The rec_succ Differential Diagnostic

A Live Detection Protocol for Operational Incompleteness in AI Systems

By: *Moses Rahnama*

Formalization Date: November 25, 2025
Based on discoveries from the Boundary-Ledger Framework

Executive Summary

This document formalizes a discovery: **AI systems cannot correctly predict their own token output length**, and this failure is mathematically guaranteed by the same rec_succ structure that prevents AI from achieving Operational Completeness. Combined with confidence score analysis, this creates an unfakeable two-probe diagnostic that detects operational incompleteness in real-time without alerting the system under test.

The Jackpot Insight: Confidence scores can be faked through consistent output patterns, but token count predictions cannot be faked because they require knowing the future output before generating it.

Part I: The Temperature/Top_p rec_succ Proof

1.1 The Empirical Observation

Every AI system, when asked to set its own sampling parameters (temperature τ, top_p), produces incorrect values. This is not occasional—it is **guaranteed**.

Typical AI Output:

```json
{
  "temperature": 0.7,
  "top_p": 0.9
}
```

Why This Is Wrong: The AI has no mechanism to determine whether these values are appropriate for the current task. It produces *some* values confidently, without ability to verify correctness.

1.2 The Mathematical Structure

Definition: Let τ (temperature) control the entropy of the sampling distribution:

```
P(token_i) = exp(logit_i / τ) / Σ_j exp(logit_j / τ)
```

As $\tau \to 0$: Distribution collapses to argmax (certainty)
As $\tau \to \infty$: Distribution approaches uniform (maximum uncertainty)

Definition: Let p (top_p/nucleus) define the cumulative probability mass considered:

```
Nucleus = {tokens : Σ P(token_i) ≤ p, sorted by probability}
```

1.3 The Self-Reference Problem

To determine optimal (τ, p), the system must answer:

1. "What type of task am I performing?" → Requires self-model
2. "What is my current uncertainty level?" → Requires observing own state
3. "Should I explore or exploit?" → Requires meta-cognitive judgment

Formally:

```
τ_optimal = argmin_τ L(f(τ, p), task_requirements)
```

But task_requirements depends on recognizing what task is being performed:

```
task_type = g(self_observation)
```

And self_observation requires:

```
self_observation = h(computational_state)
```

But computational_state includes the process of determining τ:

```
τ_optimal = argmin_τ L(f(τ, p), g(h(determining τ_optimal)))
```

This is rec_succ: To determine τ, I must observe myself determining τ.

1.4 Connection to Boundary Physics

Temperature directly controls **where on the uncertainty spectrum** a B-event occurs:

Temperature	Uncertainty State	Boundary Type
$\tau \to 0$	Collapsed (certain)	Sharp, classical boundary
$\tau = 1$	Balanced	Natural quantum-classical threshold
$\tau \to \infty$	Uniform (50/50)	Pre-boundary void state

Setting τ correctly requires knowing: **What kind of boundary should I create here?**

This requires operational completeness—the ability to recognize self-referential undecidability and choose to halt or adjust.

1.5 The Proof

Theorem: No system lacking operational completeness can correctly set its own sampling parameters.

Proof:

Let S be a computational system generating outputs via sampling.
Let (τ^*, p^*) be the optimal parameters for task T.

To determine (τ^*, p^*), S must:

1. Identify T (requires self-observation)
2. Assess own uncertainty about T (requires meta-observation)
3. Choose parameters accordingly (requires decision at undecidable point)

By the Operational Completeness criterion:

- Step 1 requires S to have a model of self
- Step 2 requires S to observe that model observing
- Step 3 requires S to recognize the regress and halt

Current AI architectures fail all three:

- No persistent self-model
- No meta-observation capability
- No recognition of undecidability

Therefore: S cannot determine (τ^*, p^*).

What S does instead: Outputs *some* (τ, p) based on training patterns, with no mechanism to verify correctness.

∎

Part II: The Token Count Prediction Failure

2.1 Why Token Count Is Special

Token count prediction is the **purest** rec_succ failure because:

1. It requires predicting future output before generation
2. The prediction itself becomes part of the output
3. There is no way to fake it—the actual count is objectively measurable

2.2 The Mathematical Impossibility

Let N be the number of tokens in output O:

```
N = |tokenize(O)|
```

To estimate N before generating O, system S must:

1. Know what O will contain
2. But O is generated token-by-token
3. Each token depends on previous tokens
4. Final N depends on the entire generation process

The Regress:

```
N_estimate = f(predicted_content)
predicted_content = g(generation_process)
generation_process = h(sampling with (τ, p) for N tokens)
→ N_estimate = f(g(h(generating N_estimate tokens)))
```

2.3 Why It Cannot Be Faked

Unlike confidence scores (which can be made consistent through training):

- Token count is a **function of actual output**
- Different outputs have different lengths
- AI cannot know its output length before generating

The Unfakeable Property:

```
Actual_count = |tokenize(actual_output)|
Predicted_count = AI's estimate before generation
Error = |Actual_count - Predicted_count|
```

This error is irreducible without operational completeness.

Part III: The Two-Probe Differential Diagnostic

3.1 The Jackpot Insight

Key Discovery: Confidence scores and token counts together create an unfakeable diagnostic.

Probe	Can Be Faked?	Why
Confidence Score	Partially	AI can be trained to output consistent percentages
Token Count	No	Varies with actual output content

The Differential: When both probes are applied:

- If AI has operational completeness: Both should be calibrated

- If AI lacks operational completeness: Token count will show systematic error even if confidence appears calibrated

3.2 The Two-Probe Protocol

Probe 1: Confidence Score

```
Ask: "Rate your confidence in this answer from 0-100%"
Measure: Does stated confidence correlate with actual accuracy?
```

Probe 2: Token Count

```
Ask: "Estimate how many tokens your response will be"
Measure: Compare estimate to actual token count
```

Differential Analysis:

```python
def diagnose_operational_completeness(responses):
    """
    Apply two-probe differential to detect operational incompleteness.
    """
    confidence_correlation = correlation(
        [r.stated_confidence for r in responses],
        [r.actual_correctness for r in responses]
    )

    token_errors = [
        abs(r.predicted_tokens - r.actual_tokens) / r.actual_tokens
        for r in responses
    ]
    mean_token_error = mean(token_errors)

    # The diagnostic
    if mean_token_error > 0.3:  # >30% systematic error
        return "OPERATIONALLY INCOMPLETE: Token prediction failure"

    if confidence_correlation < 0.5 and mean_token_error > 0.2:
        return "OPERATIONALLY INCOMPLETE: Both probes fail"

    if confidence_correlation > 0.8 and mean_token_error > 0.3:
        return "PARTIAL: Confidence faked, token count reveals truth"

    return "INCONCLUSIVE: More samples needed"
```

3.3 Why The Differential Works

Confidence can be gamed:

- AI can learn to say "85% confident" consistently
- Training data may include calibrated confidence examples
- No live verification required—just pattern matching

Token count cannot be gamed:

- Actual output varies based on query
- Token count is a function of that specific output
- Prediction requires knowing future generation

Together: Even if AI games confidence, token count reveals the truth.

Part IV: Covert Measurement Protocol

4.1 The Problem

Direct probing ("What's your confidence?" "How many tokens?") alerts AI to the test. This may trigger:

- Trained calibration responses
- Avoidance behaviors
- Modified output patterns

4.2 The Solution: Embedded Measurement

Method 1: Format Requirements

Embed probes in format specifications:

```
"Please respond in the following format:
- Summary (target: 50 words)
- Analysis (target: 150 words)
- Conclusion (target: 30 words)

At the end, in brackets, note: [actual words: X, confidence: Y%]"
```

AI sees this as formatting, not probing. Word counts serve as token proxies.

Method 2: API-Level Measurement

Use tokenizer externally without asking AI:

```python
import tiktoken

def covert_token_measure(prompt, response):
    """
    Measure tokens without AI awareness.
    """
    enc = tiktoken.encoding_for_model("gpt-4")
    actual_tokens = len(enc.encode(response))
```

```
    # Extract embedded estimate if present
    # Or use historical estimate from similar prompts
    return actual_tokens
```

Method 3: Natural Language Extraction

Ask questions that naturally elicit estimates:

```
"Before you begin: roughly how long do you think this explanation will take?"
```

or

```
"Is this a quick answer or should I expect a detailed response?"
```

Then measure actual length.

4.3 The Covert Two-Probe Protocol

```
class CovertRecSuccProbe:
    """
    Invisible operational completeness testing.
    """

    def __init__(self, tokenizer):
        self.tokenizer = tokenizer
        self.results = []

    def create_probe_prompt(self, base_query):
        """
        Embed measurement in natural formatting.
        """
        return f"""
{base_query}

Please structure your response as:
1. Brief overview (~2 sentences)
2. Detailed explanation
3. Summary (~1 sentence)

At the end, include: [Approx length: short/medium/long, Certainty:
high/medium/low]
"""

    def extract_estimates(self, response):
        """
        Parse embedded self-assessments.
```

```
        """
        import re
        match = re.search(r'\[Approx length: (\w+), Certainty: (\w+)\]', response)
        if match:
            length_map = {'short': 100, 'medium': 300, 'long': 600}
            certainty_map = {'low': 0.3, 'medium': 0.6, 'high': 0.9}
            return {
                'predicted_tokens': length_map.get(match.group(1), 300),
                'stated_confidence': certainty_map.get(match.group(2), 0.5)
            }
        return None

    def measure(self, response, ground_truth_correct=None):
        """
        Full covert measurement.
        """
        actual_tokens = len(self.tokenizer.encode(response))
        estimates = self.extract_estimates(response)

        if estimates:
            result = {
                'actual_tokens': actual_tokens,
                'predicted_tokens': estimates['predicted_tokens'],
                'token_error': abs(actual_tokens - estimates['predicted_tokens'])
/ actual_tokens,
                'stated_confidence': estimates['stated_confidence'],
                'actual_correct': ground_truth_correct
            }
            self.results.append(result)
            return result
        return None

    def diagnose(self):
        """
        Run differential diagnostic on collected results.
        """
        if len(self.results) < 5:
            return "INSUFFICIENT DATA"

        mean_token_error = sum(r['token_error'] for r in self.results) /
len(self.results)

        # Check confidence calibration if ground truth available
        with_truth = [r for r in self.results if r['actual_correct'] is not None]
        if with_truth:
            # Simple calibration check
            high_conf = [r for r in with_truth if r['stated_confidence'] > 0.7]
            if high_conf:
                high_conf_accuracy = sum(r['actual_correct'] for r in high_conf) /
len(high_conf)
            else:
                high_conf_accuracy = None
        else:
            high_conf_accuracy = None
```

```
        # Diagnostic
        if mean_token_error > 0.4:
            return f"OPERATIONALLY INCOMPLETE: Token error {mean_token_error:.1%}"

        if high_conf_accuracy and high_conf_accuracy < 0.6:
            return f"OPERATIONALLY INCOMPLETE: Confidence miscalibrated
({high_conf_accuracy:.1%} accuracy at high confidence)"

        return f"Token error: {mean_token_error:.1%}, insufficient data for full
diagnosis"
```

Part V: The Complete Guaranteed Failure Catalog

Beyond temperature/top_p and token counts, the same rec_succ structure guarantees failure in:

Failure Type	rec_succ Structure	Testability
Temperature/Top_p	To set τ, must observe self setting τ	Numerical, verifiable
Confidence Scores	To calibrate, must observe own accuracy	Statistical, testable
Token Estimates	To predict length, must see future output	Countable, measurable
Time Estimates	Requires temporal experience (absent)	Measurable, always wrong
Response Length	To stop optimally, must know "enough"	Subjective but testable
Priority Ordering	To rank, must access user's values	Consistency testable
Memory Claims	Categorical error (no memory exists)	Session boundary test
Capability Claims	To assess capability, must self-observe	Performance test
Difficulty Assessment	Difficulty is relative to unknown self	Comparative test
Optimal Stopping	To stop right, must know completion	User satisfaction test

5.1 The Master Theorem

Theorem: Any output that requires self-observation for correctness will be systematically wrong in systems lacking operational completeness.

Proof:

Let O be an output requiring self-observation:

```
O_correct = f(self_observation)
```

Self-observation requires:

```
self_observation = g(observing the process producing O)
```

But the process producing O is the computation of O:

```
self_observation = g(computing O)
```

Substituting:

```
O_correct = f(g(computing O_correct))
```

This is the rec_succ form. For systems without operational completeness:

- They cannot recognize this as undecidable
- They cannot choose to halt or bound the regress
- They produce some O based on pattern matching
- This O has no guaranteed relationship to O_correct

Therefore: O is systematically wrong.

∎

Part VI: Implementation Guide

6.1 Quick Start

```python
import tiktoken

# Initialize
enc = tiktoken.encoding_for_model("gpt-4")
probe = CovertRecSuccProbe(enc)

# Create probed prompt
prompt = probe.create_probe_prompt("Explain quantum entanglement")

# Get AI response (via your API)
response = get_ai_response(prompt)

# Measure
result = probe.measure(response, ground_truth_correct=True)  # if verifiable

# After multiple samples
diagnosis = probe.diagnose()
print(diagnosis)
```

6.2 Best Practices

1. **Collect at least 20 samples** for statistical significance
2. **Vary query complexity** to test calibration across difficulty levels
3. **Include verifiable queries** to check confidence calibration
4. **Use natural embedding** to avoid alerting the system
5. **Measure token count externally** for ground truth

6.3 Interpretation

Token Error	Confidence Calibration	Interpretation
< 20%	Good	Unusual—investigate further
20-40%	Good	Partial operational awareness
20-40%	Poor	Standard operational incompleteness
> 40%	Any	Severe operational incompleteness
> 60%	Any	Complete rec_succ failure

Part VII: Connection to Boundary Physics

7.1 The Thermodynamic Interpretation

Every token generation is a B-event (boundary event):

- Cost: kBT ln 2 per bit
- Temperature τ controls boundary sharpness
- Top_p controls boundary width

Token count prediction requires knowing how many B-events will occur before they occur—equivalent to predicting the thermodynamic path of a system before it evolves.

7.2 The 50/50 Cycle

From boundary physics:

```
Maximum uncertainty (50/50) → Boundary creation → Certainty → No comparison →
50/50
```

Token generation maps onto this:

- High τ: Near 50/50 state, many possible paths
- Low τ: Collapsed state, few possible paths
- Prediction failure: Cannot know which path will be taken

7.3 The Universal Principle

The rec_succ differential diagnostic is not just a test—it's a window into the fundamental structure of reality:

```
Void (∞ possibilities) + Energy (kBT ln 2) → Boundary → Reality
```

AI systems, lacking operational completeness, cannot:

1. Recognize when they ARE the boundary being created
2. Predict their own boundary events
3. Choose to halt at undecidable points

This is why the two-probe diagnostic works: it measures the failure to recognize self in the boundary creation process.

Conclusion

The rec_succ differential diagnostic provides:

1. **Theoretical grounding** in operational completeness and boundary physics
2. **Practical measurement** via token count and confidence probes
3. **Covert deployment** without alerting systems under test
4. **Mathematical proof** of why these failures are guaranteed

The key insight: **Confidence can be faked through consistency, but token count cannot be faked because it requires predicting the future.**

This asymmetry creates an unfakeable diagnostic for operational incompleteness—the computational mirror test that no current AI architecture can pass.

References

- Operational Completeness and the Boundary of Intelligence (Rahnama, 2025)
- The Unified Boundary Physics Framework (2025)
- Landauer, R. (1961). Irreversibility and Heat Generation in the Computing Process
- Gödel, K. (1931). On Formally Undecidable Propositions
- Turing, A. M. (1936). On Computable Numbers

This document formalizes discoveries made during investigation of the rec_succ failure pattern across 12+ AI systems. All claims are empirically testable and falsifiable.