A Flexible Mathematical Framework for Model Comparison: U-TIM (version 1.2)

João Lucas Meira Costa

Thursday, February 6, 2025 at 3:32 PM -03 (UTC)

The author acknowledges contributions from ChatGPT, DeepSeek, and Gemini for assistance with equations, code, and documentation.

Abstract

The Universal Theory Incoherence Measure (U-TIM) is a mathematical framework for comparing and evaluating models across diverse domains by quantifying the inconsistencies between their predictions. This document introduces the U-TIM framework, details its mathematical formulation and Python implementation, and discusses its potential applications, particularly in physics for evaluating Theories of Everything (TOEs). The flexible design of U-TIM allows for adaptation to various domains, including physics, biology, and economics, with the goal of providing a universal tool for model assessment and comparison. Detailed domain-specific applications will be provided in companion documents.

1 Introduction

The Universal Theory Incoherence Measure (U-TIM) is a flexible **mathematical** framework designed to identify and quantify inconsistencies between different models in various domains. It provides a general mathematical structure that can be adapted to a wide range of models, focusing on the **incoherence** between their predictions. This document explains U-TIM's foundation, implementation, and potential applications in physics (particularly for evaluating Theory of Everything candidates). Domain-specific guides will be published separately.

2 Mathematical Formulation

U-TIM compares model outputs across a shared input space using these components:

- $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$: Set of n models

- M_r: Reference model (domain-specific baseline)
- X: Input space (common parameters across models)
- Y: Output space (comparable predictions/measurements)
- $f_i : X \to Y$: Model M_i's output function
- $d_Y(y_1, y_2)$: Distance metric on output space Y
- $w : X \to \mathbb{R}^+$: Weight function over input space

2.1 U-TIM Score

The U-TIM score for model M_i relative to M_r:

$$\text{U-TIM}(M_i) = \int_X w(x) \cdot d_Y(f_i(x), f_r(x))\, dx \tag{1}$$

Lower scores indicate greater similarity to the reference model, though this does not inherently imply superiority if the reference has known limitations.

2.2 Pairwise Coherence

The pairwise coherence between models M_i and M_j:

$$C(M_i, M_j) = \int_X w(x) \cdot d_Y(f_i(x), f_j(x))\, dx \tag{2}$$

3 Implementation (Python)

The Python implementation uses discrete sampling for numerical integration:

```
import numpy as np
from scipy.spatial import distance

class UniversalTIM:
    def __init__(self, models, reference_fn,
                 X_samples,
                 distance_metric='euclidean',
                 weight_fn=lambda x: 1):
        """
        U-TIM implementation for cross-domain model comparison

        Parameters:
        - models: dict {name: callable(x: array(n_dims,)) -> y}
        - reference_fn: callable(x: array(n_dims,)) -> y
        - X_samples: array(n_samples, n_dims) parameter samples
        - distance_metric: str/callable for scipy.spatial.distance.cdist
```

```python
        - weight_fn: callable(x: array(n_dims,)) -> scalar weight
        """
        self.models = models
        self.ref_fn = reference_fn
        self.X = X_samples
        self.dist_metric = distance_metric
        self.weight_fn = np.vectorize(weight_fn, signature='(n)->()')

    def _compute_distances(self, fn1, fn2):
        """Compute distances between model outputs"""
        Y1 = np.array([fn1(x) for x in self.X])
        Y2 = np.array([fn2(x) for x in self.X])
        return distance.cdist(Y1, Y2, self.dist_metric).diagonal()

    def calculate_utim(self):
        """Calculate U-TIM scores for all models"""
        scores = {}
        w = self.weight_fn(self.X)

        for name, model in self.models.items():
            dists = self._compute_distances(model, self.ref_fn)
            scores[name] = np.sum(w * dists)

        return scores

    def pairwise_coherence_matrix(self):
        """Compute full model coherence matrix"""
        model_names = list(self.models.keys())
        n_models = len(model_names)
        matrix = np.zeros((n_models, n_models))
        w = self.weight_fn(self.X)

        # Consider vectorization for large model sets
        for i in range(n_models):
            for j in range(i, n_models):
                dists = self._compute_distances(self.models[model_names[i]],
                                                self.models[model_names[j]])
                matrix[i,j] = matrix[j,i] = np.sum(w * dists)

        return matrix, model_names
```

4 Applying U-TIM to Physics (TOE Evaluation)

Application to physics requires careful implementation:

1. **TOE Candidates**: Select theories with common mathematical ground-

ing 2. **Reference Model**: Typically Standard Model + General Relativity 3. **Input Space (X)**: Common fundamental parameters (may require normalization) 4. **Output Space (Y)**: Observable predictions (e.g., scattering amplitudes) 5. **Distance Metric**: Physically meaningful comparison (e.g., weighted L2) 6. **Weight Function**: Emphasize experimentally accessible regions 7. **Sampling**: For high-dimensional spaces, use Monte Carlo integration 8. **Calculation**: Compute U-TIM scores using domain-appropriate implementation 9. **Interpretation**: Consider reference model limitations and measurement uncertainties

5 Conclusion

U-TIM provides a consistent mathematical framework for model comparison across domains. While particularly promising for TOE evaluation, successful application requires careful domain-specific implementation. Future work will address computational challenges in high-dimensional spaces and develop standardized metrics for different fields.

License and Attribution