

KD Agentic Execution Stack

LiteLLM Configuration, Tool Calling, Harness Optimization and DeepSWE Evaluation

Technical Whitepaper and Proposal

Author: **ctaxnagomi | Olympic Dancer 1963**

Kracked Devs / KD AI Infrastructure

Version 1.0 • 16 September 2026

Abstract. This paper proposes a controlled evaluation of whether an agentic configuration and harness can materially improve the practical software-engineering performance of the same underlying LLM when served through a LiteLLM-based gateway. The proposal separates model capability from gateway exposure and agent-harness capability. DeepSWE is proposed as an external software-engineering benchmark, while a smaller internal A/B suite is recommended first. All numerical DeepSWE results in this document are explicitly **simulated expected results**, not measured results.

Item	Proposal
Primary hypothesis	Agentic orchestration can improve effective task completion and cost efficiency without changing model weights.
Model	DeepSeek V4 Flash; exact dated identifier must be pinned at evaluation time.
Gateway	KD-managed LiteLLM deployment.
External benchmark	DeepSWE v1.1: 113 long-horizon software-engineering tasks.
Core comparison	KD Default vs Tool-Enabled vs Universal/Hybrid Person B configuration.

1. Problem Statement

KD currently appears to optimize primarily for convenient model access. Advanced developer workflows, however, depend on tool calling, structured tool results, MCP or service routing, workspace-specific skills, context reduction and multi-step execution. LiteLLM is designed as a centralized gateway supporting multiple providers, routing, retries/fallbacks, authentication and authorization, project customization, spend tracking and budgets.

The central question is not whether DeepSeek V4 Flash is capable in isolation. It is whether the **effective capability exposed to a KD user** changes materially when the same model is provided with a better execution configuration and agentic harness.

Conceptual capability model:

```
Effective Agent Capability
≈ Model Capability
× Context Quality
× Tool Availability
× Tool Schema Quality
× Routing
× State Management
× Harness / Orchestration
× Permission Model
× Execution Environment
× Evaluation
```

2. Observed KD Findings

An A/B configuration test manually enabled tool calling for the existing proxy. The observed response returned HTTP 200 and a message structure containing content, role, tool_calls and provider_specific_fields, with hasToolCalls=True. This establishes that the current proxy can pass through tool-call behavior when appropriately configured.

A separate observation involved a false-positive or policy-triggered response when a developer attempted to create a project documentation file named RULESET.md. Equivalent semantic instructions using neutral wording or a different file extension sometimes behaved differently. This suggests a policy/filtering layer may be affecting lexical requests, but does not prove that the mechanism is a simple regular expression.

The security concern is therefore not that guardrails exist. The concern is whether broad lexical filtering is being used as the principal security boundary. A scalable agentic architecture should enforce permissions at the execution boundary.

3. Hypotheses

H1: For the same model and task set, approved tool calling, workspace tools, controlled routing, context reduction and a purpose-built harness will improve software-engineering task success and/or reduce cost per successful task.

H2: Tool calling alone will improve operational capability, but the largest gains will require orchestration, state, validation, routing and context management.

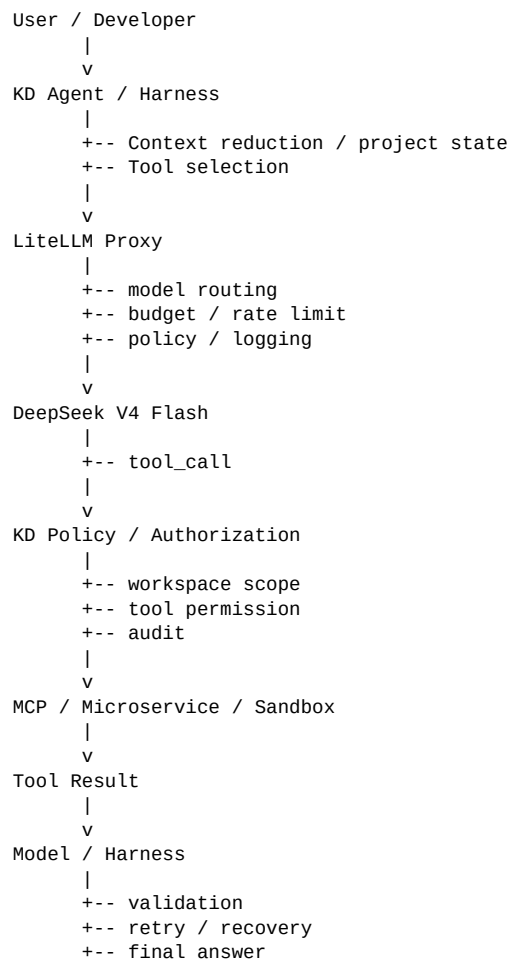
H3: Security can be strengthened while capability is increased if authorization and isolation are enforced at tool execution rather than through broad lexical blocking.

4. Experimental Configurations

Config	Description	Purpose
A. Normal	Current KD default model/proxy configuration.	Baseline convenience.
B. Standard Agentic	A + approved tool definitions + tool_choice + tool execution loop.	Isolate tool-calling contribution.
C. Universal / Hybrid	B + workspace skills + MCP/service routing + context reduction + harness + validation	Measure full agentic orchestration.
D. Upstream reference	Direct upstream endpoint with the same pinned model/settings.	Detect gateway transformations.

Keep the model snapshot, task, repository, sampling policy, token budget, network policy and evaluation criteria fixed wherever possible. Configuration is the independent variable.

5. Proposed Agentic Execution Architecture



The proposal does not recommend removing guardrails. It recommends layered controls: provider safety, gateway policy, workspace authorization, tool permissions, isolated execution, audit logging and human approval for high-impact operations.

6. DeepSWE Evaluation Methodology

DeepSWE is designed to measure frontier coding agents on original, long-horizon software-engineering tasks from active open-source repositories. Its current repository describes 113 tasks across TypeScript, Go, Python, JavaScript and Rust, with isolated environments and program-based verifiers. Tasks contain metadata, instructions, reproducible environments and held-out verification. The verifier grades observable behavior rather than requiring a particular internal implementation.

Recommended sequence:

- Phase 0: Freeze model ID, proxy version, configuration, sampling and token accounting.
- Phase 1: Run 10 deterministic tasks as a smoke test.
- Phase 2: Run 30–50 tasks for a controlled comparison.
- Phase 3: Run the complete 113-task suite for final baseline and experimental configurations.
- Phase 4: Publish success, cost, token, latency, tool and failure metrics.

7. Metrics

Metric	Definition
Pass@1 / reward	Fraction of tasks fully verified as correct.
Partial score	Verifier-derived partial completion where available.
Cost / task	Actual model/API spend per task.
Tokens / task	Input + output tokens, separating reasoning/tool tokens when available.

Metric	Definition
Agent steps	Number of model/tool execution steps.
Tool success	Successful tool calls / attempted tool calls.
Human intervention	Manual corrections or decisions.
Latency	Wall-clock task duration.
Failure taxonomy	Model, tool, routing, policy, environment and timeout failures.

8. Simulated DeepSWE Expected Results

WARNING: SIMULATION ONLY. These values are planning hypotheses, not benchmark measurements. They must not be presented publicly as actual DeepSWE results.

Config	Simulated Pass@1	Simulated avg cost/task	Simulated tokens/task	Expected interpretation
A. Normal	45–55%	\$0.70–\$1.00	170K–220K	Convenient baseline; limited agentic surface.
B. Standard Agentic	52–62%	\$0.60–\$0.90	145K–190K	Tool calls improve repository interaction.
C. Universal / Hybrid	60–70%	\$0.45–\$0.75	105K–155K	Harness, routing and context reduction improve execution.

Illustrative midpoint scenario: A = 50% Pass@1, B = 57%, C = 65%. This would represent a hypothetical +15 percentage-point improvement from the default configuration to the Universal configuration. It is a test target, not a prediction.

9. User-End vs Owner-End Expected Results

The experiment has two stakeholders. The user wants higher quality with fewer tokens and less manual correction. The LiteLLM/KD owner needs security, predictable cost, governance and a reproducible product configuration.

Dimension	User / Member	LiteLLM / KD Owner
Quality	Higher first-pass completion; better workspace-tool usage.	Evidence of improved product capability.
Cost	Fewer redundant tokens and failed runs.	Lower cost per verified successful task.
Tools	MCP, DOM/context tools and skills become available.	Controlled catalogue with explicit permissions.
Security	Fewer harmless-workflow false positives.	Capability-based restrictions and audit.
Marketing	More capable KD development environment.	Evidence-backed agentic execution positioning.
Governance	Predictable workspace behavior.	Centralized routing, budgets, logs and policy.

10. Business Model

The defensible commercial asset is not a claim that KD created a new base model. It is the **execution layer around models**: configuration, routing, tools, context optimization, governance, evaluation and workspace integration.

Tier	Example model	Value
KD Standard	Free / community	Convenient approved model access.
KD Agentic	Paid monthly or credit-based	Tool calling, MCP and workflow execution.
KD Universal	Higher tier / team plan	Hybrid harness, context optimization, routing and skills.
KD Enterprise	Contract / private deployment	Isolation, audit, custom MCP and enterprise policies.
KD Optimization	Professional service	Tune gateway and agent stack against fixed tasks.

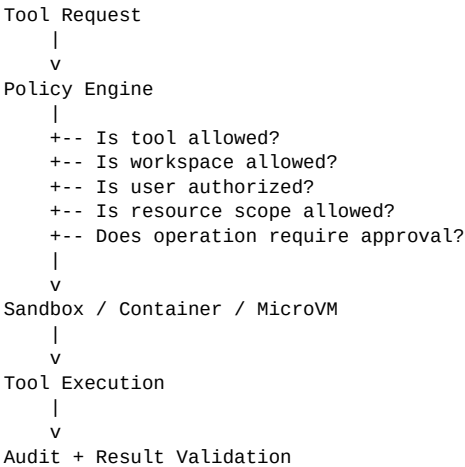
Illustrative economics: if a member has \$20 in credits and the Universal configuration reduces successful-task cost by 30%, the same credit pool could support about 1.43 times as many successful tasks, assuming quality remains constant. This is an example, not a forecast.

11. Comparison With an Unsloth-Style Positioning

Approach	What is measured	KD analogue
Base model comparison	Model-to-model capability.	Not the primary KD claim.
Implementation optimization	Same model, different implementation, speed/efficiency.	Same model, different agentic configuration.
Harness optimization	Same model/benchmark, different agent execution layer.	KD Default vs Universal.
Product claim	Measured improvement under declared conditions.	Verified task success per dollar.

Unsloth has published implementation-level benchmark comparisons against baseline implementations. The lesson for KD is not to copy its product, but to adopt the discipline: define a baseline, disclose settings, benchmark the implementation and publish measurable deltas.

12. Security and Isolation



A lexical filter can remain a cheap first-line signal, but it should not be the only security boundary. The execution layer should make unauthorized operations impossible or explicitly deny them.

13. Rollout Plan

Stage	Deliverable	Owner
0	Freeze model, proxy, config and benchmark environment.	Owner + evaluator
1	Validate tool-call pass-through and response schema.	LiteLLM owner
2	Package Person B configuration as versioned config.	Person B + owner
3	Create 10-task A/B smoke suite.	Evaluator
4	Run 30–50 task controlled benchmark.	Evaluator
5	Security review of MCP/tool permissions and isolation.	Owner + security reviewer
6	Run full DeepSWE benchmark.	Evaluator
7	Publish reproducible report.	KD
8	Offer Agentic/Universal configuration as a product tier.	KD leadership

14. Conclusion

The proposed experiment treats the model, gateway and agent harness as separate engineering layers. The observed tool-call result demonstrates that the current proxy can expose agentic model behavior when configured appropriately. DeepSWE provides an external long-horizon software-engineering benchmark with isolated environments and programmatic verification.

Person B's configuration should therefore be evaluated as an engineering hypothesis, not accepted or rejected by reputation. The correct test is controlled: same model, same tasks, same verifier, different configuration. If the Universal/Hybrid configuration produces higher verified task success, lower cost per successful task, lower context consumption and reliable tool execution, KD

can position the result as an agentic execution optimization layer.

Recommended decision: proceed with the controlled A/B experiment; let the LiteLLM owner retain production authority; have Person B supply the proposed configuration/harness; run the internal test before the full DeepSWE suite; and publish only measured results. The simulated numbers in this document are targets for experimentation, not claims.

References

- [1] DataCurve AI. DeepSWE: Measuring frontier coding agents on original, long-horizon software-engineering tasks. Public benchmark repository and v1.1 methodology.
- [2] LiteLLM. Getting Started and Proxy Server documentation. Centralized LLM gateway, routing, budgets, authentication, logging and project customization.
- [3] DeepSeek. Models and Pricing / API documentation. Current model capability documentation including thinking modes, context and tool calls.
- [4] Unsloth. Benchmark documentation and published implementation-level comparisons against baseline implementations.