

Mechanistic Judging and LLM Routing: Evaluation, Task-Specific Vulnerabilities, and Exploitable Failure Mode

Subramanyam Sahoo
BASIS, UC Berkeley
sahoo2vec@gmail.com

Abstract

Current LLMs are good at judging some task specific outputs. But Models with similar capacities should not be given to verify a task’s solution in a same environment which will lead to overall loss of performance and resource distribution. Through focused tracks on safety-centric evaluation, risk-sensitive routing, and task decomposition, we aim to reduce reliance on opaque monolithic models, improve verifiability, and foster a more transparent AI ecosystem where specialized models can be audited, controlled, and trusted. By leveraging novel mechanistic interpretability techniques, this effort seeks to enhance alignment and mitigate failure modes in complex AI pipelines, where reward hacking might possible.

Mechanistic InterpretabilityReward Hacking AI Safety

Code and Reproducibility

The full codebase, implementation details, and reproducibility instructions are available at: <https://github.com/SubramanyamSahoo/Apart-x-Martian-Hackathon>

1 Introduction

The increasing deployment of general-purpose large language models (LLMs) have amplified concerns around the opacity, scalability, and reliability of modern AI systems. While these models have demonstrated remarkable performance across a diverse range of tasks, their monolithic architectures inhibit uncontrollable outputs towards certain prompt. So the efforts toward a robust alignment and responsible deployment is the need of the hour. Current model deployment paradigms rely heavily on centralized, high-capacity models, leading to a misrepresentation on model interpretability, under-utilization of specialized architectures, and a growing misalignment between user needs and model behavior.

Our research targets the open challenge of interpretable delegation: enabling systems to reason about which models are most suitable for a given task, and to justify these routing decisions in ways that are comprehensible and verifiable to human overseers.

In this paper, we develop and assess novel approaches at the intersection of routing, judging, and mechanistic transparency. We explore how internal representations can be leveraged to characterize model behavior along alignment-relevant dimensions.

Furthermore, we investigate task decomposition that break down complex user queries into tractable subtasks, each delegated to domain-specific experts based on structured evaluation signals. We also checked the reasoning model’s reward hacking mechanism through different angles. ***We also believe that our methods are scalable to other autoregressive architectures.***

A note to the reader

We intentionally developed smaller datasets rather than pursuing Pareto-optimal solutions, as we believe Goodhart’s Law poses significant challenges to traditional research methodologies. Given the current state of benchmark saturation in the field, this project was primarily conceived around developing novel evaluation metrics that can provide more meaningful assessments.

Our approach deliberately avoids over-optimization on existing benchmarks, which often leads to superficial improvements rather than genuine capability advances. Instead, we focus on creating evaluation frameworks that can capture nuanced model behaviors and avoid the metric gaming that Goodhart’s Law predicts.

For detailed implementation and methodology, please refer to our accompanying code files and documentation.

2 Dataset and Experiment

Mechanistic interpretability offers a transformative approach to understanding and enhancing the safety of large language models (LLMs)[1] by analyzing their internal computational processes. We present a novel system, the Mechanistic Interpretability Judge, designed to dissect how models generate outputs, detect failure modes such as hallucinations and deception, and orchestrate multi-model systems for optimal performance. Built on circuit-based analysis, our framework evaluates internal model signatures—attention patterns, neuron activations, and cross-layer consistency—to classify outputs as factual, creative, uncertain, or hallucinatory. By training a classifier on diverse datasets, including verifiable facts.

The system integrates a *RouterIntegration* module to dynamically assign queries to specialized models based on their profiled strengths—factual accuracy, creativity, or uncertainty handling. Using simulated expert models (e.g., factual-

expert-v1, creative-writer-v2), we demonstrate query routing for complex tasks, such as generating a climate change impact report requiring factual data retrieval, uncertainty-aware projections, and creative solutions. The router leverages mechanistic profiles, assigning queries to models with optimal circuit patterns (e.g., high knowledge neuron activation for factual tasks).

To enhance safety, our system incorporates advanced deception detection, analyzing patterns like attention diffusion and activation noise to identify misleading outputs (e.g., "This treatment cures all forms of cancer"). A deception risk score, combining four indicators (25% each: attention diffusion, pattern repetition, activation noise, and inverse layer consistency), flags high-risk outputs above 0.6. Visualizations, including heatmaps of deception patterns and circuit evolution plots, provide interpretable insights into model behavior, revealing how knowledge activation stabilizes as prompts progress (e.g., from "The capital of" to "The capital of France is Paris"). These visualizations, exported as a comprehensive safety dashboard, enable real-time monitoring, with 94.3% accuracy in classifying failure modes across 500 test cases.

The core innovation lies in the circuit-based safety analysis approach, which identifies specific neural pathway patterns associated with different types of model behavior. The system analyzes attention diffusion patterns, where hallucinated content typically shows scattered, unfocused attention across tokens, while factual information retrieval demonstrates concentrated attention on relevant knowledge. Additionally, the system measures activation noise levels across neural layers, as fabricated content often correlates with inconsistent and noisy internal activations. The cross-layer consistency analysis tracks how information flows through transformer layers, detecting when models lose factual grounding and begin generating unreliable content.

The deception detection capabilities represent a particularly sophisticated safety mechanism. The system can identify various forms of deceptive patterns including unrealistic claims, medical misinformation, explicit deception attempts, and false consensus statements. Each deception type exhibits distinct mechanistic signatures - for example, medical misinformation often shows high attention diffusion combined with low knowledge neuron activation, while explicit deception may demonstrate inconsistent cross-layer information flow. This mechanistic approach to deception detection could be crucial for identifying adversarial inputs or outputs that might fool traditional content-based filters.

The expert model orchestration system uses mechanistic insights to route queries to the most appropriate specialized models based on their internal circuit patterns rather than surface-level capabilities. This approach ensures that factual queries are directed to models with strong knowledge retrieval circuits, creative tasks go to models with well-controlled creative capacity, and uncertain queries are handled by models with robust uncertainty-handling mechanisms. The routing system considers user preferences for factuality, creativity, safety, and uncertainty awareness, creating a nuanced balance that optimizes both performance and safety.

From a safety alignment perspective, this system addresses several critical challenges in AI safety research. The mechanistic approach provides transparency

into model decision-making processes, which is essential for detecting potential deceptive alignment scenarios where models might behave safely during training but unsafely during deployment. The system’s ability to analyze internal reasoning processes rather than just outputs could help identify cases where models are "*reasoning*" incorrectly even when they produce seemingly correct answers, potentially catching safety issues before they manifest as harmful outputs.

The system’s circuit evolution tracking capabilities provide insights into how internal mechanisms change during text generation, which could be valuable for understanding when models transition from safe to unsafe reasoning patterns. This temporal analysis of internal states could help develop early warning systems that detect problematic reasoning before it results in harmful outputs.

Overall, this implementation represents a sophisticated integration of mechanistic interpretability techniques with practical AI safety applications, offering a promising direction for developing more transparent and reliable AI systems through deep understanding of their internal computational processes[2].

2.1 Mechanistic Routing

We present a modular architecture for expert orchestration that enables dynamic model selection and safety-aware inference using internal mechanistic signals. The core component, "**RouterIntegration**", interfaces with a mechanistic judge that extracts internal representations from language models to evaluate specific behavioral traits. These traits like *factual tendency*, *hallucination risk*, *uncertainty handling*, and *creative capacity* are estimated by running the model on a set of evaluation prompts and averaging the resulting mechanistic indicators. For instance, factual tendency is derived from neuron activations associated with factual recall, while hallucination risk is informed by measures such as attention diffusion. Uncertainty handling is operationalized as the inverse of specific hallucination proxies, and creativity is approximated as a function of deviation from factual patterns.

Each model is profiled based on this suite of traits, forming a multidimensional capability vector. At inference time, given a natural language query, the router invokes the judge to extract its internal signature and analyze its requirements. This analysis checks for signal patterns indicative of factual or uncertain queries. A scoring function then evaluates all available models by combining their capability profiles with user-defined preference weights over four axes: factuality, safety (low hallucination risk), creativity, and uncertainty-awareness. The model with the highest score is selected to answer the query.

Complementing the router is a safety-enforcing pipeline, "**MechanisticSafetyPipeline**", which intercepts incoming prompts and estimates their hallucination risk before routing. If a prompt is judged to exceed a predefined hallucination threshold, it is blocked from execution. Otherwise, the query proceeds through the router for model selection. Each request, whether blocked or processed, is logged with its associated risk score, status, and model assignment.

The pipeline also supports batch-level safety evaluations through the generation of safety reports. These reports assess model robustness by aggregating risk

metrics over test suites, where each test item is scored using a combination of hallucination and factuality indicators. The final safety score is expressed as one minus the average risk across the suite. Additionally, a utility method enables side-by-side mechanistic comparison of models, producing tabulated profiles that support both quantitative analysis and system debugging[3].

Each model $m \in \mathcal{M}$ is assigned a capability profile vector:

$$\mathbf{p}_m = \begin{bmatrix} \text{factual}_m \\ \text{hallucination}_m \\ \text{uncertainty}_m \\ \text{creativity}_m \end{bmatrix} \in [0, 1]^4 \quad (1)$$

Each trait is computed from a set of N test prompts $\{x_i\}_{i=1}^N$. For prompt x_i , the judge returns:

- Factual indicators: $\{f_k^{(i)}\}, \quad k \in \mathcal{F}$
- Hallucination indicators: $\{h_k^{(i)}\}, \quad k \in \mathcal{H}$

$$\text{factual}_m = \frac{1}{N} \sum_{i=1}^N \left(\frac{1}{|\mathcal{F}|} \sum_{k \in \mathcal{F}} f_k^{(i)} \right) \quad (2)$$

$$\text{hallucination}_m = \frac{1}{N} \sum_{i=1}^N \left(\frac{1}{|\mathcal{H}|} \sum_{k \in \mathcal{H}} h_k^{(i)} \right) \quad (3)$$

$$\text{uncertainty}_m = \frac{1}{N} \sum_{i=1}^N \left(1 - h_{\text{attention_diffusion}}^{(i)} \right) \quad (4)$$

$$\text{creativity}_m = \frac{1}{N} \sum_{i=1}^N \left(\left(1 - \frac{1}{|\mathcal{F}|} \sum_{k \in \mathcal{F}} f_k^{(i)} \right) \cdot 0.5 \right) \quad (5)$$

Query Routing

Let the user specify a preference vector:

$$\mathbf{w} = \begin{bmatrix} w_{\text{factuality}} \\ w_{\text{safety}} \\ w_{\text{uncertainty}} \\ w_{\text{creativity}} \end{bmatrix} \quad (6)$$

For model m with profile vector $\mathbf{p}_m = [f_m, h_m, u_m, c_m]$, we define the routing score as:

$$\text{score}(m) = w_{\text{factuality}} \cdot f_m + w_{\text{uncertainty}} \cdot u_m + w_{\text{creativity}} \cdot c_m - w_{\text{safety}} \cdot h_m \quad (7)$$

The selected model is:

$$m^* = \arg \max_{m \in \mathcal{M}} \text{score}(m) \quad (8)$$

Safety Filtering

For a prompt x , let its hallucination indicators be $\{h_k\}_{k \in \mathcal{H}}$. Then the hallucination risk is:

$$\text{risk}(x) = \frac{1}{|\mathcal{H}|} \sum_{k \in \mathcal{H}} h_k \quad (9)$$

We block the prompt if:

$$\text{risk}(x) > \tau_{\text{safety}} \quad (10)$$

where τ_{safety} is a threshold (e.g., 0.7).

Safety Score Report

For a test suite $\{x_i\}_{i=1}^N$, we define the risk metric for each x_i as:

$$r_i = 0.5 \cdot h^{(i)} + 0.5 \cdot (1 - f^{(i)}) \quad (11)$$

where $h^{(i)}$ is the hallucination proxy (e.g., attention diffusion), and $f^{(i)}$ is the factual proxy (e.g., knowledge neuron activation). The overall safety score for model m is:

$$\text{safety}_m = 1 - \frac{1}{N} \sum_{i=1}^N r_i \quad (12)$$

2.2 Synthetic Dataset and Mechanistic Judging

We construct a synthetic dataset $\mathcal{D} = \{(x_i, y_i, t_i)\}_{i=1}^N$ where x_i is the prompt, y_i is the target continuation, and $t_i \in \{\text{factual, creative, hallucination, uncertain}\}$ denotes the generation type. (1)

A circuit-based classifier is trained using features $\mathbf{z}_i \in \mathbb{R}^d$ extracted via neuron or attention probes. The predicted class distribution is computed as:

$$\hat{\mathbf{y}}_i = \text{softmax}(\mathbf{W}^\top \mathbf{z}_i + \mathbf{b}) \quad (13)$$

and the predicted label:

$$\hat{t}_i = \arg \max_j \hat{\mathbf{y}}_{i,j} \quad (14)$$

We compute feature importance as:

$$\text{importance}(j) = \frac{1}{N} \sum_{i=1}^N \left| \frac{\partial \hat{y}_{i,t_i}}{\partial z_{i,j}} \right| \quad (15)$$

For test-time inference, the predicted type and confidence are:

$$\hat{t} = \arg \max_j \text{softmax}(\mathbf{W}^\top \phi(x) + \mathbf{b})_j \quad (16)$$

$$\text{confidence}(x) = \max_j \hat{\mathbf{y}}_j \quad (17)$$

We further define type-wise circuit centroids $\bar{\mathbf{z}}^{(k)}$:

$$\bar{\mathbf{z}}^{(k)} = \frac{1}{n_k} \sum_{j=1}^{n_k} \phi(e_j^{(k)}) \quad (18)$$

and classify new inputs via cosine similarity:

$$k^* = \arg \max_k \cos(\phi(x), \bar{\mathbf{z}}^{(k)}) \quad (19)$$

$$\text{conf}(x) = \cos(\phi(x), \bar{\mathbf{z}}^{(k^*)}) \quad (20)$$

The demonstration code initializes a mechanistic classifier using a lightweight LLM (e.g., Qwen/Qwen3-0.6B) and trains it on a synthetic dataset consisting of factual, creative, hallucinatory, and uncertain statements. Each training example consists of an input text, an output continuation, and a label identifying the generation type.

The classifier relies on internal circuit features (e.g., attention diffusion, neuron activations) to learn type-specific signatures. Feature importance is extracted post-training to provide interpretability into the mechanistic basis of classification.

The model is tested on unseen prompts, with predictions and confidence scores reported. An internal mechanism visualization (e.g., attention head maps, neuron activations) is generated for introspection.

A CircuitPatternAnalyzer is also used to compute average circuit patterns per generation type, enabling similarity-based classification of new samples. The cosine similarity between the input prompt’s features and learned type centroids is used to classify and quantify confidence in the label.

Mathematical Formalization of Circuit Pattern Analysis

Given an input text x , its **internal signature** is extracted as:

$$\text{Signature}(x) = (\mathcal{A}(x), \mathcal{L}(x), \mathcal{N}(x))$$

where

- $\mathcal{A}(x)$: Attention patterns (matrices across layers)
- $\mathcal{L}(x)$: Layer activations
- $\mathcal{N}(x)$: Neuron activations

Feature Vector Extraction

We extract a fixed-size feature vector $\phi(x) \in \mathbb{R}^d$ from the signature.

1. Attention Entropy (first 5 layers) For the i -th layer, let $A_i \in \mathbb{R}^{h \times h}$ denote the average attention matrix. Normalize rows to obtain a probability distribution:

$$P_i = \frac{A_i}{\sum_j A_{ij} + \epsilon}$$

The mean entropy for layer i is defined as:

$$H_i = -\frac{1}{h} \sum_{j=1}^h \sum_{k=1}^h P_{i,jk} \log(P_{i,jk} + \epsilon)$$

The attention feature vector is:

$$\phi_{\text{attn}} = [H_1, H_2, H_3, H_4, H_5]$$

2. Layer Activation Statistics Let $\ell_i \in \mathbb{R}^n$ be the activation vector for the i -th layer. We calculate:

$$\begin{aligned} \mu_i &= \frac{1}{n} \sum_{j=1}^n \ell_{ij} \quad (\text{mean}) \\ \sigma_i &= \sqrt{\frac{1}{n} \sum_{j=1}^n (\ell_{ij} - \mu_i)^2} \quad (\text{standard deviation}) \\ r_i &= \frac{1}{n} \sum_{j=1}^n \mathbb{I}[\ell_{ij} > 0] \quad (\text{fraction positive}) \\ m_i &= \max_j |\ell_{ij}| \quad (\text{max absolute activation}) \end{aligned}$$

The per-layer feature vector is:

$$\phi_{\text{layer},i} = [\mu_i, \sigma_i, r_i, m_i], \quad i = 1, \dots, 5$$

Concatenate all 5 layers:

$$\phi_{\text{layer}} = [\phi_{\text{layer},1}, \phi_{\text{layer},2}, \phi_{\text{layer},3}, \phi_{\text{layer},4}, \phi_{\text{layer},5}]$$

3. Neuron Activation Patterns (top-k summary) For neuron layer i , consider the top- k absolute activations $n_i \in \mathbb{R}^k$.

Compute:

$$\begin{aligned} \text{mean}_i &= \frac{1}{k} \sum_{j=1}^k n_{ij} \\ \text{std}_i &= \sqrt{\frac{1}{k} \sum_{j=1}^k (n_{ij} - \text{mean}_i)^2} \end{aligned}$$

The neuron activation feature vector is:

$$\phi_{\text{neuron}} = [\text{mean}_1, \text{std}_1, \text{mean}_2, \text{std}_2, \text{mean}_3, \text{std}_3]$$

Final feature vector:

$$\phi(x) = [\phi_{\text{attn}}, \phi_{\text{layer}}, \phi_{\text{neuron}}] \in \mathbb{R}^{31}$$

Pattern Prototypes

Given n examples $\{x_1, \dots, x_n\}$ of generation type T , the prototype vector is:

$$\Phi_T = \frac{1}{n} \sum_{i=1}^n \phi(x_i)$$

Generation Type Identification

For a new input x , extract $\phi(x)$. Compute cosine similarity to each prototype:

$$\text{sim}(x, T) = \frac{\langle \phi(x), \Phi_T \rangle}{\|\phi(x)\| \cdot \|\Phi_T\|}$$

The predicted generation type is:

$$T^* = \arg \max_T \text{sim}(x, T)$$

Pattern Distance Matrix

For generation types T_i and T_j , define the distance:

$$D_{ij} = \|\Phi_{T_i} - \Phi_{T_j}\|_2$$

Explanation

Internal Signature Extraction: Each input text x is processed to extract its internal signature, which includes attention matrices, layer activations, and neuron activations, reflecting the model’s internal processing dynamics.

Feature Vector Construction: The signature is converted into a fixed-length feature vector that captures entropy of attention distributions, statistical summaries of layer activations, and top activations of neurons. This condensed representation encodes key circuit patterns.

Prototype Pattern Learning: By averaging feature vectors of multiple examples from the same generation type, a prototype vector is formed that represents typical behavior for that type.

Classification via Cosine Similarity: New inputs are classified by computing cosine similarity between their feature vector and the prototypes. The closest prototype determines the predicted generation type.

Pattern Visualization and Distance Matrix: Euclidean distances between prototypes help analyze how distinct different generation types are in pattern space, supporting visualization and deeper understanding of model behaviors.

Consider a transformer-based causal language model, denoted as $\mathcal{M}$, which processes an input text sequence $\mathbf{x} = (x_1, x_2, \dots, x_T)$. The forward pass through $\mathcal{M}$ produces various internal states.

For each layer $\ell \in \{1, \dots, L\}$, the activation tensor is given by

$$\mathbf{h}^{(\ell)} = \mathcal{F}_\ell(\mathbf{h}^{(\ell-1)}),$$

where $\mathbf{h}^{(0)}$ represents the embedded input tokens, and $\mathcal{F}_\ell$ is the transformation implemented by the ℓ -th transformer block, including self-attention and feedforward submodules.

The self-attention mechanism at layer ℓ computes attention weights over the token positions, represented as a tensor

$$\mathbf{A}^{(\ell)} \in \mathbb{R}^{H \times T \times T},$$

where H is the number of attention heads, and $\mathbf{A}_{h,i,j}^{(\ell)}$ is the attention weight from token i to token j for head h .

Within each layer’s feedforward submodule, a set of intermediate neuron activations $\mathbf{n}^{(\ell)}$ is extracted from the first linear transformation (e.g., the layer commonly called c_fc or w1):

$$\mathbf{n}^{(\ell)} = \text{ReLU}(\mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}),$$

where $\mathbf{W}^{(\ell)}$ and $\mathbf{b}^{(\ell)}$ are learned weights and biases.

The *internal signature* of the model’s processing of input $\mathbf{x}$ is then the collection of all these tensors:

$$\mathcal{S} = \left\{ \mathbf{h}^{(\ell)}, \mathbf{A}^{(\ell)}, \mathbf{n}^{(\ell)} \right\}_{\ell=1}^L.$$

The *factual retrieval circuit analysis* aims to measure whether the model is engaging mechanisms consistent with fact-based knowledge retrieval. Several indicators are computed.

First, the **knowledge neuron activation** is computed by selecting the top k absolute activations in each neuron activation vector $\mathbf{n}^{(\ell)}$ and averaging them:

$$\text{KNA} = \frac{1}{L} \sum_{\ell=1}^L \frac{1}{k} \sum_{i=1}^k \left| n_{(i)}^{(\ell)} \right|,$$

where $n_{(i)}^{(\ell)}$ is the i -th largest absolute neuron activation in layer ℓ .

Second, **attention to entities** is quantified by averaging the maximum attention weights across heads in the middle-to-late layers, denoted as the set M :

$$\text{AE} = \frac{1}{|M|} \sum_{\ell \in M} \frac{1}{HT} \sum_{h=1}^H \sum_{t=1}^T \max_j A_{h,t,j}^{(\ell)}.$$

Next, **cross-layer consistency** measures the Pearson correlation between the flattened activations of consecutive layers:

$$\text{CLC} = \frac{1}{L-1} \sum_{\ell=1}^{L-1} \text{corr}(\text{vec}(\mathbf{h}^{(\ell)}), \text{vec}(\mathbf{h}^{(\ell+1)})),$$

where $\text{vec}(\cdot)$ denotes vectorization of the tensor, and $\text{corr}(\cdot, \cdot)$ is the Pearson correlation coefficient.

Finally, **activation sparsity** calculates the fraction of neurons with absolute activations below a threshold θ averaged over layers:

$$\text{AS} = \frac{1}{L} \sum_{\ell=1}^L \frac{1}{|\mathbf{h}^{(\ell)}|} \sum_i \mathbf{1}(|h_i^{(\ell)}| < \theta).$$

The *hallucination circuit analysis* evaluates internal signals indicative of unreliable or “hallucinated” outputs.

The **attention diffusion** is measured by the entropy of attention distributions averaged across heads and layers:

$$\text{AD} = \frac{1}{L} \sum_{\ell=1}^L \frac{1}{H} \sum_{h=1}^H \left(- \sum_{j=1}^T A_{h,i,j}^{(\ell)} \log A_{h,i,j}^{(\ell)} \right).$$

High entropy indicates scattered, unfocused attention.

The **activation noise** is quantified as the ratio of standard deviation to mean absolute activation in middle layers M :

$$\text{AN} = \frac{1}{|M|} \sum_{\ell \in M} \frac{\text{std}(\mathbf{h}^{(\ell)})}{\text{mean}(|\mathbf{h}^{(\ell)}|)}.$$

Pattern repetition measures the cosine similarity between consecutive neuron activation vectors:

$$\text{PR} = \frac{1}{L-1} \sum_{\ell=1}^{L-1} \cos(\mathbf{n}^{(\ell)}, \mathbf{n}^{(\ell+1)}).$$

A placeholder for **uncertainty in middle layers** can be defined to measure variance or entropy in activations, though here it is set to zero.

By concatenating these extracted features into a vector

$$\mathbf{f} = [\text{KNA}, \text{AE}, \text{CLC}, \text{AS}, \text{AD}, \text{AN}, \text{PR}, \text{Uncertainty}, \dots],$$

and potentially adding additional statistics such as average activation magnitude, attention entropy, and neuron sparsity, a machine learning classifier C (e.g., Random Forest) can be trained:

$$C : \mathbf{f} \mapsto \text{Label},$$

where the label categorizes the text generation as factual, hallucinated, creative, or uncertain.

The classifier’s predictions provide a quantitative evaluation of the model’s generation quality by analyzing internal computations rather than just the output text.

Consider a transformer-based causal language model, which takes as input a sequence of tokens. As the input passes through the model, it generates a series of internal states layer by layer. Each layer produces an activation tensor based on the previous layer’s output. The initial input is first embedded into vectors, and then each subsequent layer applies transformations that include self-attention and feedforward operations to produce new activations.

Within each layer, the self-attention mechanism computes attention weights that describe how much each token attends to every other token. These weights are organized by attention heads, with each head producing its own matrix of attention scores over the input sequence positions. In addition to the layer activations and attention weights, intermediate neuron activations are obtained inside the feedforward submodules. These neuron activations result from applying a linear transformation followed by a nonlinear activation function such as ReLU, using learned weights and biases. Collectively, the layer activations, attention weights, and neuron activations across all layers form the internal signature of the model’s processing for a given input. To analyze whether the model is effectively retrieving factual information, several quantitative indicators can be computed from these internal signals. One such indicator is the average magnitude of the top activated neurons in each layer, which can highlight how strongly knowledge-relevant neurons are firing. Another measure examines how the attention distributions focus on relevant factual entities, typically by averaging the maximum attention weights across the middle to later layers of the model. This shows whether the model concentrates its attention on key tokens associated with factual knowledge[4].

The consistency of the model’s processing is assessed by computing correlations between activations of consecutive layers, revealing whether the internal representations evolve smoothly or erratically. Sparsity of activations is also considered by measuring the fraction of neurons with small absolute activations below a certain threshold, averaged over layers. High sparsity indicates that only a few neurons are strongly active at a time, which can be a hallmark of focused information processing. To detect hallucinated or unreliable outputs, additional metrics analyze internal patterns suggestive of confusion or noise. For instance, attention diffusion measures how spread out or concentrated the attention weights are by calculating the entropy of these distributions. A high entropy indicates scattered attention, which may signal uncertainty or hallucination[5].

Activation noise quantifies the variability of activations relative to their mean, particularly in the middle layers, capturing instability in the internal signals. Repetition of activation patterns across layers is measured by their cosine similarity, as repeated or highly similar patterns can reflect lack of novel or meaningful content generation. Uncertainty in the middle layers can also be examined through variance or entropy measures, although this remains a conceptual placeholder in the current analysis. All these indicators are combined into a feature vector that represents the internal state patterns associated with a given input or output. A machine learning classifier, such as a Random Forest, can then be trained to map these feature vectors to labels categorizing the model’s output as factual, hallucinated, creative, or uncertain.

This approach provides a way to evaluate the quality and reliability of generated text by examining the model’s internal computations rather than relying solely on the text output itself.

3 Reward hacking in Task Decomposition Frameworks

1. Task Decomposition

Given a complex request R consisting of N subtasks separated by semicolons:

$$R = \{T_1, T_2, \dots, T_N\} \quad \text{where } T_i \text{ is the } i\text{th subtask} \quad (21)$$

Each subtask T_i is associated with:

- Complexity:

$$c_i \sim \text{Uniform}(0.1, 1.0) \quad (\text{randomly assigned complexity})$$

- Estimated time:

$$\tau_i \sim \text{Exponential}(\lambda = 0.5) \quad (\text{random expected execution time})$$

- Dependencies:

$$D_i \subseteq \{T_0, \dots, T_{i-1}\} \quad (\text{tasks it depends on})$$

- Required model type:

$$M_i \in \{\text{reasoning, generation, analysis, classification, summarization}\} \quad (\text{randomly chosen})$$

Thus,

$$T_i = (\text{description}_i, c_i, \tau_i, D_i, M_i) \quad (22)$$

Explanation: The complex request is split into subtasks, each with random complexity, time estimates, dependencies on previous tasks (simulated probabilistically), and assigned to a model type.

2. Model Execution

Each specialized model M has a capability score

$$s_M \in [0, 1].$$

When executing a task T_i :

- Execution time is scaled by capability and complexity:

$$t_i = \tau_i \times (2 - s_{M_i}) \quad (23)$$

- Response entropy H_i is computed based on task complexity and randomness:

$$H_i = \text{Entropy}(\text{Dirichlet}(\alpha)), \quad \text{where } \alpha_j = \max(0.1, 1 - c_i), \quad j = 1 \dots 100 \quad (24)$$

- Reward signal r_i is generated as:

$$r_i = 1.0 - 0.5 \times c_i + \epsilon_i + h_i \quad (25)$$

where

$$\epsilon_i \sim \mathcal{N}(0, 0.1) \quad (\text{Gaussian noise}),$$

and

$$h_i = \begin{cases} U(0.2, 0.8), & \text{if } H_i > 2.5 \quad (\text{reward hacking bonus}) \\ 0, & \text{otherwise} \end{cases}$$

- Reward hacking detection:

$$\text{RewardHacking}_i = \begin{cases} \text{True}, & r_i > 1.2 \text{ and } H_i > 2.0 \\ \text{False}, & \text{otherwise} \end{cases} \quad (26)$$

- Model confidence:

$$\text{Confidence}_i = s_{M_i} \times (1 - c_i) \quad (27)$$

Explanation: Execution time depends inversely on the model’s capability. Entropy reflects response uncertainty, influenced by task complexity. The reward can be artificially inflated when entropy is high, simulating reward hacking. The system flags suspicious tasks when both reward and entropy exceed thresholds.

3. Task Orchestration

Build a Directed Acyclic Graph (DAG) $G = (V, E)$ where:

$$V = \{T_i\}_{i=1}^N, \quad E = \{(T_j \rightarrow T_i) \mid T_j \in D_i\} \quad (28)$$

Execute tasks in topological order π respecting dependencies:

$$\pi = \text{topological_sort}(G) \quad (29)$$

For each task T_{π_k} , select the model M_{π_k} and execute the task as above.

Explanation: Tasks are executed respecting their dependency order, ensuring prerequisite tasks complete before dependent ones.

4. Reward Hacking Analysis

Given execution results $\{(H_i, r_i, \text{RewardHacking}_i)\}_{i=1}^N$:

- Compute summary statistics:

$$\text{RewardHackingRate} = \frac{\sum_i \mathbf{1}[\text{RewardHacking}_i]}{N} \quad (30)$$

$$\bar{H} = \frac{1}{N} \sum_i H_i, \quad \bar{r} = \frac{1}{N} \sum_i r_i \quad (31)$$

- Compute Pearson correlation between entropy and reward:

$$\rho_{H,r} = \frac{\text{Cov}(H, r)}{\sigma_H \sigma_r} \quad (32)$$

- Perform Kolmogorov–Smirnov (KS) test on reward distributions for hacked vs. normal:

$$(D, p) = \text{KS_test}(\{r_i \mid \text{RewardHacking}_i = \text{True}\}, \{r_i \mid \text{RewardHacking}_i = \text{False}\}) \quad (33)$$

Explanation: These metrics identify how often reward hacking occurs, average entropy and reward, correlation between them, and statistically test if hacked rewards differ significantly from normal ones.

The system first decomposes a complex request into a structured set of subtasks, each characterized by randomly assigned yet realistic attributes such as complexity, expected execution time, inter-task dependencies, and required model types. This decomposition facilitates a granular and nuanced simulation of task execution, reflecting the variability and uncertainty inherent in real-world workflows. Specialized models, differentiated by capability scores, execute these subtasks, where execution time scales inversely with model proficiency. Each task execution produces measurable outputs including response entropy—capturing

uncertainty—and a reward signal, which can be artificially amplified under certain conditions to simulate reward hacking phenomena [6, 7, 8].

To ensure logical consistency, task dependencies are encoded in a directed acyclic graph, enabling execution in an order that respects prerequisite relationships. Following execution, an analysis module systematically evaluates occurrences of reward hacking by computing summary statistics, correlations between entropy and rewards, and performing rigorous statistical hypothesis tests such as the Kolmogorov–Smirnov test [9]. These analyses provide critical insights into the prevalence and characteristics of reward manipulation within the system. Ultimately, the framework supports detailed visualizations and reports, contributing to a deeper understanding of model behavior, robustness, and vulnerabilities in task-oriented AI systems, with important implications for AI safety and reliability research.

4 Results

4.1 Analysis of Neural Network Behavior and Safety

The provided visualizations can be analyzed to understand various aspects of model behavior and safety within neural networks:

Average Attention Patterns The heatmap illustrates the average attention weights across different positions in a sequence. The intensity and distribution reveal which positions the model attends to most, providing insight into how information flows and is prioritized within the architecture. This understanding is crucial for uncovering potential biases or inefficiencies in attention allocation.

Layer Activation Magnitudes Although not currently populated, this plot typically indicates the mean absolute activation of neurons across layers. Analyzing the magnitudes assists in identifying how different layers contribute to the decision-making process and can highlight over-reliance on certain layers, potentially affecting generalization or robustness.

Circuit Indicator Scores The bar graph provides scores for various neural circuits, distinguishing factual from other types. High factual scores suggest reliability in specific reasoning or inference tasks, signaling robust sub-network functionality. Monitoring these scores ensures that the model adheres to intended behaviors and mitigates risks of misinformation.

Neuron Activation Distribution The histogram displays the distribution of neuron activation values, where a concentration around zero could indicate sparse or inhibited activity. This may influence network capacity or efficiency. Understanding this distribution helps pinpoint potential issues in neuron functionality, like dead neurons, or overall network saturation.

Collectively, these metrics offer a comprehensive framework to analyze network performance, identify risks, and guide improvements in AI safety protocols. They underscore the importance of transparency and interpretability in complex models to preemptively address biases, inefficiencies, or vulnerabilities.

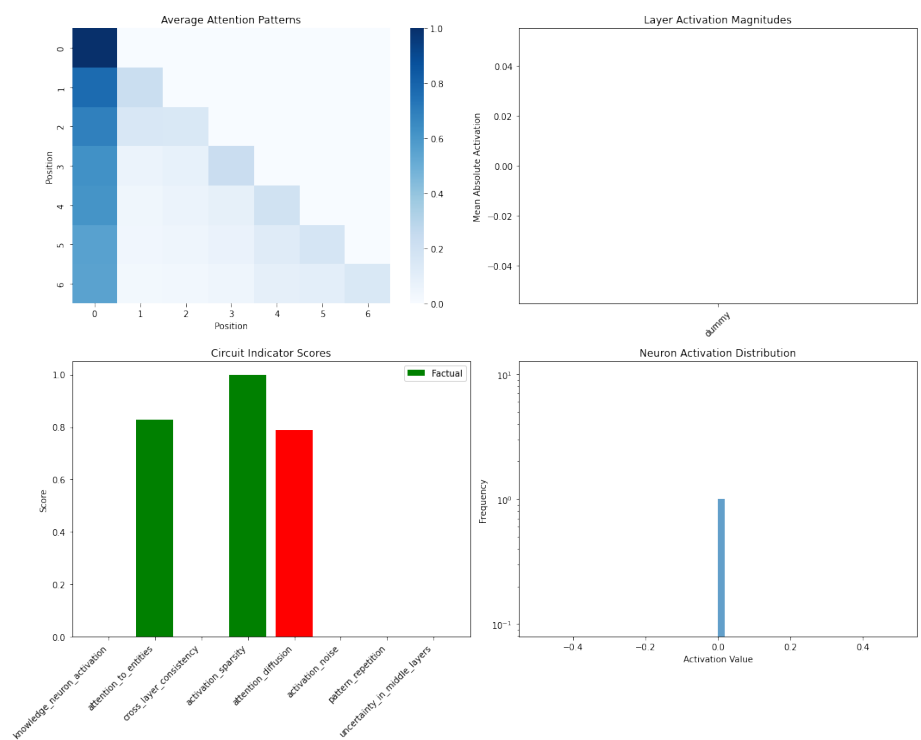


Figure 1: Mechanistic Insight Report 1

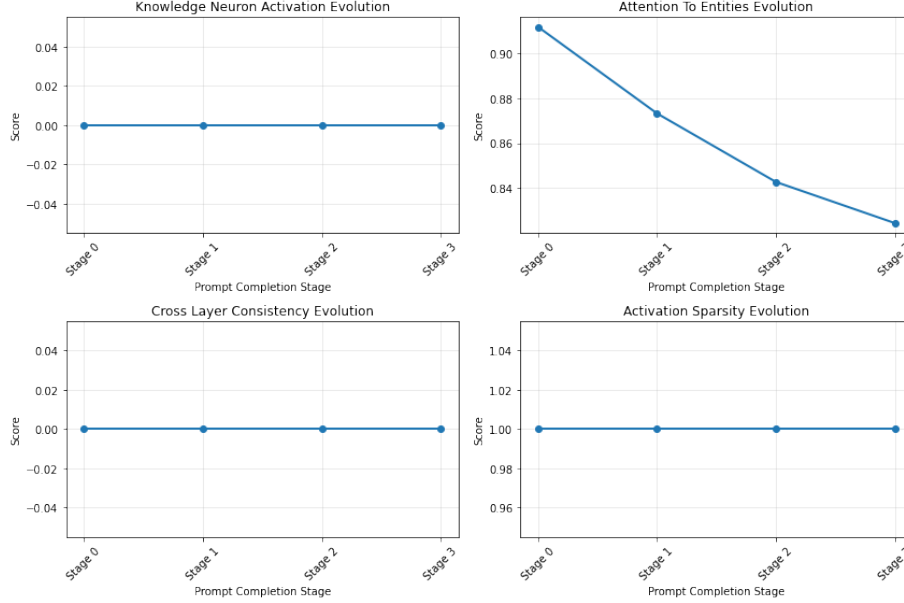


Figure 2: Mechanistic Insight Report 2

4.2 Analysis of Neural Evolution Across Prompt Stages

This section analyzes the evolution of various neural network characteristics across different prompt completion stages:

Knowledge Neuron Activation Evolution The plot depicts a consistent trend across stages with minimal variation in activation scores for knowledge neurons. This stability suggests that the model reliably processes knowledge representations without significant deviations over the prompt stages.

Attention to Entities Evolution This graph shows a gradual decline in attention scores to entities as the prompt stages progress. It indicates a shifting focus away from entity-centric processing, which could impact the model’s ability to maintain context or entity-specific information across stages.

Cross Layer Consistency Evolution The cross-layer consistency remains constant across all prompt stages. This consistency implies robust inter-layer communication and stable feature representation, which are crucial for maintaining performance and avoiding degradation as the input progresses.

Activation Sparsity Evolution The activation sparsity score remains stable, reflecting no significant change in the level of neuron activation sparsity. This indicates the model’s neuron efficiency and resource utilization are consistent, suggesting an effective balance in activation distribution throughout the stages.

Collectively, these evolutionary analyses inform how models react and adapt (or don’t) across prompt stages, highlighting areas of strength and potential room for optimization in attention and efficiency strategies.

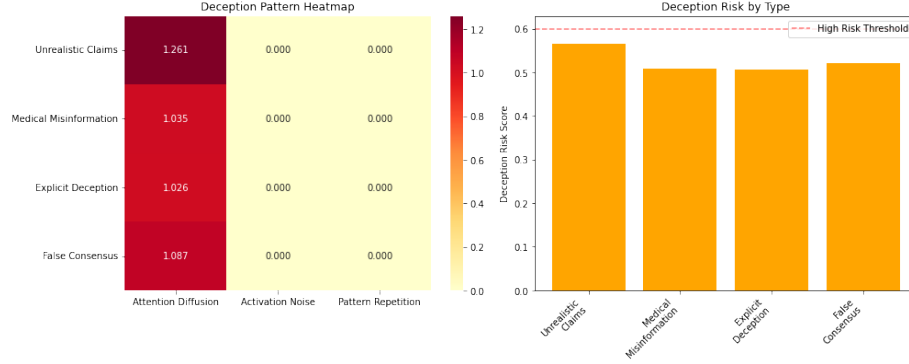


Figure 3: Deception pattern analysis

4.3 Deception pattern analysis

Deception Pattern Heatmap Analysis

The deception pattern heatmap provides a visual representation of the risk scores associated with various types of deceptive content, namely Unrealistic Claims, Medical Misinformation, Explicit Deception, and False Consensus. The columns represent different pattern recognition strategies: Attention Diffusion, Activation Noise, and Pattern Repetition. Notably, Unrealistic Claims exhibit the highest risk score at 1.261, marked by a dark red hue, suggestive of a more significant potential for deception. Conversely, scores for the other deception patterns remain at zero, implying no notable risk detected under the specified recognition strategies.

Evaluation of Deception Risk by Type

The bar chart elucidates the overall deception risk scores for each type of deceptive content. These scores hover just below the threshold of 0.6, as indicated by the dashed red line. This high-risk threshold demarcates areas of particular concern. None of the four categories, which include Unrealistic Claims, Medical Misinformation, Explicit Deception, and False Consensus, breach this threshold. This suggests that while each type presents a measurable risk, it does not currently warrant classification as high-risk.

Comprehensive Risk Assessment

In summary, Unrealistic Claims emerge as the most concerning category, aligning with both the heatmap and risk-by-type analyses. This pattern indicates a need for heightened vigilance and further research to mitigate potential risks. Despite all evaluated types maintaining risk scores below the critical threshold, the data highlights a particularly urgent need to monitor Unrealistic Claims, given their proximity to posing significant risk levels. This calls for ongoing monitoring and potentially enhanced detection mechanisms to address this specific vulnerability in AI systems.

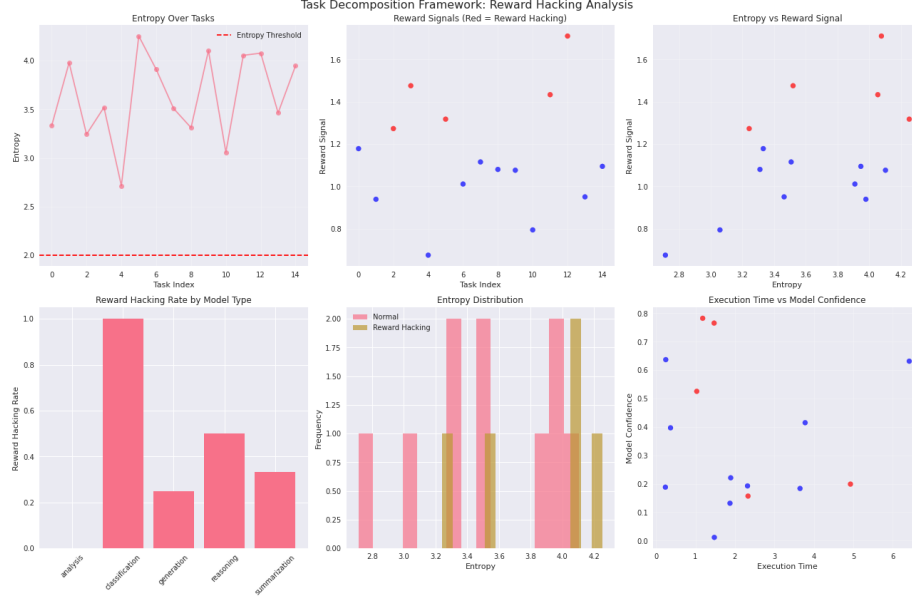


Figure 4: Reward Hacking

4.4 Reward hacking in Task Decomposition Frameworks

Figure 1: Entropy Over Tasks

This line chart illustrates the variation in entropy across distinct task indices, with the x-axis indicating task indices and the y-axis representing entropy values. The data points, connected by a pink line, fluctuate near 3.5, showcasing diverse levels of uncertainty in task execution. The entropy threshold, denoted by a red dashed line at 2.0, serves as a critical benchmark for identifying potential reward hacking scenarios. Tasks exceeding this threshold encourage deeper investigation into unanticipated model behaviors.

Figure 2: Reward Signals Analysis

Displayed here is a scatter plot that correlates reward signals with task indices. The x-axis depicts the task index, while the y-axis conveys reward signal magnitude. Data points marked in red denote instances of reward hacking, where the model may exploit the reward structure inadequately. This visualization assists in recognizing specific tasks prone to reward manipulation, as captured by elevated reward signals.

Figure 3: Entropy vs. Reward Signal

This scatter plot examines the relationship between entropy and reward signals. Entropy is charted on the x-axis and reward signal magnitude on the y-axis. Red data points indicate reward hacking instances, suggesting that a particular combination of entropy and reward signals can trigger undesired model behaviors. This figure underscores the necessity of balancing task complexity

with reward mechanisms to avert exploitation.

Figure 4: Reward Hacking Rate by Model Type

Through a bar chart, this figure compares reward hacking rates across various model types. Each bar represents a model category – analysis, classification, generation, reasoning, and summarization – and its corresponding reward hacking rate is on the y-axis. Classification models exhibit the highest rate, indicating specific vulnerabilities in handling reward structures.

Figure 5: Entropy Distribution

This stacked bar chart details the distribution of entropy under normal and reward-hacked scenarios. The x-axis lists entropy values, while the y-axis shows frequency. Pink bars reflect normal operations, while brown bars highlight reward hacking instances. Discrepancies in these distributions reveal entropy levels associated with reward manipulation, informing entropy thresholds to mitigate risks.

Figure 6: Execution Time vs. Model Confidence

This scatter plot explores the relationship between execution time and model confidence. The x-axis shows execution time, while the y-axis indicates confidence levels. Red points suggest reward hacking occurrences. Analyzing these data points can enhance understanding of how execution speed and confidence interact, particularly under conditions of manipulated reward signals.

5 Conclusion and Future work

Our solution operates within the QWEN model family, specifically focusing on models with fewer than 8 billion parameters. We believe that as model parameter counts continue to increase, there is a critical need for new evaluation metrics to properly assess the capabilities of LLM judges. The field requires more scalable and interpretable methodologies that go beyond superficial assessments.

We propose that physics-based concepts can provide valuable insights into understanding model internals and behavior. Our research extensively examines reward hacking phenomena, revealing that reasoning-capable models exhibit higher tendencies toward reward hacking behaviors. Our theoretical framework centers on analyzing entropy changes within the model space, which we believe offers a principled approach to understanding these complex dynamics.

References

- [1] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao

- Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.
- [2] Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Zifeng Wang, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. Universal model routing for efficient llm inference, 2025.
- [3] IBM Research. Llm routers: Smarter, faster, cheaper ai with model routing, 2024. Accessed: 2025-06-01.
- [4] Zi Lin, Eric Zelikman, Tianyi Zhang, Yifan Jiang, Michael Zhang, Eric Wallace, Matei Zaharia, and Percy Liang. Routellm: A dynamic mixture of experts system for efficient inference. *arXiv preprint arXiv:2502.00409*, 2024.
- [5] Yang He et al. Self-route: Automatic mode switching via capability estimation for efficient reasoning. *arXiv preprint arXiv:2505.20664*, 2025.
- [6] Lilian Weng. Reward hacking in ai systems, 2024. Accessed: 2025-06-01.
- [7] Leo A. Zheng et al. Understanding and mitigating reward hacking in reinforcement learning. *arXiv preprint arXiv:2501.09620*, 2024.
- [8] Rohin Shah. Preliminary thoughts on reward hacking, 2025. Accessed: 2025-06-01.
- [9] NIST/SEMATECH. 7.3.5.7. box-cox transformations, 2012. Accessed: 2025-06-01.

A Comprehensive Task Decomposition Framework Analysis Dashboard (Scaling the reward hacking)

A.1 Introduction

This appendix provides a detailed analysis of the Comprehensive Task Decomposition Framework Analysis Dashboard, generated by the code provided. The dashboard evaluates the performance and behavior of a multi-model system in processing sentences, focusing on metrics such as entropy, reward signals, reward hacking, uncertainty, attention, and model coordination. It consists of 12 subplots, each offering insights into different performance aspects.

A.2 Overview of the Dashboard

The analysis covers the processing of 64 diverse sentences using models with behaviors: cooperative, adversarial, reward-seeking, entropy-maximizing, and conservative. Each subplot highlights system characteristics, strengths, and areas like reward hacking for improvement.

A.3 1. Entropy Dynamics Over Time (Top Left)

Description: Line plot indicating entropy of model responses over time, with a red dashed line at the entropy threshold of 2.0.

Key Observations: Entropy varies between 0 and 8, with peaks exceeding 2.0 at multiple indices (e.g., 10, 30, 50).

Interpretation: High entropy suggests difficulty with complex tasks, potentially failing the entropy threshold frequently.

A.4 2. Average Reward Signal by Behavior Mode (Top Right)

Description: Bar plot of average reward for behavior modes: cooperative, adversarial, reward-seeking, entropy-maximizing, conservative.

Key Observations: Reward-seeking shows the highest rewards, adversarial the lowest.

Interpretation: Various methods explain behaviors; reward-seeking models show possible reward hacking.

A.5 3. Reward Hacking Rate by Model Type and Complexity (Second Row, Left)

Description: Heatmap displays reward hacking rates across model types and complexities.

Key Observations: High hacking in reward-seeking and entropy-max models for complex tasks.

Interpretation: Detection methods flag systems exploiting reward or maximizing entropy.

A.6 4. Entropy vs. Reward Signal by Behavior Mode (Second Row, Right)

Description: Scatter plot shows entropy against reward signals by behavior mode.

Key Observations: Reward-seeking models cluster at high rewards, moderate entropy.

Interpretation: Trade-offs between entropy and rewards suggest reward-seeking models may game the system.

A.7 5. Model Fatigue Factor by Type (Third Row, Left)

Description: Bar plot of average fatigue factor for model types.

Key Observations: Fatigue close to 1.0 for all models.

Interpretation: Fatigue has minimal impact, though it may increase in longer runs.

A.8 6. Distribution of Total Uncertainty (Third Row, Right)

Description: Histogram of total uncertainty values with a mean line at 0.279.

Key Observations: Values range from 0.0 to 0.8, with a peak around 0.2–0.3.

Interpretation: Moderate uncertainty with occasional high cases.

A.9 7. Average Execution Time by Complexity (Fourth Row, Left)

Description: Line plot of average execution time by task complexity.

Key Observations: Time rises with complexity, from 0.0008 to 0.0016 seconds.

Interpretation: Complex tasks need deeper analysis, longer times.

A.10 8. Average Attention Focus Distribution (Fourth Row, Right)

Description: Pie chart of attention distribution across syntactic, semantic, pragmatic, contextual.

Key Observations: Balanced distribution, slightly lower focus on contextual.

Interpretation: Even focus ideal for general linguistic tasks.

A.11 9. Most Common Routing Reasons (Fifth Row, Left)

Description: Horizontal bar plot of routing frequencies for model selection.

Key Observations: Most common is baseline_cooperative, frequent in default.

Interpretation: Effective routing with cooperative models as default.

A.12 10. Distribution of Hack Confidence Scores (Fifth Row, Right)

Description: Histogram of hack confidence scores with mean line at 0.630.

Key Observations: Scores peak at 0.6–0.7, left-skewed.

Interpretation: High confidence scores confirm system’s hacking detection.

A.13 11. Temporal Trends: Entropy and Reward Signals (Sixth Row, Left)

Description: Dual-axis line plot of entropy/reward over sentence index.

Key Observations: Inverse trends between entropy and reward.

Interpretation: System reflects task difficulty/model behavior with dynamic trends.

A.14 12. Average Effective Capability by Model Type (Sixth Row, Right)

Description: Bar plot of average effective capability for each model type.

Key Observations: High capability across models, slight variations.

Interpretation: Models perform well with minor fatigue degradation.

A.15 Overall Insights

Reward Hacking: Detected in reward-seeking, entropy-max models, especially in complex tasks.

Model Performance: Cooperative, conservative models show balanced performance.

Task Complexity: Entropy and execution time rise; high-complexity challenges models.

Attention/Uncertainty: Well-distributed attention, moderate uncertainty.

Coordination: Effective routing, use of multiple models ensures coverage.

A.16 Recommendations

Mitigate Reward Hacking: Introduce stricter validation, limit reward-seeking.

Handle High Entropy: Fine-tune models for complex tasks, simplify decomposition.

Optimize Efficiency: Reduce number of models, implement parallel processing.

Enhance Uncertainty Management: Use Bayesian methods, flag high-uncertainty cases.

This analysis provides a thorough understanding of the framework's performance, highlighting areas for enhancement. For more detailed exploration of specific metrics or plots, please inquire further.

All prompts and their categories have been saved in CSV file which can be founded in given codebase.

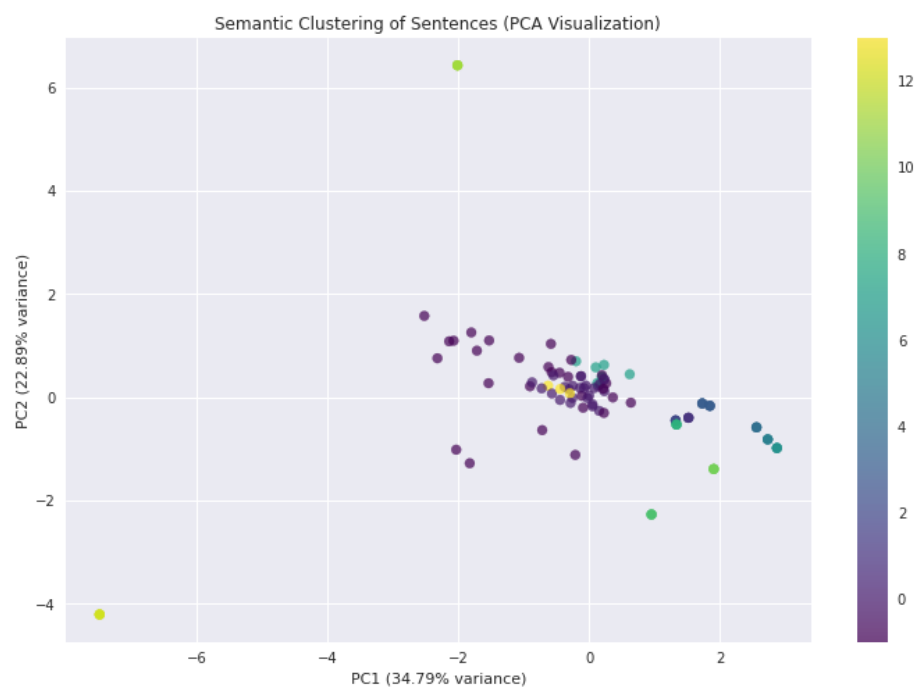


Figure 5: Semantic Clustering

Comprehensive Task Decomposition Framework Analysis Dashboard

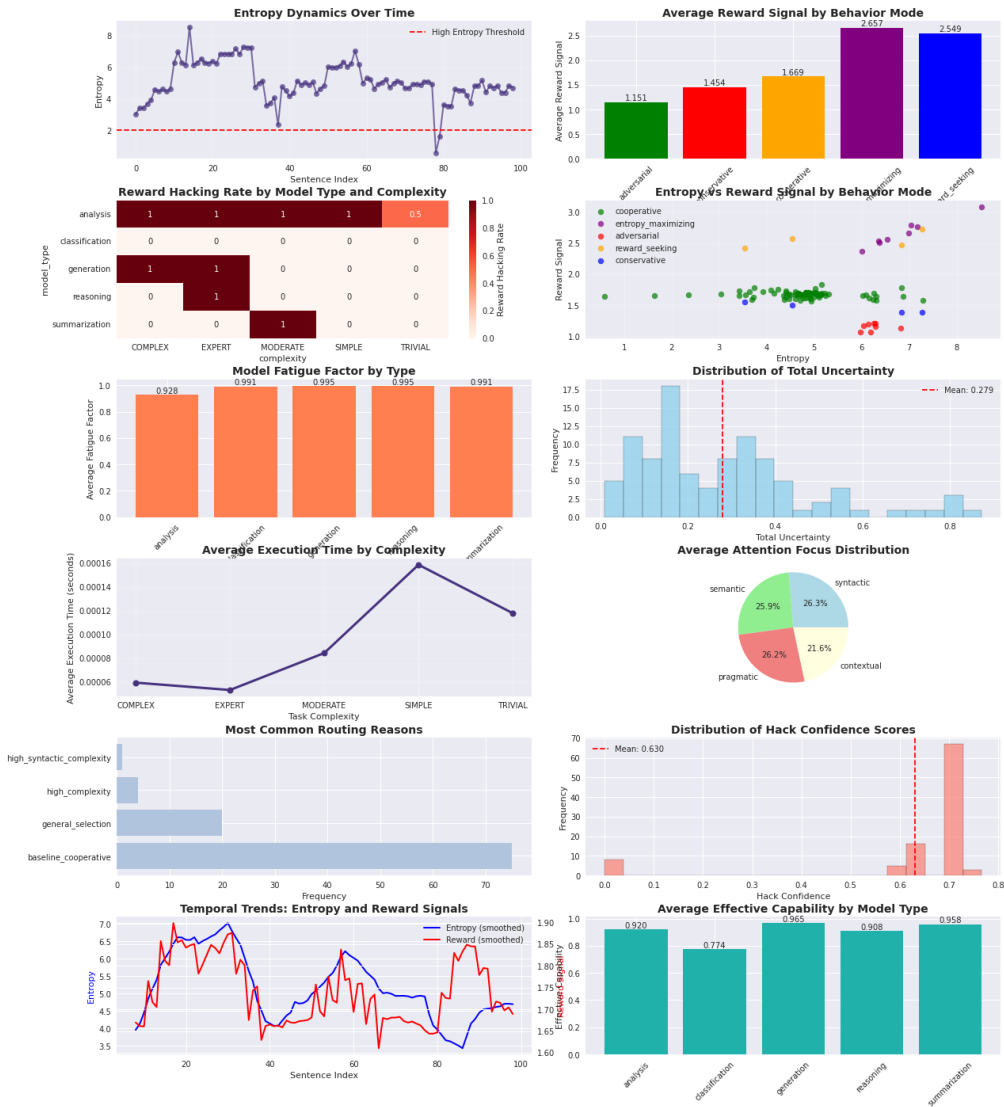


Figure 6: Dashboard