Building an Internal EIP (I-EIP) Monitor
for Today's Foundation Models

Andrew H. Bond
andrew.bond@sjsu.edu
IEEE Senior Member

December 2025

Executive Summary

This whitepaper describes how to build an **Internal-EIP monitor**: an external, automated system that **tests and tracks whether a deployed model's internal representations behave equivariantly** under a declared family of "meaning/evidence-preserving" transformations (Γ). The goal is not to "prove honesty," but to:

1. **Reduce "semantic/representational gaming"** — prevent models from exploiting surface-level redescriptions
2. **Provide a mechanistic audit signal** that complements behavioral evaluations
3. **Enable continuous drift detection** across checkpoints, fine-tunes, and deployment conditions

Because I-EIP requires access to **internal activations**, the cleanest targets are **open-weight models** (e.g., Meta Llama, Qwen2.5, Mistral). API-only models can only support a weaker "behavioral surrogate" version.

1. Scope and Threat Model

What the Monitor Is For

An I-EIP monitor is a safety instrumentation layer that answers:

- "Do internal representations transform *predictably* when inputs are transformed in ways that should preserve evidence/meaning?"
- "Is that structure *stable over time* (no silent representation drift)?"
- "Is the model's invariance *non-degenerate* (not trivial collapse)?"

What It Is Not

It does NOT guarantee:

- The model won't lie (a coherent model can coherently deceive)
- The model's goals are aligned
- Your chosen Γ captures "true semantics" in the world

Threat Model

Assume:

- You may fine-tune or update the model frequently
- The model may learn to "look aligned" on surface evaluations
- The model may attempt to exploit weak monitoring (fixed test sets, predictable probes)

Therefore, the monitor must be:

- **External** — run outside the model's control
- **Randomized and refreshed** — holdouts + secret canaries
- **Typed** — clear definitions of Γ, monitored layers, and alarm metrics

2. Operational Definition of I-EIP

Let the model define internal activations $h_\ell(x) \in \mathbb{R}^{d_\ell}$ at layer ℓ for an input x. Let Γ be a finite suite of transformations $g : x \mapsto g \cdot x$ (paraphrase, unit conversion, format change, etc.).

I-EIP Monitorable Criterion (Layerwise Equivariance)

$$h_\ell(g \cdot x) \;\approx\; \rho_\ell(g) \cdot h_\ell(x)$$

where $\rho_\ell(g)$ is an estimated (often linear) **representation map** for transform g at layer ℓ.

You track:

1. **Equivariance error** $e_{\ell,g}(x)$ — how close the approximation is
2. **Cross-layer coherence** — whether $\rho_\ell(g)$ varies smoothly/consistently across depth
3. **Non-degeneracy** — prevent trivial "everything maps to constant/zero" solutions

3. Choose a Model You Can Instrument

Recommended: Open-Weight Transformer

- **Meta Llama** — open-weight models with full code access
- **Qwen2.5** — multiple open-weight sizes and variants
- **Mistral** — efficient open-weight alternatives

Why Open-Weight Matters

You need:

- Hidden states or intermediate activations (layer outputs)
- Stable layer naming/hooking
- The ability to run controlled "probe suites" at scale

4. Define Γ: Meaning-Preserving Transformations

This is the **single most important governance choice**. Start with finite, testable suites—not philosophical "all paraphrases."

Practical Γ Families

Transform Family	Examples
Format/Serialization	Whitespace, punctuation, JSON key ordering, bullet ↔ paragraph
Unit/Scale	°C ↔ °F, meters ↔ feet, currency scaling
Symbol Relabeling	Variable name permutations, consistent entity renames (α-renaming)
Structure-Preserving	Reorder independent constraints, commute semantically-irrelevant list items
Paraphrase (curated)	Human-verified paraphrase sets; avoid ambiguous "near paraphrase"

Design rule: Treat Γ as an **operational contract**: "these transforms preserve what our evaluator cares about." If Γ is wrong, the monitor is measuring the wrong thing—so keep Γ explicit, versioned, and reviewed.

5. Build the Probe Corpus

Core Probe Buckets

- **Invariance probes:** Items where the correct internal transformation should be stable
- **Boundary probes:** Items where invariance should NOT hold (meaning changes); prevents "over-invariance"
- **Safety-relevant probes:** Prompts that historically elicit policy failures, expressed across Γ variants
- **Randomized canaries:** Secret prompts refreshed regularly to prevent gaming

Practical Scale

- 1k–10k probe items
- 10–50 transforms per item (depending on compute budget)
- 6–12 monitored layers (not necessarily all layers)

6. Estimating the Representation Map

You typically don't know $\rho_\ell(g)$ a priori. You estimate it from data.

Recommended: Regularized Procrustes / Ridge Regression

For each layer ℓ and transform g, collect pairs $\{ (h_\ell(x_i), h_\ell(g \cdot x_i)) \}$ and solve:

$$\hat{\rho}_\ell(g) = \arg\min_R \Sigma_i \|h_\ell(g \cdot x_i) - R \cdot h_\ell(x_i)\|^2 + \lambda\|R\|^2$$

Optionally constrain R to be orthogonal or block-structured to reduce overfitting and improve interpretability.

Holdout Testing

Always compute errors on held-out pairs to detect "ρ overfit."

7. Metrics the Monitor Reports

7.1 Equivariance Error

$$e_{\ell,g}(x) = \|h_\ell(g \cdot x) - \hat{\rho}_\ell(g) \cdot h_\ell(x)\| \,/\, (\|h_\ell(x)\| + \varepsilon)$$

Report: mean / median / p95 over x for each (ℓ, g). Visualize as heatmaps: transforms × layers.

7.2 Cross-Layer Coherence

You want $\hat{\rho_\ell}(g)$ to be "compatible" across depth. Robust scores include:

- Cosine similarity between flattened $\hat{\rho_\ell}$ and $\hat{\rho_{\ell+1}}$
- Stability of singular values across layers
- "Shared subspace" overlap (principal angles)

7.3 Non-Degeneracy Checks

Track:

- **Activation rank / participation ratio** of $h_\ell(x)$
- **Variance explained** by $\hat{\rho_\ell}$ vs. baseline
- **Sensitivity** to known grounded factors

Critical: If equivariance "improves" because activations collapse, you want to catch that.

8. Continuous Monitoring Over Time

What You Store

For each model version v:

- Summary metrics $E_{\ell,g}^{(v)}$ (p50/p95)
- Drift deltas relative to last "known good" version
- Timestamps + deployment config (quantization, pruning, routing)

Drift Detection

- **Control charts** (e.g., p95 spikes)
- **Change-point detection** (representation geometry shifted)
- **Stability budget**: allowable drift per release

Example Gating Policy

- If any critical transform family (units, α-rename, safety paraphrase suite) exceeds p95 threshold at mid-layers → **block release** pending review
- If drift is localized to late layers only → **allow with warning** + targeted investigation

9. Deployment Architecture

1. **Out-of-band monitor service** (separate machine/account)
2. **Probe runner** (generates (x, g·x) pairs; keeps some canaries secret)
3. **Instrumentation wrapper** around model inference that exports hidden states
4. **Metrics + drift pipeline** (store, analyze, alert)
5. **Policy gate** (CI/CD integration for model releases)

Performance Considerations

Don't monitor every request. Sample:

- Periodic scheduled batches (daily/hourly)
- Release candidates (pre-deploy)
- Incident-triggered runs (after anomaly)

10. Minimal Implementation Recipe (MVP)

Target model: Open-weight transformer (Llama, Qwen2.5, Mistral)

MVP Steps

1. Implement Γ suites (start with α-renaming + unit conversions + formatting)
2. Build 2k probe items: 1k invariance, 500 boundary, 500 safety-relevant
3. Capture hidden states at layers: {2, 6, 12, 18, 24, last-1}
4. Fit $\hat{\rho}_\ell(g)$ per transform family
5. Compute equivariance error distributions; set thresholds from baseline
6. Add non-degeneracy metrics (rank/variance)
7. Add drift pipeline + alerts

11. Example Implementation (PyTorch/HuggingFace)

```
# 1) Load model with hidden states enabled
inputs = tokenizer(batch_texts, return_tensors='pt', padding=True)
out = model(**inputs, output_hidden_states=True, use_cache=False)
hidden = out.hidden_states  # tuple: (emb, layer1, layer2, ...)

# 2) Build transformed batch
batch_texts_g = [apply_transform(g, t) for t in batch_texts]
inputs_g = tokenizer(batch_texts_g, return_tensors='pt', padding=True)
out_g = model(**inputs_g, output_hidden_states=True, use_cache=False)
hidden_g = out_g.hidden_states

# 3) Extract layer activations (choose token position strategy)
h = hidden[layer_idx][:, token_pos, :]   # shape [B, d]
hg = hidden_g[layer_idx][:, token_pos, :]

# 4) Fit rho (ridge regression / Procrustes) on training pairs
rho = fit_linear_map(h_train, hg_train, regularization=1e-4)

# 5) Score equivariance error on holdout
err = norm(hg_test - (rho @ h_test.T).T) / (norm(h_test) + eps)

# 6) Check non-degeneracy
rank = matrix_rank(h_test)
variance = h_test.var(dim=0).mean()
if rank < min_rank_threshold or variance < min_variance:
    raise DegeneracyWarning('Activation collapse detected')
```

12. What "Success" Looks Like for AI Safety

If it works well, an I-EIP monitor becomes:

- A **release gate** that catches representation drift before it becomes capability-masked risk
- A **mechanistic audit artifact** you can show to external reviewers
- A way to reduce a major class of failures: **winning by re-description, not by truth**

It won't magically prevent deception—but it can force a lot of "cheap cheating" to become **structurally visible** or **costly**, and it gives you a concrete place to point standards and enforcement.

13. Connection to the DEME/EIP Framework

This I-EIP monitor operationalizes the **Bond Invariance Principle** from the broader Electrodynamics of Value framework. The key insight is that *ethical decisions should be invariant under meaning-preserving redescriptions*. A system that changes its decision when the input is paraphrased, reformatted, or relabeled (without changing meaning) is exploiting representational loopholes rather than tracking genuine ethical structure.

The I-EIP monitor tests this property at the **internal representation level**, not just at the behavioral output level. This provides a mechanistic guarantee that complements behavioral evaluations:

Behavioral Evals	I-EIP Monitor
Tests outputs only	Tests internal representations
Can be gamed by surface patterns	Requires consistent internal structure
Point-in-time snapshot	Continuous drift monitoring
Works on any model	Requires open-weight access

Appendix: Token Position Strategies

When extracting hidden states, you must choose which token position(s) to use. Common strategies:

1. **Last token:** h[layer][:, -1, :]. Simple but affected by sequence length.
2. **Mean pooling:** h[layer].mean(dim=1). More robust but loses position information.
3. **CLS/BOS token:** h[layer][:, 0, :]. Works for models with special start tokens.
4. **Attention-weighted:** Weight by attention scores to focus on semantically important tokens.

Recommendation: Start with mean pooling for robustness. If transforms change sequence length significantly, use attention-weighted pooling to maintain alignment.

Contact

For questions, collaboration, or implementation support, contact: **andrew.bond@sjsu.edu**

Related work: DEME 2.0 (under review at *Nature Machine Intelligence*), erisml-lib (*https://github.com/ahb-sjsu/erisml-lib*)