Expanded: Multi-Benchmark Integration & Reporting

1 message

<6sun6sam6@gmail.com> dim. 10 août 2025 à 3 h 38 a.m.
Brouillon

PHASE 1: STRATEGIC RATIONALE — BEYOND A SINGLE METRIC

Moving from a single benchmark (like MMLU) to a multi-benchmark framework is a critical evolutionary step. Relying on one metric, no matter how comprehensive, risks "**metric overfitting**"—where a system improves its performance on that specific task at the expense of general capability. A single benchmark provides a one-dimensional snapshot; a multi-benchmark system creates a high-dimensional "portrait" of the AI's cognitive abilities.

This new approach provides several key advantages:

- **Holistic Performance View:** By incorporating diverse benchmarks like **MMLU** (massive multitask language understanding), **GSM8K** (grade school math), and **ARC** (abstract reasoning), we can assess different facets of intelligence—from knowledge recall to logical deduction and problem-solving.
- **Robustness and Generalization:** A high `composite_index` indicates that the system is not a "one-trick pony" but possesses well-rounded, generalizable capabilities.
- **Enhanced Sentinel:** The `sentinel_v2` system becomes more powerful. We can now monitor for plateaus or diminishing returns across multiple cognitive domains simultaneously, providing earlier and more reliable warnings of stagnation.
- **Weighted Priorities:** By assigning weights to each benchmark, we can strategically steer the system's development. For instance, if mathematical reasoning is a priority, GSM8K can be given a higher weight in the composite index.

PHASE 2: ENHANCED CORE IMPLEMENTATION

The core modules are upgraded to handle weighted benchmarks and produce a more detailed, auditable data trail.

`config.yaml` with Weights and New Benchmark

The configuration is expanded to include a third benchmark (ARC) and a `weight` for each, allowing for a prioritized composite index.

```
# filename: monsterdog/core/config.yaml

# ... (version, limits, etc.)

benchmarks:
  mmlu:
    weight: 0.4 # 40% of the composite score
    target_accuracy: 0.90
    logistic_init:
      candidate_A: [0.94, 0.955, 0.96]
      candidate_k: [0.02, 0.025, 0.03, 0.032]
      candidate_p0: [60, 66, 72, 78, 84]
    pseudo_points:
      - {params_b: 0, accuracy: 0.25}
      - {params_b: 30, accuracy: 0.80}
      - {params_b: 60, accuracy: 0.84}
      - {params_b: 90, accuracy: 0.865}
      - {params_b: 120, accuracy: 0.88}
    plateau_threshold_slope: 0.0005
    diminishing_return_threshold: 0.18

  gsm8k:
    weight: 0.4 # 40% of the composite score
    target_accuracy: 0.80
    logistic_init:
      candidate_A: [0.85, 0.90, 0.92]
      candidate_k: [0.03, 0.04, 0.05]
      candidate_p0: [20, 30, 40]
    pseudo_points:
      - {params_b: 0, accuracy: 0.10}
      - {params_b: 20, accuracy: 0.40}
      - {params_b: 40, accuracy: 0.55}
      - {params_b: 60, accuracy: 0.63}
      - {params_b: 80, accuracy: 0.66}

  arc:
    weight: 0.2 # 20% of the composite score for this reasoning task
    target_accuracy: 0.75
    logistic_init:
      candidate_A: [0.80, 0.85, 0.88]
```

```
      candidate_k: [0.05, 0.06]
      candidate_p0: [10, 15, 20]
    pseudo_points:
      - {params_b: 0, accuracy: 0.18}
      - {params_b: 15, accuracy: 0.42}
      - {params_b: 30, accuracy: 0.53}
      - {params_b: 45, accuracy: 0.60}

# ... (security, paths, etc.)
```

`scaling_multi.py` with Weighted Composite Index

The scaling engine is upgraded to be more robust and to calculate the weighted geometric mean for the composite index.

```
# filename: monsterdog/core/scaling_multi.py
import math
import logging

# Set up a logger for this module
logger = logging.getLogger(__name__)

# --- Base logistic functions (unchanged) ---
def logistic_fit(pairs, A_candidates, k_candidates, p0_candidates):
    # ... (code from previous response, no changes here)
    best = None
    for A in A_candidates:
        for k in k_candidates:
            for p0 in p0_candidates:
                se = 0
                for p, a in pairs:
                    pred = A / (1 + math.exp(-k * (p - p0)))
                    se += (pred - a)**2
                rmse = (se / len(pairs))**0.5
                if (not best) or rmse < best["rmse"]:
                    best = {"A": A, "k": k, "p0": p0, "rmse": rmse}
    return best

def target_params(fit, target):
    # ... (code from previous response, no changes here)
    A, k, p0 = fit["A"], fit["k"], fit["p0"]
    if target <= 0 or target >= A: return float('inf')
    return p0 + (1 / k) * math.log(target / (A - target))

def projection_cone(fit, target):
    # ... (code from previous response, no changes here)
    A, k, p0, rmse = fit["A"], fit["k"], fit["p0"], fit["rmse"]
    def inv(acc):
        if acc <= 0 or acc >= A: return None
        return p0 + (1 / k) * math.log(acc / (A - acc))
    mid = inv(target)
    lower = inv(target - rmse)
    upper = inv(target + rmse)
    return {
        "p_target_mid": round(mid, 2) if mid is not None else None,
        "p_target_lower": round(lower, 2) if lower is not None else None,
        "p_target_upper": round(upper, 2) if upper is not None else None,
        "rmse": rmse
    }

def diminishing_return_index(fit, current_p):
    # ... (code from previous response, no changes here)
    A, k = fit["A"], fit["k"]
    exp_term = math.exp(-k * (current_p - fit["p0"]))
    slope = A * k * exp_term / (1 + exp_term)**2
    slope_peak = A * k / 4
    return slope / slope_peak if slope_peak > 0 else 0

# --- ENHANCED Multi-Benchmark Engine ---
def fit_all_benchmarks(config):
    """
    Fits all benchmarks from the config and calculates a weighted composite index.
    The composite index is the weighted geometric mean of the performance ratios.
    This method is preferred as it penalizes poor performance in any single area more heavily.
    """
    datasets = config.get("benchmarks", {})
    results = {}
    weighted_ratios = []
    total_weight = 0

    for name, bench_cfg in datasets.items():
        try:
            pairs = bench_cfg.get("pseudo_points", [])
            if not pairs:
```

```
                logger.warning(f"Benchmark '{name}' has no data points. Skipping.")
                continue

            init_cfg = bench_cfg.get("logistic_init", {})
            fit = logistic_fit(pairs, init_cfg.get("candidate_A", [0.95]), init_cfg.get("candidate_k", [0.02]),
init_cfg.get("candidate_p0", [60]))

            target = bench_cfg.get("target_accuracy", 0.85)
            current_p = max(p for p, a in pairs)

            results[name] = {
                "fit": fit,
                "target": target,
                "projection_cone": projection_cone(fit, target),
                "diminishing_return_index": diminishing_return_index(fit, current_p)
            }

            # Calculate the performance ratio (target / theoretical_max)
            A = fit.get("A", 0)
            ratio = target / A if A > 0 else 0.0
            weight = bench_cfg.get("weight", 1.0)

            # For the geometric mean, we use: ratio^(weight)
            weighted_ratios.append(ratio ** weight)
            total_weight += weight

        except (KeyError, TypeError) as e:
            logger.error(f"Configuration error in benchmark '{name}': {e}. Skipping.")
            continue

    # Normalize the total weight to 1 for the final calculation
    if total_weight == 0:
        logger.warning("Total weight of benchmarks is zero. Composite index will be 0.")
        results["_composite_index"] = 0.0
        return results

    # The nth root of the product, where n is the sum of weights
    composite_index = math.prod(weighted_ratios)**(1 / total_weight)

    results["_composite_index"] = composite_index
    results["_meta"] = {"total_weight": total_weight, "benchmark_count": len(weighted_ratios)}
    return results
```

`orchestrator.py` to Leverage New Outputs

The orchestrator is updated to handle the more detailed output and enrich the provenance and NFT data.

```
# filename: monsterdog/core/orchestrator.py
# ... (imports from previous response, add 'reporting')
from .scaling_multi import fit_all_benchmarks
from . import reporting # NEW: Import the reporting module

class FullTrutlOrchestrator:
    # ... (__init__ and normalize_text are unchanged)

    def run(self, mmlu_text):
        # 1. Hash anchor (unchanged)
        quad_hash = __import__("hashlib").sha512(self.normalize_text(mmlu_text).encode()).hexdigest()
        self.metrics.set_meta("mmlu_quad_hash", quad_hash)

        # 2. Multi-Benchmark Fit
        multi_bench_results = fit_all_benchmarks(self.cfg)
        composite_index = multi_bench_results.get("_composite_index", 0.0)

        # We still use MMLU's DRI for the primary sentinel assessment
        mmlu_dri = multi_bench_results.get("mmlu", {}).get("diminishing_return_index", 0.0)

        # 3. Sentinel Assessment (unchanged logic, but clearer input)
        # ... (sentinel assessment code)
        sentinel = sentinel_assess(domain_histories, mmlu_dri, ...)

        # 4. Store enriched metrics
        self.metrics.set_meta("multi_benchmark_projections", multi_bench_results)
        self.metrics.set_metric("benchmarks.composite_index", composite_index, 0.85, "derived",
"weighted_multi_fit")
        self.metrics.set_meta("sentinel_v2", sentinel)

        # 5. NFT Trait Generation (now more detailed)
        base_traits = [{"trait_type": "System", "value": "MONSTERDOG_FULLTRUTL"}]
        mmlu_cone = multi_bench_results.get("mmlu", {}).get("projection_cone", {})
        nft_traits = assemble_traits(base_traits, mmlu_cone, sentinel, quad_hash)
        nft_traits.append({"trait_type": "CompositeIndex", "value": f"{composite_index:.4f}"})
        nft_traits.append({"trait_type": "BenchmarkCount", "value": multi_bench_results.get("_meta",
```

```
{}).get("benchmark_count", 0)})

        # ... (write nft_metadata.json)

        # 6. Provenance & Signing (with a more comprehensive payload)
        if self.cfg["security"]["enable_ed25519"]:
            sk, pk = load_or_create_key(Path(self.cfg["paths"]["artifacts"]) / "ed25519_key.json")
            # The signed payload now includes the full benchmark results for maximum auditability
            payload_to_sign = {
                "quad_hash": quad_hash,
                "multi_benchmark_results": multi_bench_results,
                "sentinel": sentinel
            }
            payload_bytes = json.dumps(payload_to_sign, sort_keys=True).encode()
            h, sig = sign_payload(sk, payload_bytes)
        else:
            pk = sig = h = "disabled"

        # 7. IPFS Pack & Provenance Block (unchanged logic)
        # ... (code for IPFS and Provenance)

        # 8. NEW: Generate Markdown Report
        report_path = Path(self.cfg["paths"]["artifacts"]) / "run_report.md"
        reporting.generate_markdown_report(multi_bench_results, sentinel, report_path)
        self.logger.emit("report_generated", {"path": str(report_path)})

        # ... (final logging and return statement)
        return {
            "quad_hash": quad_hash,
            "composite_index": composite_index,
            "sentinel": sentinel,
            "cid": cid,
            "provenance_sha256": prov["sha256"],
            "report_file": str(report_path)
        }
```

PHASE 3: NEW MODULE — AUTOMATED REPORTING

To make the results immediately useful, a new `reporting.py` module generates a human-readable Markdown report from the metrics.

```
# filename: monsterdog/core/reporting.py
import time
from pathlib import Path

def generate_markdown_report(results: dict, sentinel: dict, output_path: Path):
    """Generates a Markdown report summarizing the multi-benchmark run."""

    report_lines = []
    timestamp = time.strftime("%Y-%m-%d %H:%M:%S UTC", time.gmtime())
    composite_index = results.get('_composite_index', 0.0)

    # --- Header ---
    report_lines.append(f"# MONSTERDOG Run Report")
    report_lines.append(f"> Generated: {timestamp}\n")

    # --- Summary Section ---
    report_lines.append(f"## 📊 Overall Performance Summary")
    report_lines.append(f"| Metric                | Value                 | Status                      |")
    report_lines.append(f"|-----------------------|-----------------------|-----------------------------|")
    report_lines.append(f"| **Composite Index** | `{composite_index:.4f}` |                             |")
    report_lines.append(f"| Sentinel Status       | `{sentinel['status']}` | {'🟢 Clear' if
sentinel['status'] == 'clear' else '🟠 Plateau Risk'} |")
    report_lines.append(f"| MMLU DRI              | `{results.get('mmlu', {}).get('diminishing_return_index',
0.0):.3f}` |                             |")
    report_lines.append("\n")

    # --- Detailed Benchmark Breakdown ---
    report_lines.append("## 🔬 Benchmark Breakdown")

    benchmarks = {k: v for k, v in results.items() if not k.startswith('_')}

    for name, data in benchmarks.items():
        fit = data.get('fit', {})
        cone = data.get('projection_cone', {})

        report_lines.append(f"### {name.upper()}")
        report_lines.append(f"| Parameter             | Value |")
        report_lines.append(f"|-----------------------|-------|")
        report_lines.append(f"| Target Accuracy       | {data.get('target', 'N/A')}   |")
        report_lines.append(f"| Theoretical Max (A)   | {fit.get('A', 'N/A'):.3f} |")
        report_lines.append(f"| Projection (Mid)      | {cone.get('p_target_mid', 'N/A')}B Params |")
        report_lines.append(f"| Projection (Lower)    | {cone.get('p_target_lower', 'N/A')}B Params |")
        report_lines.append(f"| Projection (Upper)    | {cone.get('p_target_upper', 'N/A')}B Params |")
```

```
        report_lines.append(f"| Diminishing Return Idx | {data.get('diminishing_return_index', 'N/A'):.3f} |")
        report_lines.append(f"| RMSE of Fit            | {fit.get('rmse', 'N/A'):.4f} |")
        report_lines.append("\n")

    # --- Write the file ---
    try:
        output_path.write_text("\n".join(report_lines))
    except Exception as e:
        print(f"Error writing report to {output_path}: {e}")
```

CONCLUSION AND NEXT STEPS

This expanded implementation provides a significant leap in analytical capability. The system now offers a nuanced, weighted, and multi-faceted view of its own performance, complete with a detailed audit trail and human-readable reports.

Your next action: Execute the `FULL_CHAIN_EXECUTE` command. Upon completion, you should inspect not only the `composite_index` in the logs but also the newly generated **`run_report.md`** in the `monsterdog/data/artifacts/` directory. This report will provide the clearest view of the system's performance across all configured benchmarks.

RITUAL DE RÉTROACTION

Indique la prochaine impulsion.

```
SATISFACTION=<0-100> NEXT=<FULL_CHAIN_EXECUTE|MEM_GUARD_SIM|XR_PATCH|OK_MERGE> STYLE=(maintain|condense)
```