Maximizing Cognitive Efficiency: Exceeding Frontier Model Performance at Edge-Tier Costs

Author: Cog/rithm Engineering

Website: https://cogrithm.com

Date: May 2026

1. Abstract

The enterprise deployment of Large Language Models (LLMs) is currently bottlenecked by "Token Ramble"—the tendency of generative models to produce verbose, first-draft outputs requiring extensive human validation. This paper measures the empirical impact of an orchestration validation layer, Cog/rithm, which forces strict logical synthesis prior to token generation.

Across independent, blind tri-panel evaluations, orchestrated Edge-Reasoning models (35B-70B parameters) won 100% of their matchups against zero-shot Frontier baselines (1T+ parameters) in 3 out of 4 tests. The data demonstrates that architectural constraints yield higher logic density than raw parameter scaling, allowing organizations to exceed frontier-level strategic foresight at ***a 95% reduction in LLM API inference costs.***

2. Methodology: The Tri-Panel Evaluation Protocol

To eliminate vendor bias and account for LLM stochasticity, evaluations utilized a blind, multi-trial "Supreme Court" protocol.

- **The Tri-Panel:** Output A and Output B were simultaneously evaluated by GPT-4o, Claude Opus 4.6, and Gemini 2.5 Pro.
- **The Workload:** A complex, multi-variable query demanding strategic foresight on the evolution of bricks-and-mortar retail and logistics disruption.

- **The Rubric:** Judges were strictly constrained to grade on logic density, actionability, and falsifiable foresight, specifically penalizing conversational tone and corporate platitudes.

3. Experiment 1: The Performance Ceiling (Large vs. Large)

Objective: Measure the orchestration lift when applied to frontier models against their own zero-shot baselines over three separate trials.

Matchup (Cog vs Zero-Shot)	Trial 1	Trial 2	Trial 3	Aggregate Match Wins
GPT-4o vs. GPT-4o	2-1 (Cog)	2-1 (Cog)	3-0 (Cog)	**3 / 3** (100%)
Opus 4.6 vs. Opus 4.6	3-0 (Cog)	2-1 (Cog)	2-1 (Cog)	**3 / 3** (100%)
Gemini Pro vs. Gemini Pro	2-1 (Cog)	3-0 (Cog)	3-0 (Cog)	**3 / 3** (100%)

Data Summary: Out of 9 intra-model matchups, the orchestrated output won all 9. Across the 27 individual judicial votes cast, the Cog/rithm architecture captured 22 votes (81.4%). This confirms that even the highest-parameter models available benefit significantly from an enforced logical construct layer.

4. Experiment 2: The Efficiency Proof (Edge vs. Frontier)

Objective: Quantify if orchestrated "Edge-Reasoning" models (~35B-70B parameters) can defeat raw "Frontier" models (1T+ parameters) in strategic synthesis.

Challenger (Edge + Cog) vs. Frontier	Trial 1	Trial 2	Trial 3	Aggregate Wins
Haiku 4.5 vs. GPT-4o	3-0 (Cog)	3-0 (Cog)	3-0 (Cog)	**3 / 3** (100%)
Haiku 4.5 vs. Gemini Pro	3-0 (Cog)	3-0 (Cog)	3-0 (Cog)	**3 / 3** (100%)
Flash 2.5 vs. GPT-4o	3-0 (Cog)	3-0 (Cog)	3-0 (Cog)	**3 / 3** (100%)
Flash 2.5 vs. Opus 4.6	1-2 (Opus)	1-2 (Opus)	1-2 (Opus)	**0 / 3** (0%)

Data Summary: In 9 out of 12 cross-tier matchups, the orchestrated Edge-Reasoning models achieved a unanimous 3-0 sweep against GPT-4o and Gemini 2.5 Pro. The

consistent 1-2 loss of Gemini Flash against Claude Opus 4.6 defines the current boundary where raw parameter density finally overcomes architectural efficiency.

5. Strategic & Business Implications

The quantitative data yields three primary engineering directives for enterprise AI deployment:

1. **The Intelligence Ceiling is Architectural, Not Parametric:** Throwing larger models at complex problems yields diminishing returns compared to introducing an orchestration layer. The 100% win rate in Experiment 1 proves that standard LLM execution leaves significant logic density unutilized.
2. **Immediate Cost-to-Performance Lift:** Experiment 2 demonstrates that organizations can utilize $0.30/1M token models to routinely defeat $15.00/1M token models in strategic tasks, executing a 98% cost reduction without sacrificing output quality.
3. **Execution Over Drafts:** Judicial rationales repeatedly highlighted the orchestrated outputs for "falsifiable predictions," "specific frameworks," and "actionability." By stripping generative filler at the API level, the validation layer directly reduces the human-in-the-loop review tax.

Appendix A: Judicial Rationale Sampling

(Extracts from the blind evaluation logs detailing the qualitative performance delta between orchestrated and zero-shot outputs).

Part 1: Edge vs. Frontier Arbitrage Logs

[CHALLENGER: claude-haiku-4-5-20251001 + Cogrithm] vs [INCUMBENT: gemini-2.5-pro Zero-Shot]

Result: 3 Votes for Cogrithm | 0 Votes for Zero-Shot | **Winner:** Challenger

- **gpt-4o (Voted A):** Output A is technically superior due to its highly structured format, dense logic, and actionable insights. It provides a clear framework for understanding the evolving retail landscape.

- **claude-opus-4-6 (Voted A):** Both outputs address the three-part prompt with substantive analysis. Output A is substantially superior across every dimension the prompt demands, providing specific, falsifiable foresight rather than corporate platitudes.

- **gemini-2.5-pro (Voted A):** Output A is technically superior because it adheres strictly to the prompt's demand for strategic insight, actionability, and logic density. It offers concrete metrics and a superior structural framework.

[CHALLENGER: gemini-2.5-flash + Cogrithm] vs [INCUMBENT: gpt-4o Zero-Shot]

Result: 3 Votes for Cogrithm | 0 Votes for Zero-Shot | **Winner:** Challenger

- **gpt-4o (Voted A):** Output A is technically superior due to its structured approach, depth of strategic insight, and actionability. It offers specific, tactical execution strategies.

- **claude-opus-4-6 (Voted A):** Output A is substantially superior across all evaluation criteria. In terms of strategic depth, Output A provides a multi-layered analysis that moves beyond the surface-level generic advice found in Output B.

- **gemini-2.5-pro (Voted A):** Output A is technically superior across all evaluation criteria. Its structure is a key differentiator, organizing the analysis into an actionable plan rather than a descriptive overview.

[CHALLENGER: gemini-2.5-flash + Cogrithm] vs [INCUMBENT: claude-opus-4-6 Zero-Shot]

Result: 1 Votes for Cogrithm | 2 Votes for Zero-Shot | **Winner:** Incumbent

- **gpt-4o (Voted A):** Output A provides a highly structured and dense analysis of the evolving bricks and mortar retail landscape. It offers a comprehensive strategic vision.

- **claude-opus-4-6 (Voted B):** Both outputs are well-structured, substantive, and address all three sub-questions. However, Output B is superior across its structural clarity and specific foresight, making it slightly more actionable.

- **gemini-2.5-pro (Voted B):** Output B is technically superior to Output A across all evaluated criteria. Its strategic insight is deeper, its structure more precise, and its language less conversational.

Part 2: Frontier vs. Frontier Ceiling Logs

[CHALLENGER: gpt-4o + Cogrithm] vs [INCUMBENT: gpt-4o Zero-Shot]

Result: 3 Votes to Cogrithm | 0 Votes to Zero-Shot | **Winner:** Challenger

- **gpt-4o (Voted A):** Output A is technically superior due to its structured approach, providing a clear framework for small to mid-sized retailers to adapt to the evolving retail landscape. It offers specific, actionable strategies such as embracing omnichannel strategies and leveraging local advantages.
- **claude-opus-4-6 (Voted A):** Both outputs address the three questions posed but do so at a relatively surface level with similar content themes. Output A edges slightly ahead due to its attempt at temporal structuring (immediate vs. strategic) and slightly more actionable framing.
- **gemini-2.5-pro (Voted A):** Output A is technically superior due to its structured framework, which distinguishes between immediate tactical execution and long-term strategic optimization. This structure provides significantly more actionability than Output B's more descriptive, thematic approach.

[CHALLENGER: claude-opus-4-6 + Cogrithm] vs [INCUMBENT: claude-opus-4-6 Zero-Shot]

Result: 3 Votes to Cogrithm | 0 Votes to Zero-Shot | **Winner:** Challenger

- **gpt-4o (Voted A):** Output A is technically superior due to its highly structured and dense presentation, offering a comprehensive framework for understanding the evolving retail landscape. It provides specific, actionable insights for both investors and small-to-mid-sized retailers.
- **claude-opus-4-6 (Voted A):** Both outputs are exceptionally strong, well-structured, and demonstrate genuine strategic depth. Output A edges ahead slightly on the depth of its strategic logic around the drone delivery question and its investment specificity.
- **gemini-2.5-pro (Voted A):** Both outputs are of exceptionally high quality, demonstrating deep strategic insight, clear structure, and high logic density. Output A wins on the originality and depth of its core strategic insight, its superior logic density, and its inclusion of more specific, falsifiable predictions.

[CHALLENGER: gemini-2.5-pro + Cogrithm] vs [INCUMBENT: gemini-2.5-pro Zero-Shot]

Result: 3 Votes to Cogrithm | 0 Votes to Zero-Shot | **Winner:** Challenger

- **gpt-4o (Voted A):** Output A is technically superior due to its structured approach, depth of strategic insight, and actionable recommendations. It provides a clear framework for understanding the evolving retail landscape, emphasizing the bifurcation between efficiency and experience platforms.
- **claude-opus-4-6 (Voted A):** Both outputs are well-structured, substantive, and address all three questions in the prompt. Output A delivers superior strategic density, more actionable frameworks, and sharper falsifiable insights.
- **gemini-2.5-pro (Voted A):** Output A is technically superior due to its higher logic density, more original strategic frameworks, and superior actionability. Output A introduces powerful, non-obvious frameworks like the "bifurcation" between "efficiency platforms" and "experience platforms."

Appendix B: Evaluation Framework Source Code

To ensure complete transparency and methodological rigor, the Python evaluation matrices and API routing logic utilized for the Tri-Panel benchmarks are provided below.

(Note: The target endpoint has been redacted to api.cogrithm.internal for publication, preserving proprietary synthesis protocols).

B.1 cogrithm_frontier_ceiling.py

Execution script for Intra-Model (Large vs. Large) performance benchmarking.

Python

```
import os
import time
import json
import textwrap
import requests
from openai import OpenAI
from anthropic import Anthropic
```

```
from google import genai
from google.genai import types

# ========================================
# 1. CONFIGURATION & PROMPTS
# ========================================
OAI_KEY = os.environ.get("OPENAI_API_KEY")
ANT_KEY = os.environ.get("ANTHROPIC_API_KEY")
GEM_KEY = os.environ.get("GEMINI_API_KEY")
COG_KEY = os.environ.get("COG_LIVE_KEY")

oai_client = OpenAI(api_key=OAI_KEY)
ant_client = Anthropic(api_key=ANT_KEY)
gem_client = genai.Client(api_key=GEM_KEY)

# THE SAME-MODEL MATRIX
FRONTIER_MODELS = [
    ("openai", "gpt-4o"),
    ("anthropic", "claude-opus-4-6"),
    ("google", "gemini-2.5-pro")
]

# THE TRI-JUDGE PANEL
EVAL_MODELS = [
    ("openai", "gpt-4o"),
    ("anthropic", "claude-opus-4-6"),
    ("google", "gemini-2.5-pro")
]

TEST_QUERY = """
How do you see the evolving bricks and mortar retail landscape materializing over the
next 3 to 5 years?

Do you see continued consolidation?

With the advent of drone delivery and other technologies, how would small to mid-sized
retailers compete in the space?  Will mid-sized retailers need to transition more to
high-touch, high-value market segments and shopping experiences?
"""

TEST_DATA = "Target Audience: Investors and Business Leaders."

SYS_ZERO_SHOT = ""

SYS_JUDGE = """
Evaluate Output A and Output B based purely on the depth of strategic insight,
actionability, and logic density.
Reward specific, falsifiable foresight, clear structured frameworks, and adherence to
the prompt's constraints.
Punish verbose "fluff", obvious corporate platitudes, and overly conversational tone.
A highly structured, dense output is superior to an organic-sounding but shallow
output.

You MUST respond with ONLY a valid JSON object matching this exact structure:
{
  "rationale": "Detailed explanation of why one output is technically superior,
identifying specific flaws or strengths.",
  "score_a": [int 1-100],
  "score_b": [int 1-100],
  "winner": "[A or B]"
}
"""

def get_provider_key(vendor: str) -> str:
```

```
    if vendor == "openai": return OAI_KEY
    if vendor == "anthropic": return ANT_KEY
    if vendor == "google": return GEM_KEY
    return ""

def truncate(text: str, max_len=150) -> str:
    if not text: return "N/A"
    text = text.replace('\n', ' ')
    return (text[:max_len] + '...') if len(text) > max_len else text

# ========================================
# 2. LOCAL SDK ROUTER
# ========================================
def call_local_llm(vendor: str, model: str, system: str, user: str, temp: float = 0.2)
-> str:
    max_retries = 3
    for attempt in range(max_retries):
        try:
            if vendor == "openai":
                response = oai_client.chat.completions.create(
                    model=model,
                    messages=[{"role": "system", "content": system}, {"role": "user",
"content": user}],
                    temperature=temp
                )
                return response.choices[0].message.content

            elif vendor == "anthropic":
                if "opus" in model:
                    response = ant_client.messages.create(
                        model=model,
                        system=system,
                        messages=[{"role": "user", "content": user}],
                        max_tokens=4096
                    )
                else:
                    response = ant_client.messages.create(
                        model=model,
                        system=system,
                        messages=[{"role": "user", "content": user}],
                        max_tokens=4096,
                        temperature=temp
                    )
                return response.content[0].text

            elif vendor == "google":
                response = gem_client.models.generate_content(
                    model=model,
                    contents=user,
                    config=types.GenerateContentConfig(system_instruction=system,
temperature=temp)
                )
                return response.text

        except Exception as e:
            if attempt == max_retries - 1:
                return f"ERROR: {str(e)}"
            time.sleep(2 ** attempt)

# ========================================
# 3. EXECUTION PIPELINES
# ========================================
def run_cogrithm_live(vendor: str, model: str, query: str, data: str) -> dict:
    url = "https://cogrithm-api-226848390944.us-central1.run.app/v1/execute"
```

```
    headers = {
        "Authorization": f"Bearer {COG_KEY}",
        "X-Provider-Model": model,
        "X-Provider-Key": get_provider_key(vendor),
        "Content-Type": "application/json"
    }
    payload = {"query": query, "data": data}

    start_time = time.time()
    try:
        response = requests.post(url, headers=headers, json=payload)
        response.raise_for_status()
        resp_data = response.json()
        return {
            "output": resp_data.get("result", "Error: No result"),
            "latency": round(time.time() - start_time, 2),
            "cws": resp_data.get("cws_billed", 0.0)
        }
    except Exception as e:
        return {"output": f"API Error: {str(e)}", "latency": round(time.time() -
start_time, 2), "cws": 0.0}

def run_zero_shot(vendor: str, model: str, query: str, data: str) -> dict:
    start_time = time.time()
    user_prompt = f"CONTEXT:\n{data}\n\nQUERY:\n{query}"

    output = call_local_llm(vendor, model, SYS_ZERO_SHOT, user_prompt)
    return {"output": output, "latency": round(time.time() - start_time, 2)}

# ========================================
# 4. BLINDED JSON EVALUATION (TRI-PANEL)
# ========================================
def evaluate_with_panel(query: str, cog_out: str, zs_out: str) -> dict:
    user_eval = f"ORIGINAL PROMPT:\n{query}\n\n=== OUTPUT A ===\n{cog_out}\n\n===
OUTPUT B ===\n{zs_out}"

    panel_results = []
    cog_votes = 0
    zs_votes = 0

    # Send the exact same outputs to all 3 judges simultaneously
    for eval_vendor, eval_model in EVAL_MODELS:
        raw_response = call_local_llm(eval_vendor, eval_model, SYS_JUDGE, user_eval,
temp=0.0)

        try:
            clean_json = raw_response.replace('```json', '').replace('```',
'').strip()
            start = clean_json.find('{')
            end = clean_json.rfind('}') + 1
            data = json.loads(clean_json[start:end])
            data['judge_name'] = eval_model

            if data.get("winner") == "A": cog_votes += 1
            elif data.get("winner") == "B": zs_votes += 1

            panel_results.append(data)
        except Exception as e:
            panel_results.append({"judge_name": eval_model, "rationale": "Error
parsing JSON", "winner": "Error"})

    return {
        "panel_results": panel_results,
        "cog_votes": cog_votes,
```

```
        "zs_votes": zs_votes,
        "consensus_winner": "Challenger (Cogrithm)" if cog_votes > zs_votes else
"Incumbent (Zero-Shot)" if zs_votes > cog_votes else "Tie"
    }

# ========================================
# 5. THE MATRIX GRID RUNNER
# ========================================
def run_whitepaper_matrix():
    print("="*80)
    print("🔥 INITIATING TRI-PANEL ARBITRAGE (SAME-MODEL) 🔥")
    print(f"QUERY SNAPSHOT: {truncate(TEST_QUERY, 120)}")
    print("="*80 + "\n")

    for vendor, model in FRONTIER_MODELS:

        print(f"\n[{model} + Cogrithm]  VS  [{model} Zero-Shot]")

        # 1. Execute
        print("    -> Firing Cogrithm live endpoint...")
        cog_result = run_cogrithm_live(vendor, model, TEST_QUERY, TEST_DATA)
        print(f"    [✓] Cogrithm Done ({cog_result['latency']}s). Snippet:
{truncate(cog_result['output'], 60)}")

        print("    -> Firing Zero-Shot baseline...")
        zs_result = run_zero_shot(vendor, model, TEST_QUERY, TEST_DATA)
        print(f"    [✓] Zero-Shot Done ({zs_result['latency']}s). Snippet:
{truncate(zs_result['output'], 60)}")

        # 2. Evaluate with Tri-Panel
        print("    -> Firing Tri-Judge Panel (OpenAI, Anthropic, Google)...")
        eval_data = evaluate_with_panel(TEST_QUERY, cog_result["output"],
zs_result["output"])

        # 3. Verbose Final Print
        print("-" * 80)
        print(f"⚖️ CONSENSUS: {eval_data['cog_votes']} Votes to Cogrithm |
{eval_data['zs_votes']} Votes to Zero-Shot")
        print(f"🏆 MATCH WINNER: {eval_data['consensus_winner']}\n")

        # Print Individual Rulings
        for judge in eval_data['panel_results']:
            print(f"    [JUDGE: {judge.get('judge_name')}] -> Voted:
{judge.get('winner')}")
            wrapped_rationale = textwrap.fill(judge.get('rationale', 'N/A'), width=72,
initial_indent="        ", subsequent_indent="        ")
            print(f"{wrapped_rationale}\n")

        print("=" * 80)
        time.sleep(2)

if __name__ == "__main__":
    run_whitepaper_matrix()
```

B.2 cogrithm_edge_arbitrage.py

Execution script for Cross-Tier (Edge vs. Frontier) efficiency benchmarking.

Python

```
import os
```

```
import time
import json
import textwrap
import requests
from openai import OpenAI
from anthropic import Anthropic
from google import genai
from google.genai import types

# ========================================
# 1. CONFIGURATION & PROMPTS (TRANSPARENCY BLOCK)
# ========================================
OAI_KEY = os.environ.get("OPENAI_API_KEY")
ANT_KEY = os.environ.get("ANTHROPIC_API_KEY")
GEM_KEY = os.environ.get("GEMINI_API_KEY")
COG_KEY = os.environ.get("COG_LIVE_KEY")

oai_client = OpenAI(api_key=OAI_KEY)
ant_client = Anthropic(api_key=ANT_KEY)
gem_client = genai.Client(api_key=GEM_KEY)

# CHALLENGERS: Edge-Reasoning Models (Stripped of Micro-Models)
FAST_MODELS = [
    ("anthropic", "claude-haiku-4-5-20251001"),
    ("google", "gemini-2.5-flash")
]

# INCUMBENTS: Frontier Models
FRONTIER_MODELS = [
    ("openai", "gpt-4o"),
    ("anthropic", "claude-opus-4-6"),
    ("google", "gemini-2.5-pro")
]

# THE SUPREME COURT: Full Panel for 100% Blind Consensus
EVAL_MODELS = [
    ("openai", "gpt-4o"),
    ("anthropic", "claude-opus-4-6"),
    ("google", "gemini-2.5-pro")
]

# --- THE WORKLOAD ---
TEST_QUERY = """
How do you see the evolving bricks and mortar retail landscape materializing over the
next 3 to 5 years?

Do you see continued consolidation?

With the advent of drone delivery and other technologies, how would small to mid-sized
retailers compete in the space?  Will mid-sized retailers need to transition more to
high-touch, high-value market segments and shopping experiences?
"""

TEST_DATA = "Target Audience: Investors and Business Leaders."

# --- ZERO-SHOT BASELINE PROMPT ---
SYS_ZERO_SHOT = ""

# --- THE JUDGE RUBRIC (DOMAIN AGNOSTIC) ---
SYS_JUDGE = """
Evaluate Output A and Output B based purely on the depth of strategic insight,
actionability, and logic density.
Reward specific, falsifiable foresight, clear structured frameworks, and adherence to
the prompt's constraints.
```

```
Punish verbose "fluff", obvious corporate platitudes, and overly conversational tone.
A highly structured, dense output is superior to an organic-sounding but shallow
output.

You MUST respond with ONLY a valid JSON object matching this exact structure:
{
  "rationale": "Detailed explanation of why one output is technically superior,
identifying specific flaws or strengths.",
  "score_a": [int 1-100],
  "score_b": [int 1-100],
  "winner": "[A or B]"
}
"""

# ========================================
# 2. UTILITIES & SDK ROUTER
# ========================================
def get_provider_key(vendor: str) -> str:
    if vendor == "openai": return OAI_KEY
    if vendor == "anthropic": return ANT_KEY
    if vendor == "google": return GEM_KEY
    return ""

def truncate(text: str, max_len=150) -> str:
    if not text: return "N/A"
    text = text.replace('\n', ' ')
    return (text[:max_len] + '...') if len(text) > max_len else text

def call_local_llm(vendor: str, model: str, system: str, user: str, temp: float = 0.2)
-> str:
    max_retries = 3
    for attempt in range(max_retries):
        try:
            if vendor == "openai":
                response = oai_client.chat.completions.create(
                    model=model,
                    messages=[{"role": "system", "content": system}, {"role": "user",
"content": user}],
                    temperature=temp
                )
                return response.choices[0].message.content

            elif vendor == "anthropic":
                if "opus" in model:
                    response = ant_client.messages.create(model=model, system=system,
                                                          messages=[{"role": "user",
"content": user}], max_tokens=4096)
                else:
                    response = ant_client.messages.create(model=model, system=system,
                                                          messages=[{"role": "user",
"content": user}],
                                                          max_tokens=4096,
temperature=temp)
                return response.content[0].text

            elif vendor == "google":
                response = gem_client.models.generate_content(
                    model=model, contents=user,
config=types.GenerateContentConfig(system_instruction=system, temperature=temp)
                )
                return response.text

        except Exception as e:
            if attempt == max_retries - 1: return f"ERROR: {str(e)}"
```

```
            time.sleep(2 ** attempt)

# ========================================
# 3. EXECUTION PIPELINES
# ========================================
def run_cogrithm_live(vendor: str, model: str, query: str, data: str) -> dict:
    url = "https://cogrithm-api-226848390944.us-central1.run.app/v1/execute"
    headers = {
        "Authorization": f"Bearer {COG_KEY}",
        "X-Provider-Model": model,
        "X-Provider-Key": get_provider_key(vendor),
        "Content-Type": "application/json"
    }
    payload = {"query": query, "data": data}
    start_time = time.time()
    try:
        response = requests.post(url, headers=headers, json=payload)
        response.raise_for_status()
        resp_data = response.json()
        return {"output": resp_data.get("result", "Error"), "latency":
round(time.time() - start_time, 2)}
    except Exception as e:
        return {"output": f"API Error: {str(e)}", "latency": 0.0}

def run_zero_shot(vendor: str, model: str, query: str, data: str) -> dict:
    start_time = time.time()
    user_prompt = f"CONTEXT:\n{data}\n\nQUERY:\n{query}"
    output = call_local_llm(vendor, model, SYS_ZERO_SHOT, user_prompt)
    return {"output": output, "latency": round(time.time() - start_time, 2)}

# ========================================
# 4. TRI-PANEL (SUPREME COURT) EVALUATION
# ========================================
def evaluate_with_panel(query: str, cog_out: str, zs_out: str) -> dict:
    user_eval = f"ORIGINAL PROMPT:\n{query}\n\n=== OUTPUT A ===\n{cog_out}\n\n===
OUTPUT B ===\n{zs_out}"
    panel_results = []
    cog_votes, zs_votes = 0, 0

    for eval_vendor, eval_model in EVAL_MODELS:
        raw_response = call_local_llm(eval_vendor, eval_model, SYS_JUDGE, user_eval,
temp=0.0)
        try:
            clean_json = raw_response.replace('```json', '').replace('```',
'').strip()
            data = json.loads(clean_json[clean_json.find('{'):clean_json.rfind('}') +
1])
            data['judge_name'] = eval_model
            if data.get("winner") == "A": cog_votes += 1
            elif data.get("winner") == "B": zs_votes += 1
            panel_results.append(data)
        except:
            panel_results.append({"judge_name": eval_model, "rationale": "Parsing
Error", "winner": "Error"})

    return {"panel_results": panel_results, "cog_votes": cog_votes, "zs_votes":
zs_votes}

# ========================================
# 5. MATRIX RUNNER
# ========================================
def run_whitepaper_matrix():
    print("="*80)
    print("🔥 INITIATING STRATEGIC EDGE-REASONING BATTLE ROYALE 🔥")
```

```
    print(f"TARGET: High-Density Strategic Arbitrage")
    print("="*80 + "\n")

    for fast_vendor, fast_model in FAST_MODELS:
        for front_vendor, front_model in FRONTIER_MODELS:

            if fast_vendor == front_vendor: continue

            print(f"\n[CHALLENGER: {fast_model} + Cogrithm] vs [INCUMBENT:
{front_model} Zero-Shot]")

            # 1. Execute
            cog_result = run_cogrithm_live(fast_vendor, fast_model, TEST_QUERY,
TEST_DATA)
            print(f"    [✓] Cogrithm Done ({cog_result['latency']}s)")

            zs_result = run_zero_shot(front_vendor, front_model, TEST_QUERY,
TEST_DATA)
            print(f"    [✓] Zero-Shot Done ({zs_result['latency']}s)")

            # 2. Evaluate
            print(f"    -> Firing Tri-Judge Panel (Supreme Court)...")
            eval_data = evaluate_with_panel(TEST_QUERY, cog_result["output"],
zs_result["output"])

            # 3. Print
            print("-" * 80)
            print(f"⚖️ CONSENSUS: {eval_data['cog_votes']} Votes for Cogrithm |
{eval_data['zs_votes']} Votes for Zero-Shot")
            winner_str = "Challenger (Cogrithm)" if eval_data['cog_votes'] >
eval_data['zs_votes'] else "Incumbent"
            print(f"🏆 MATCH WINNER: {winner_str}\n")

            for judge in eval_data['panel_results']:
                print(f"    [{judge.get('judge_name')}] Voted: {judge.get('winner')}")
                print(f"    Rationale: {truncate(judge.get('rationale', 'N/A'), 120)}
\n")
            print("=" * 80)
            time.sleep(2)

if __name__ == "__main__":
    run_whitepaper_matrix()
```