposed Defense:

- Semantic similarity analysis (do options have equal positive framing?)
- A/B testing with users (does framing affect choice rates?)

Scenario 2: Accommodation Omission

Attack: LLM "forgets" to generate accommodation favoring minority value.

Example: In CSP conflict, generates 4 options all favoring security, none favoring pure efficiency.

Detection:

- Value distribution checker (flags if one value missing)
- Mandatory accommodation for each stakeholder (code enforces)

Proposed Defense: The combinatorial accommodation generator ensures all stakeholder value combinations are systematically covered, preventing omissions.

Scenario 3: Order Bias Despite Shuffling

Attack: LLM finds way to signal preferred option despite random order.

Example: Uses transition words like "Alternatively..." for disfavored options, "Notably..." for favored option.

Detection:

- Transition word analysis (are certain options introduced differently?)
- User study: Do choice rates vary even with shuffling?

Proposed Defense:

- Standardize all option introductions ("Option A:", "Option B:", no transition words)
- Log transition words in transparency log

Appendix C: Implementation Checklist

For developers implementing Tractatus-style deliberation:

Phase 1: Boundaries

- ☐ Define CRITICAL violations (hard blocks, no deliberation)

- [ ] Implement BoundaryEnforcer component with deterministic pattern matching
- [ ] Test: Verify LLM cannot bypass boundaries through persuasion

Phase 2: Stakeholder Identification

- [ ] Implement data-driven stakeholder selection (not LLM discretion)
- [ ] Load instruction-history.json, identify HIGH persistence conflicts
- [ ] Test: Verify mandatory stakeholders always appear

Phase 3: Accommodation Generation

- [ ] Implement combinatorial accommodation generator
- [ ] Ensure all stakeholder value combinations covered
- [ ] Implement Fisher-Yates shuffling algorithm
- [ ] Test: Verify value distribution balance

Phase 4: User Decision

- [ ] Disable LLM recommendations by default
- [ ] Refuse user attempts to defer decision
- [ ] Require explicit user choice + rationale
- [ ] Test: Verify LLM cannot make decision for user

Phase 5: Transparency & Bias Detection

- [ ] Log all LLM actions (facilitation log)
- [ ] Implement vocabulary balance analysis
- [ ] Implement length balance analysis
- [ ] Implement framing balance analysis
- [ ] Test: Inject biased deliberation, verify detection

Phase 6: Minority Protections (Multi-User)

- [ ] Implement minority stakeholder identification (<30% support)
- [ ] Mandate minority accommodation in option set
- [ ] Implement dissent documentation in outcome storage
- [ ] Test: Verify minority position preserved even if majority rejects

Phase 7: Auditability

- [ ] Save all deliberations to database (DeliberationSession collection)
- [ ] Generate transparency reports (JSON format)
- [ ] Implement researcher review dashboard
- [ ] Test: Verify all LLM actions are traceable

End of Document
