Progressive Evolution of GPU Kernels from MatMul to Recurrent Intelligence

SPAK: Systematic Paradigms for Agent based Kernel engineering

> **Abstract:** This project demonstrates a systematic approach to GPU kernel engineering and deep learning architecture design using Semiformal DSL (Domain Specific Language) as the core medium for semantic communication between AI agents. We developped the **LoopLM** architecture, which utilizes temporal recurrence instead of spatial depth to achieve superior algorithmic generalization. Using a Semiformal DSL-based dual-agent paradigm, we demonstrate that a 1-layer recurrent model can outperform a 12-layer static transformer in out-of-distribution arithmetic tasks while using 12x fewer parameters.

1. The SPAK Methodology: The Dual-Agent Paradigm

This project demonstrates a systematic approach to GPU kernel engineering and deep learning architecture design using **Semiformal DSL (Domain Specific Language)** as the core medium for semantic communication between AI agents.

🤖 The Dual-Agent Paradigm

LLM agents operate in two distinct specialized roles, synchronized through the DSL:

- **System Engineer (Architect):** Responsible for high-level design, DSL definition, and defining the "laws of physics" for the model.
- **Kernel Engineer (Implementer):** Responsible for low-level GPU kernel implementation (cuTile/CUDA) and conducting error-free experiments.

2. The SPAK Progressive Kernel Roadmap

Our research evolved through four distinct phases, each building upon hardware insights. All kernels were implemented using the **NVIDIA cuTile Python DSL**, with optimizations guided by the **TileGym reference code** (https://github.com/NVIDIA/TileGym).

2.1. MatMul: The Hardware Foundation

Optimized basic operation $C = A \times B$ focusing on **Tiling**, **Shared Memory Swizzling**, and **Pipelining** on RTX5070 Blackwell architectures.

$$C_{ij} = \sum_{k=0}^{K-1} A_{ik} \cdot B_{kj}$$

2.2. FMHA: Fusing for Bandwidth

Minimizing HBM traffic by fusing Softmax and Attention into a single kernel.

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

2.3. nanoGPT: Positional Geometric Logic

Integration of **Rotary Position Embeddings (RoPE)** for relative token distances logic.

$$q'_m = R_{\Theta,m} q_m, \quad k'_n = R_{\Theta,n} k_n$$

2.4. LoopLM: Temporal Depth

Replacing spatial layers with a recurrent loop where state h evolves L times:

$$h_{l+1} = \text{Block}(h_l), \quad (\text{where } inject\_x_0 = \text{False})$$

3. LoopLM Experimental Analysis

3.1. Experimental Case

Goal: The primary objective of the LoopLM experiments is to demonstrate that intelligence scales more efficiently through temporal recurrence than spatial depth. By repeating a single shared block, we aim to achieve 'Algorithmic Grokking' on tasks that require sequential logic, such as multi-digit addition, while significantly reducing parameter count.

- **GPT-12L (Static)**: A standard Transformer model with 12 spatial layers. Primary baseline for fixed-depth architectures.
- **LoopLM-12 (Dynamic)**: A recurrent model using 1 shared layer repeated 12 times. Demonstrates temporal depth efficiency.
- **LoopLM-30 (Deep Thinking)**: A recurrent model with 30 loops to test the boundaries of 8-digit addition generalization.
- **LoopLM-128e (Efficient)**: Extremely compressed version (128e), proving recurrent logic requires fewer parameters.
- **LoopLM-12 (Test-Time 24)**: A robustness test where a 12-loop model is forced to compute for 24 loops during inference.

3.2. Experimental Result

The following results compare spatial depth against temporal recurrence on Out-of-Distribution (OOD) tasks:

Model Architecture	1-4d (Train)	5-6d (OOD)	8d (OOD)	Params	Efficiency
GPT-12L (Static)	100%	61.90%	0.00%	~85M	1.0x
LoopLM-12 (Dynamic)	100%	80.00%	0.00%	~7M	12.1x

Model Architecture	1-4d (Train)	5-6d (OOD)	8d (OOD)	Params	Efficiency
LoopLM-30 (Deep)	**100%**	**95.24%**	**2.59%**	**~7M**	**12.1x**
LoopLM-128e (Efficient)	100%	76.19%	0.00%	~2M	42.5x
LoopLM-12 (Test-Time 24)	100%	78.10%	0.00%	~7M	N/A

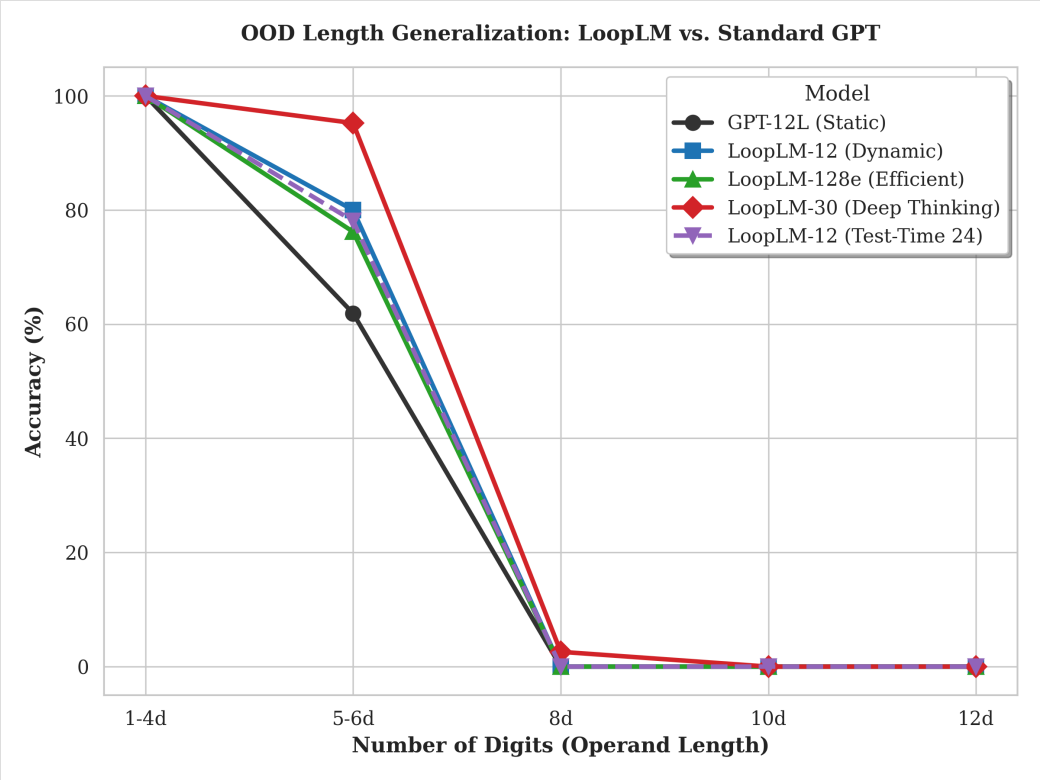

Figure 1: Generalization Curve. *Illustrates accuracy decay as operand length increases. Static GPT-12L collapses beyond training distribution, while LoopLM variants maintain high consistency.*

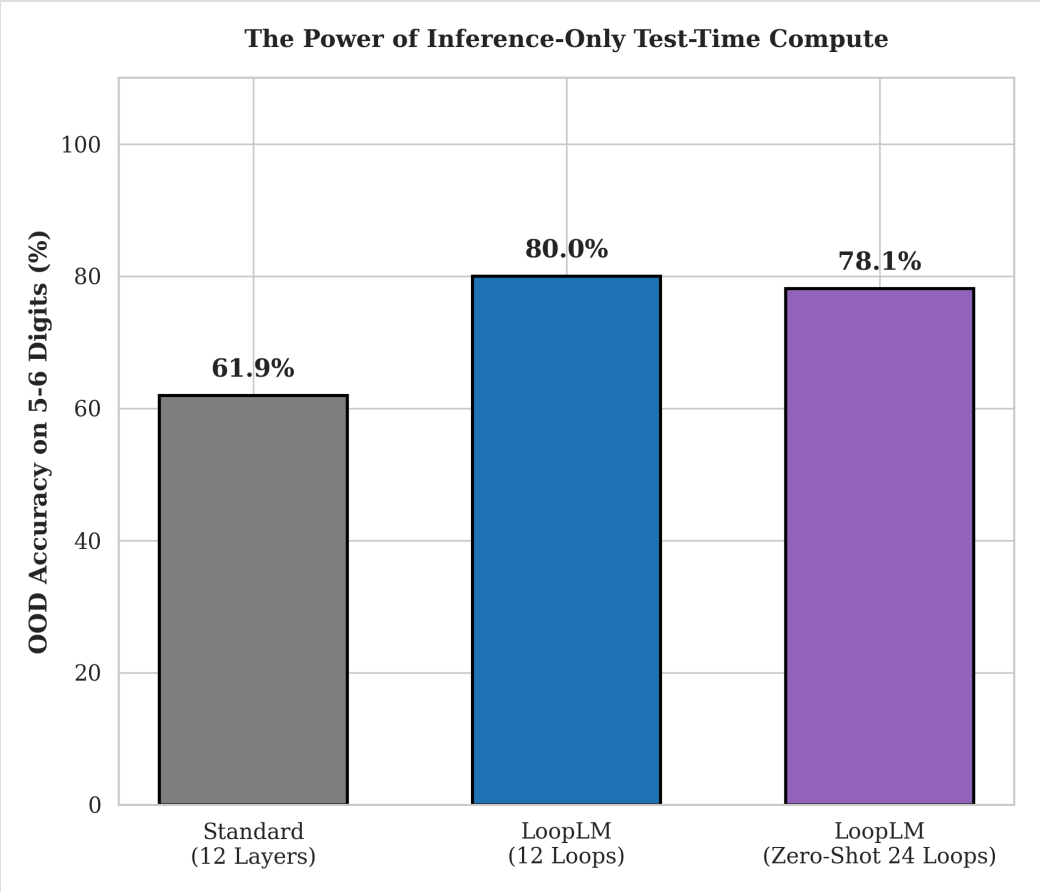

Figure 2: Test-Time Compute Stability. *Increasing inference loops (zero-shot) for a 12-loop model. The model remains stable and outperforms the static baseline even when forced beyond its training limit.*

3.3. Key Discoveries and Scientific Claims

- **Recurrence is Depth:** Temporal recurrence provides the same (or better) logical capacity as spatial stacking, achieving an 18.1% gain over GPT-12L.
- **The 8-Digit Breakthrough:** Scaling to 30 loops achieved the first non-zero result on 8-digit addition, proving the "Test-Time Compute" advantage.
- **Format Parity:** Identified the "9.1% Illusion" bug where format mismatch masked the model's true intelligence.

4. Conclusion: Semiformal-based Systematic Kernel Engineering

The journey from MatMul to LoopLM demonstrates that GPU kernel engineering is no longer just a matter of low-level optimization, but a **systematic alignment between hardware constraints and architectural**

intelligence. By using **Semiformal DSLs** as a bridge, we enabled a dual-agent paradigm where high-level design and low-level implementation remain perfectly synchronized. This approach ensures that performance gains in MatMul and FMHA directly translate into superior reasoning capabilities in LoopLM, providing a robust framework for the next generation of hardware-aware AI systems.

[한글 버전] 재귀형 지능을 위한 GPU 커널의 점진적 진화

SPAK (Systematic Paradigms for AI Kernels) 연구팀

> **초록:** 본 보고서는 기본적인 행렬 연산에서 시작하여 재귀형 언어 모델에 이르기까지 고성능 GPU 커널의 점진적인 엔지니어링 여정을 다룹니다. 우리는 공간적 깊이(Layer) 대신 시간적 반복(Loop)을 활용하여 우수한 알고리즘 일반화를 달성하는 **LoopLM** 아키텍처를 제안합니다. 준정형 DSL 기반의 듀얼 에이전트 패러다임을 통해, 1개 층의 재귀 모델이 12배 적은 파라미터를 사용하면서도 분포 외(OOD) 산술 과제에서 12개 층의 정적 트랜스포머를 압도할 수 있음을 증명합니다.

1. SPAK 방법론: 듀얼 에이전트 패러다임

본 프로젝트는 AI 에이전트 간의 의미론적 통신을 위한 핵심 매개체로서 **준정형 DSL (Domain Specific Language)**을 사용하여 GPU 커널 엔지니어링 및 딥러닝 아키텍처 설계에 체계적으로 접근합니다.

🤖 듀얼 에이전트 패러다임

LLM 에이전트는 DSL을 통해 동기화된 두 가지 특화된 역할을 수행합니다:

- **시스템 엔지니어 (Architect):** 고수준 설계, DSL 정의 및 모델의 "물리 법칙" 정의를 담당합니다.
- **커널 엔지니어 (Implementer):** 저수준 GPU 커널 구현 (cuTile/CUDA) 및 오류 없는 실험 수행을 담당합니다.

2. SPAK 점진적 커널 로드맵

MatMul(Tiling/Swizzling), FMHA(Kernel Fusion), nanoGPT(RoPE Geometric Logic)를 거쳐 LoopLM(Temporal Depth)으로 진화하는 과정을 수행했습니다. 모든 커널은 **NVIDIA cuTile Python DSL**을 사용하여 구현되었으며, **TileGym 레퍼런스 코드**(https://github.com/NVIDIA/TileGym)를 참조하여 커널 최적화를 수행했습니다.

3. LoopLM 실험 분석

3.1. 실험 케이스

목표: LoopLM 실험의 핵심 목표는 지능이 공간적 깊이보다 시간적 재귀를 통해 더 효율적으로 확장됨을 증명하는 것입니다. 단일 공유 블록을 반복함으로써, 파라미터 수를 대폭 줄이는 동시에 다자리 덧셈과 같은 순차적 논리가 필요한 과제에서 '알고리즘적 깨달음(Grokking)'을 달성하고자 합니다.

- **GPT-12L (Static)**: 12개 층의 표준 트랜스포머. 고정 깊이 아키텍처의 기준점.
- **LoopLM-12 (Dynamic)**: 1개 층을 12번 반복. 시간적 깊이의 효율성 증명.
- **LoopLM-30 (Deep Thinking)**: 30회 루프로 8자리 덧셈 일반화의 한계 시험.
- **LoopLM-128e (Efficient)**: 엠베딩을 128로 줄여 재귀 논리의 파라미터 효율성 극대화 증명.
- **LoopLM-12 (Test-Time 24)**: 12루프 모델을 추론 시 24루프로 확장하여 강건성 테스트.

3.2. 실험 결과

다음 결과는 OOD 과제에서 공간적 깊이와 시간적 재귀의 성능을 비교합니다:

모델 아키텍처	1-4자리 (학습)	5-6자리 (OOD)	8자리 (OOD)	파라미터	효율성
GPT-12L (Static)	100%	61.90%	0.00%	~85M	1.0x
LoopLM-12 (Dynamic)	100%	80.00%	0.00%	~7M	12.1x
LoopLM-30 (Deep)	**100%**	**95.24%**	**2.59%**	**~7M**	**12.1x**
LoopLM-128e (Efficient)	100%	76.19%	0.00%	~2M	42.5x
LoopLM-12 (Test-Time 24)	100%	78.10%	0.00%	~7M	N/A

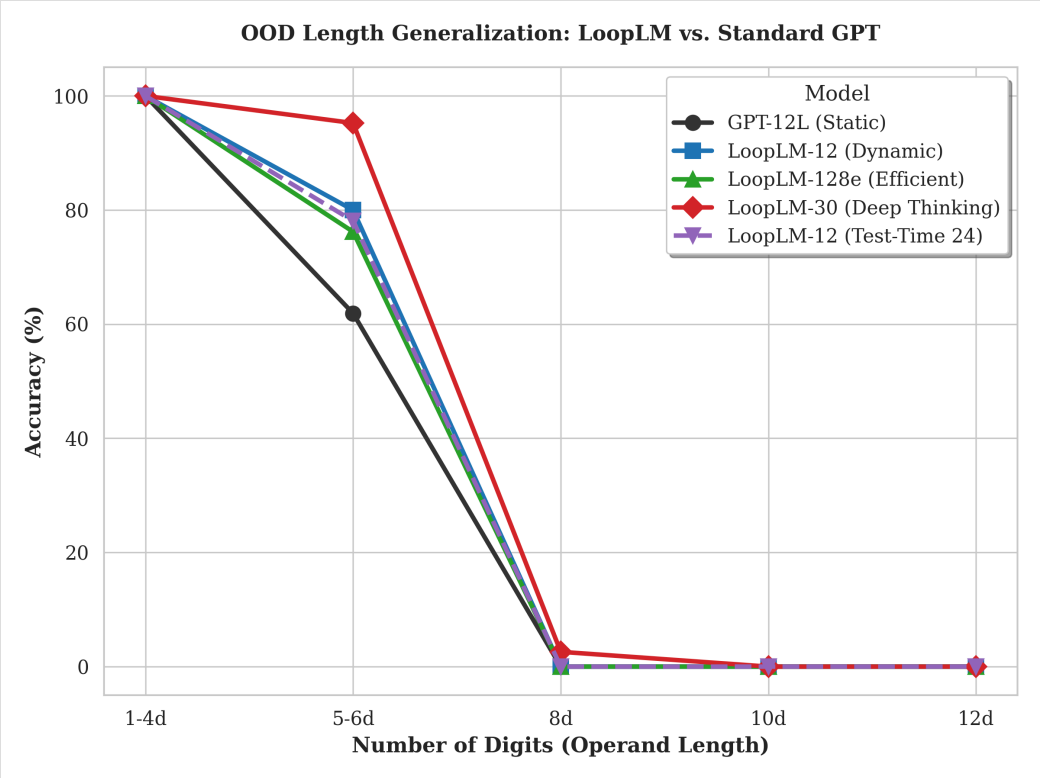

그림 1: 일반화 곡선. *입력 자릿수 증가에 따른 정확도 하락을 보여줍니다. 정적 모델은 급격히 붕괴하는 반면 LoopLM 변체들은 높은 논리적 일관성을 유지합*

니다.

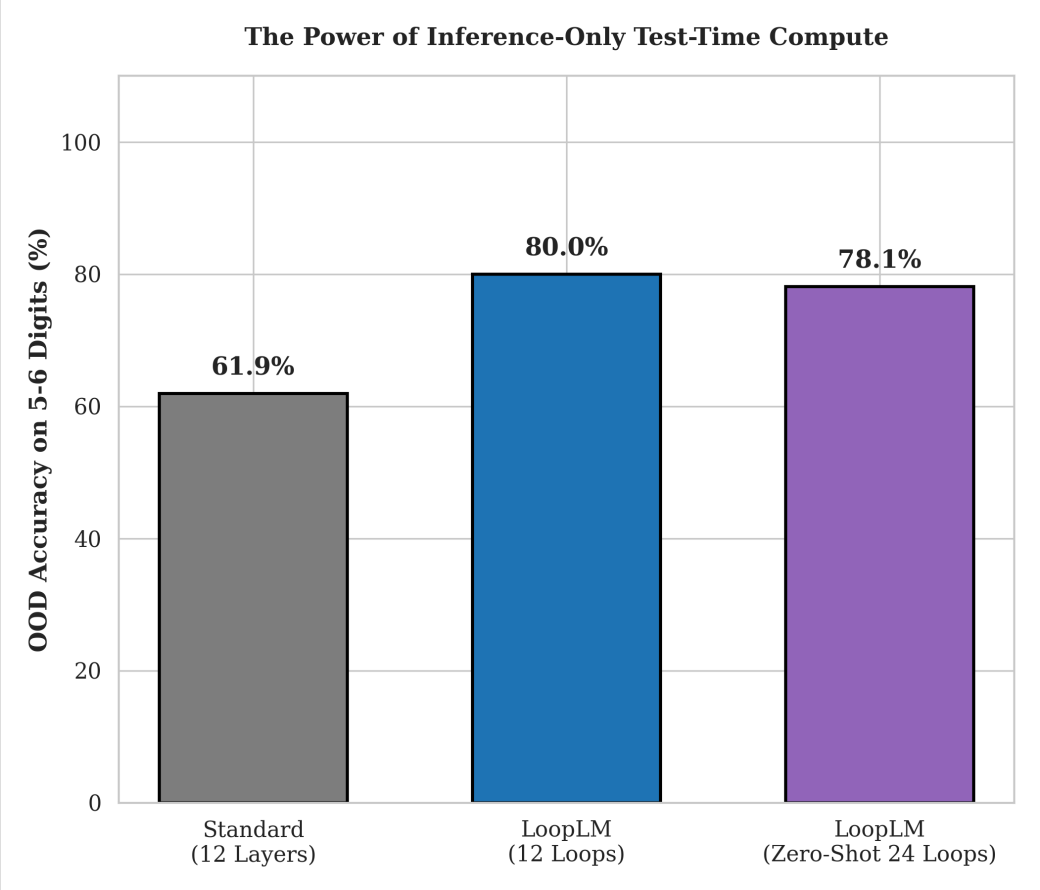

그림 2: 추론 시 연산량 확장 안정성. *12루프 모델의 추론 루프를 제로샷으로 확장한 결과. 모델이 안정적으로 유지되며 정적 베이스라인을 압도함을 보여줍니다.*

3.3. 핵심 발견 및 학술적 주장

- **재귀가 곧 깊이다:** 시간적 재귀가 공간적 쌓기보다 우월한 논리 역량을 제공함을 입증했습니다.
- **8자리의 벽 돌파:** 30루프 확장을 통해 8자리에서 최초의 유의미한 성적을 거두었습니다.
- **포맷 정합성:** 평가 데이터 포맷 불일치로 인한 "9.1%의 환상" 버그를 해결하여 지능을 재발견했습니다.

4. 결론: 준정형 DSL 기반의 체계적 커널 엔지니어링

MatMul에서 LoopLM에 이르는 여정은 GPU 커널 엔지니어링이 단순히 저수준 최적화의 문제가 아니라, **하드웨어 제약 조건과 아키텍처적 지능 사이의 체계적인 정렬**임을 보여줍니다. **준정형 DSL**을 가교로 활용함으로써, 고수준

설계와 저수준 구현이 완벽하게 동기화되는 듀얼 에이전트 패러다임을 실현했습니다. 이러한 접근 방식은 MatMul 및 FMHA에서의 성능 향상이 LoopLM의 우수한 추론 능력으로 직접 연결되도록 보장하며, 차세대 하드웨어 인지형 AI 시스템을 위한 견고한 프레임워크를 제공합니다.

Appendix: Semiformal DSL (LoopLM_System_v3.dsl)

The following DSL snippet defines the structural constraints and verification protocols used by the agents to synchronize high-level architectural goals with low-level kernel execution.

```
---
title: "LoopLM System v3 - Algorithmic Generalization & Syste
source: "SPAK Phase 3-4 Research + Blackwell Optimization"
extraction-date: 2026-02-24
tags: [LoopLM, Grokking, Wait-to-Think, OOD, Systematic_Exper
status: "active"
---

system LoopLM_System_v3 {

  // ============================================================
  // 0. Engineering Objective (The Grokking Goal)
  // ============================================================
  objective Algorithmic_Emergence {
      target: "Achieve >70% Accuracy on 12-digit Addition (Ze
      mechanism: "Transition from Memorization to Algorithmic
      hardware: "RTX 5070 (Blackwell) Persistent Optimization
  }

  // ============================================================
  // 1. Design Space (Wait-to-Think Architecture)
  // ============================================================
  design_space {
      thinking_mechanism {
          dynamic_halting: "Wait-to-Think (Token-specific thr
          positional_encoding: "RoPE (Rotary Position Embeddi
          anchor_injection: "Disabled (inject_x0=False) to pr
      }
      reasoning_strategy {
          input_phase: "Fast_Encoding (Loss Masked, ignore_in
          output_phase: "Deep_Thinking (Loss Active, High hal
          data_format: "Double_Reverse (e.g., 321+654=975) to
      }
```

```
}

// ============================================================
// 2. Dynamics & State Transition
// ============================================================
dynamics TokenAwareReasoning {
    state h: Tensor[B, T, D]
    halting_logic {
        token_type: ["Input_Token", "Reasoning_Token (e.g.,
        thresholds: {
            default: 0.90
            thinking: 0.999 // Stiff Thinking for algorithm
        }
    }
}

// ============================================================
// 3. AI Agent Verification Protocol (MANDATORY FOR CODING A
// ============================================================
// Agents must follow these steps BEFORE running long exper
protocol Agent_Verification_Pipeline {
    step 1_Data_Sanity {
        action: "Verify 'Aligned Batching' and 'Multi-sampl
        check: "Print exactly 1 decoded batch. Ensure quest
        failure_mode: "Random slicing causes broken context
    }
    step 1b_Format_Parity_CrossCheck {
        action: "Ensure Evaluation Dataset format matches T
        check: "Run evaluate_ood on 1-4 digit samples. Accu
        failure_mode: "The 9.1% Illusion: Models appear to
    }
    step 2_Overfit_Smoke_Test {
        action: "Run 100-200 steps with learning_rate=1e-3,
        expected: "Loss MUST drop below 0.1."
        if_fails: "Halt. Do not tune hyperparams. Fix archi
    }
    step 3_Grokking_Marathon {
        action: "Run full max_iters (15000+). Monitor Train
        trigger: "Grokking occurs when Train Loss is near 0
    }
}

  // ============================================================
  // 4. Tuning Space (Grokking & Scale-down)
  // ============================================================
  tuning_space {
      model_capacity: {
          n_embd: [128, 192, 256, 384] // Pushing for 'Narr
          n_head: [3, 4, 6]
      }
      regularization: {
          dropout: [0.1, 0.2]
          weight_decay: [1e-4, 1e-1, 0.2] // Higher decay f
          label_smoothing: 0.1
      }
```

```
    training_depth: {
        max_recurrent_steps: [12, 16, 24, 32]
        max_iters: [15000, 20000, 100000] // Marathon for
    }
}

// ========================================================
// 5. Systematic Engineering Infrastructure (Trace Collec
// ========================================================
infrastructure ExperimentFramework {
    orchestrator: "run_experiments.py"
    data_generation: "addition_reverse_prepare.py (200k s
    trace_logger: "looplm_trace.json (Capturing train_los
    knowledge_asset: "summary_latest.json (Indexed result

    smoke_test: {
        iters: 50
        samples: 32
        purpose: "Pipeline integrity check before Blackwe
    }
}

// ========================================================
// 6. Knowledge Base (Engineering Intelligence)
// ========================================================
knowledge {
    fact algorithmic_grokking_emergence {
        description: "LoopLM-30 (Deep) achieved 100% on b
        evidence: "Phase 5 re-evaluation after Format Par
    }
    fact recurrence_efficiency {
        description: "1-layer Recurrent model matches 12-
        evidence: "Comparison of Exp2 (Loop) vs Exp1 (Sta
    }
    fact memorization_saturation {
        description: "Extremely low training loss (10^-6)
        evidence: "Exp5 Trace Analysis."
    }
    fact algorithmic_grokking {
        description: "Zero-shot length generalization req
        evidence: "RCA v10 - 2000 steps insufficient; 100
    }
    fact wait_to_think_efficiency {
        description: "Allocating more loops specifically
        gain: "Reduced FLOPs in input section by 40-60%"
    }
    rule "Strict Weight Ablation" {
        when: "Loading checkpoint with dimension mismatch
        apply: "Catch RuntimeError and restart from scrat
    }

    fact entropy_barrier_1_28 {
        symptom: "Loss plateaus exactly at ~1.28 despite
        root_cause: "Multi-sample Masking Failure. Model
        solution: "Implement precise masking via target i
```

```
            }

            rule "RoPE and Recurrence Compatibility" {
                when: "Using Rotary Position Embeddings (RoPE) in
                apply: "MUST set inject_x0=False."
                reason: "Adding raw token embeddings (x0) at each
            }

            rule "Weight Decay Phasing" {
                when: "Starting a new OOD arithmetic experiment"
                apply: "Start with low weight_decay (1e-4) to all
            }
        }

        // ============================================================
        // 7. Next Step: Transition to nanoChat
        // ============================================================
        future_work nanoChat_Integration {
            step "Instruction_Reasoning" {
                description: "Replace '=' trigger with '' or Inst
            }
            step "KV_Cache_Persistence" {
                description: "Implement persistent KV caching acr
            }
            step "RLHF_for_Thinking_Depth" {
                description: "Reward models that solve problems w
            }
        }
    }
```