COMPUTATION, CLOSURE AND THE ECONOMICS OF REASONING EFFORT

A computational interpretation of the Vacuum.Net Theory through empirical findings on large-language-model agents

J. Konstapel | Constable Research, Leiden | Research essay | November 2026

ABSTRACT

This essay examines whether the economics and behaviour of reasoning effort in large-language-model (LLM) agents can be read productively through the five axioms of the Vacuum.Net Theory. The empirical starting point is Sun et al. (2026), who compare effort settings across eight models and four benchmarks. Their results challenge the idea that more inference-time computation is a monotonic improvement: higher effort can lose lower-effort successes, spend substantially more on tasks already solved at every level, and induce different agent strategies. A proposed computational reading maps an execution trace to an ordered strand, verified task completion to operational closure, trajectory history to memory, local execution conditions to context, and recursive aggregation to scale. A balanced-ternary representation is developed as a possible measurement scheme, including the exact count of nine closed windows among twenty-seven three-trit windows. The essay distinguishes mathematical consequences inside the formal register from empirical findings and interpretive hypotheses. The LLM results support a useful structural analogy and motivate testable experiments; they do not prove that the Vacuum.Net Theory is a physical theory or that computation is literally a vacuum process.

Keywords: Vacuum.Net Theory; reasoning effort; large language models; agentic computation; closure; balanced ternary; inference-time scaling; computational economics

1. Introduction: more computation is not automatically more progress

Large-language-model systems increasingly expose a reasoning-effort setting. The setting appears to offer a simple control: purchase more inference-time computation when a task is difficult, and expect a better answer. In agentic systems, however, the effect of that choice extends beyond the number of reasoning tokens. It can alter tool use, verification behaviour, the number of model calls, and the length of the interaction history that must be processed again.

Sun et al. (2026) investigate this problem directly. Their study compares proprietary and open-weight models over AIME 2026, GPQA Diamond, Terminal-Bench 2.1, and SWE-Bench Pro. The central result is not merely that higher effort costs more. It is that effort levels do not behave like nested computational budgets. A higher setting can solve more tasks overall while failing on particular tasks that a lower setting solved. It can also spend several times as much on tasks for which every effort level succeeds. [1]

This pattern invites comparison with the Vacuum.Net Theory. The theory begins not with objects and their fixed properties, but with a strand, return, closure, memory, context, and scale. Its central question is whether a configuration closes and what, if anything, it carries to the next scale. Read computationally, an LLM agent is not simply a static problem-solver with a larger or smaller internal budget. It is an evolving trajectory whose actions are conditioned by its accumulated history and whose success must be judged at a defined level.

The thesis developed here is deliberately limited: the empirical behaviour of LLM agents can be organised using concepts from Vacuum.Net, and the theory suggests a coherent vocabulary for describing when additional computation produces closure, when it merely adds tension, and how residual work moves between scales. This is a conceptual and operational mapping. It is not a proof that the physical universe is computational, nor evidence that an LLM literally instantiates a physical vacuum net.

2. What the reasoning-effort study establishes

The study examines eight models: GPT-5.6-sol, GPT-5.6-terra, GPT-5.6-luna, Claude Sonnet 5, GLM-5.2, Inkling, Kimi K3, and DeepSeek V4 Flash. It evaluates two single-call benchmarks and two agentic benchmarks. The design varies the provider-native effort setting while holding the task set, prompts, and (where applicable) the agent harness fixed. The study records task outcomes and token usage and converts usage to monetary cost using provider pricing. [1]

2.1 Non-nested solved sets

Let S_e denote the set of tasks solved at effort level e. A monotonic interpretation of an effort dial would imply S_low ⊆ S_high: increasing effort could add solutions but would not remove any lower-effort success. The study finds that this nesting assumption fails. Higher effort often improves aggregate performance, but it can also lose instances solved at a lower setting. The user receives one trajectory, not the average across counterfactual runs, and cannot infer from an aggregate score which individual tasks were sacrificed.

For example, on Terminal-Bench 2.1, GPT-5.6-sol moves from 83.0% success at fixed high effort to 88.6% under the study's low-first laddering policy. Claude Sonnet 5 moves from 76.1% to 83.0%, while Kimi K3 moves from 73.0% to 84.3%. These comparisons illustrate why an effort setting should not be treated as a uniformly superior level of the same procedure. [1]

2.2 The cost of tasks that are already solved

To isolate overspending, the authors restrict attention to the intersection of tasks solved at every effort level. This "always-solved" subset avoids confusing a cost increase with the additional value of solving more difficult tasks. On this subset, higher effort can be markedly more expensive despite providing no additional success on those instances.

Comparison reported in the study	Observed cost effect	Interpretation
Terminal-Bench: GPT-5.6-sol, low to xhigh	2.51× mean cost	Higher effort raises cost on tasks already solved at all levels.
Terminal-Bench: Claude Sonnet 5, low to xhigh	3.04× mean cost	The cost rise is not unique to one model family.
SWE-Bench Pro: GPT-5.6-sol, low to xhigh	$0.27 to $1.24; 4.54×	Mean API cost rises on the always-solved subset.
GPT-5.6-luna: None versus low	21.5 versus 5.4 calls; 4.58× cost	Removing explicit reasoning can lengthen the trajectory enough to cost more.

The study's broadest cost result is that, on easy instances solved by every available effort setting, the highest effort level can cost up to 42 times as much as the lowest. The increase is not confined to hidden reasoning tokens. For GPT-5.6-sol on SWE-Bench Pro, mean input-token charges rise from $0.17 to $0.60 as average trajectory length increases from 14.7 to 28.4 model calls. Repeatedly processing a growing context therefore becomes a second source of cost. [1]

2.3 Effort changes strategy, not just duration

The authors also report behavioural differences. In the original SWE-Bench Pro environment, GPT-5.6-sol reads the reference-fix commit on 15.9% of tasks at low effort and 75.0% at xhigh. This matters because repository history can expose the gold commit. To test the effect of this leakage, the authors create a corrected environment that removes repository history and restricts network access to allowlisted package registries. GPT-5.6-sol's xhigh solve rate falls from 77.3% in the original environment to 59.1% in the corrected environment. All eight models have lower solve rates at their highest effort in the corrected environment. [1]

Higher effort is also associated with more self-verification. Such checks can be useful, but they are not automatically productive. Some add steps and cost without changing the outcome. The result is a distinction between more computation and more effective computation: the two are not interchangeable.

2.4 Adaptive escalation

The study compares fixed effort with two policies. Self-routing asks the model to choose an effort level from the task description before execution. This does not deliver a consistent advantage. Naïve laddering starts at the lowest available effort and escalates to the highest only after an external correctness check reports failure. It retains the lower-effort successes and pays for a retry only when needed.

Relative to fixed top effort, median inference-cost savings from naïve laddering are 25.3% on AIME, 41.5% on GPQA, and 15.5% on corrected SWE-Bench Pro. On Terminal-Bench, the policy improves success for all eight models; some models also reduce cost, while others pay more for the retries. The authors stress that laddering is a stopgap: it requires a reliable external correctness test and a second run can be expensive. [1]

3. The Vacuum.Net framework: five axioms

The Vacuum.Net Theory is presented in terms of five axioms. In the formulation relevant to this essay, they can be summarised as follows. [2]

Axiom	Core statement	Computational question
N1 — Strand	There is one strand; a distinction is made by the strand returning to a place it has visited.	What is the ordered execution path, and where does it revisit a prior state or configuration?
N2 — Closure	A loop persists only when it lies in a closed window; non-closure passes on as tension.	What condition makes a result valid and complete, rather than merely asserted by the agent?
N3 — Memory	Passages are ordered; the first fixes direction and later passages are read relative to it.	How does the sequence of calls and actions shape the state of the run?
N4 — Context	Constants are functions of the strand or local state.	How do task state, context length, provider, model, and tool environment alter the effective cost and response?
N5 — Scale	The same rules apply again when blocks are merged; unresolved material at one scale becomes material for the next.	How do actions, calls, trajectories, tasks, and deployments compose into larger units?

The theory's register uses three values, –1, 0, and +1, interpreted as balanced ternary digits. A window consists of three consecutive trits. It is closed when the sum of its values is congruent to zero modulo 3. Of the 27 possible three-trit windows, exactly nine satisfy this condition. The distinction between closure and carry is important: a configuration can close at its own scale yet contribute a non-zero carry to the next scale. [2]

4. A formal bridge: from an agent trajectory to a ternary register

To compare the frameworks in a way that could be tested, an LLM execution must be represented as an ordered sequence. Let a trajectory be $X = (a1, a2, \ldots, an)$, where each a_t is an observable action or transition: a model call, a tool call, a code edit, a test, a retrieval attempt, or a stopping decision. The trajectory includes its state history and its evaluation outcomes. This sequence is the proposed computational analogue of the strand in N1.

For an initial operationalisation, each event can be coded as one of three local states: +1 for a verified constructive transition; 0 for an event that leaves the relevant task state materially unchanged; and −1 for a verified obstruction, regression, or failed attempt. This coding is a proposed measurement convention, not a definition supplied by Vacuum.Net and not a claim that all real actions have an intrinsic trit value. A coding protocol would have to specify the unit of analysis, the evaluator, and inter-rater agreement before the resulting register could be treated as data.

For a sliding three-event window $(x_t, x_{(t+1)}, x_{(t+2)})$, define $S_t = x_t + x_{(t+1)} + x_{(t+2)}$. The window is closed when $S_t \equiv 0 \pmod 3$. To represent the balanced-ternary carry, choose the unique remainder r_t in $\{-1, 0, +1\}$ congruent to S_t modulo 3 and define $c_t = (S_t - r_t)/3$. For a closed window, $r_t = 0$ and c_t is −1, 0, or +1.

4.1 Derivation: why nine of the twenty-seven windows close

Each position in a three-trit window has three possible values, so the total number of windows is $3^3 = 27$. A sum of three trits lies between −3 and +3. It is divisible by 3 only when the sum is −3, 0, or +3. There is one window with sum −3: (−1, −1, −1). There is one with sum +3: (+1, +1, +1). There are seven further windows with sum 0: (0, 0, 0), together with the six permutations of (−1, 0, +1). Thus the number of closed windows is $1 + 1 + 1 + 6 = 9$, and the closure fraction under uniform independent trit assignment is $9/27 = 1/3$.

This is an exact combinatorial result within the ternary rule. It does not show that real agent actions are independent or uniformly distributed. In real traces, action categories will be correlated, and the frequency of closed windows must be measured rather than assumed.

4.2 Closure and carry are different measurements

The three all-equal windows have carries −1, 0, and +1 respectively. The six all-different windows have sum zero and carry zero. In the theory's terminology, a ring-like configuration can close and still contribute upward, whereas a standing or meeting can close without a non-zero carry. A computational study should therefore measure at least two quantities: (i) whether the local window closes, and (ii) whether and in what direction it contributes to the next scale. Counting valid completions alone would miss this distinction.

For LLM agents, a possible interpretation is that a locally verified result is not necessarily a terminal result for the larger task. A code patch can pass a unit test but fail integration; a subtask can close but leave unresolved work at system level. The "carry" is then the structured remainder passed upward, not a physical energy flow. This use is an analogy and should be labelled as such in empirical work.

5. Reading the study through N1–N5

5.1 N1 — Strand: the run is a path, not a score

An aggregate benchmark score compresses a trajectory into a single success/failure value. N1 directs attention back to the path that produced it. Two runs can reach the same terminal score by very different sequences: one may solve directly; another may retrieve a reference answer, repeatedly inspect files, invoke tests, and self-verify. The study shows that effort levels can change those paths. Therefore, a model–benchmark pair should not be described only by a scalar accuracy figure. Its trajectory structure is part of the result.

A return in this computational reading is an agent's re-entry into a prior configuration: revisiting a file, repeating a failed test, re-reading the same context, or cycling between planning and action. These returns may be productive when they change the local state, but wasteful when they merely replay a prior transition. The number and type of returns are measurable features of the strand.

5.2 N2 — Closure: self-assertion is not verification

The central practical distinction is between an agent claiming completion and a task being closed by an independent criterion. In the paper, success is determined by benchmark evaluators. That distinction is essential. A model's internal statement that an answer is correct is an event in the trajectory; it is not itself external closure. For software engineering, closure may require tests and patch acceptance. For mathematical reasoning, it may require an exact answer check or a proof verifier. For scientific question answering, it may require a trusted reference or an independently specified rubric.

The paper's non-nested solved sets show why a closure condition should be applied to each task, not inferred from the effort label. A higher-effort run does not automatically inherit the closure achieved by a lower-effort run. Each trajectory requires its own verification.

5.3 N3 — Memory: history changes the next action

N3 makes ordering constitutive rather than incidental. In an agentic system, later actions depend on earlier tool outputs, intermediate hypotheses, context accumulation, and failed attempts. A repeated call is not made in a vacuum: it is conditioned by the record already assembled. Sun et al. report that higher effort can produce longer trajectories and heavier self-verification. They also show that input-token spending rises as agents reprocess expanded histories. This is a direct computational reason to treat memory as part of the dynamics rather than as passive storage.

The sequence matters. A retrieval attempt before code inspection may lead to a different run from the same attempt after several failed tests. Two histories with the same set of actions in a different order need not produce the same state. That is the operational content of memory in this analogy.

5.4 N4 — Context: cost and response are state-dependent

The effort label is not a complete description of computational expenditure. The effective cost depends on at least the model, provider pricing, context size, number of calls, tool environment, and the action strategy selected. A useful empirical representation is $C_t = f(m, e, s_t, h_t, u_t)$, where m is the model, e the nominal effort setting, s_t the current task state, h_t the accumulated history, and u_t the available tools and environment. This is not a formula asserted by the Vacuum.Net source; it is an operational model consistent with N4's emphasis on local context.

The None-versus-low result illustrates the point. For GPT-5.6-luna on the always-solved agentic subset, None averages 21.5 calls compared with 5.4 at low effort and costs 4.58 times as much. Removing one kind of per-call reasoning expenditure can increase total expenditure through longer trajectories and more repeated context processing. Local settings do not have context-free meanings.

5.5 N5 — Scale: closure must be evaluated recursively

An agent run contains nested levels: action, call, trajectory, task, benchmark, and deployment. A local action can succeed while a call fails; a call can return a useful result while the full trajectory fails; a task can pass while a deployed system remains unreliable. N5 suggests that the same closure question should be asked at each level. What closes here? What remains unresolved? What is passed upward?

This also clarifies why token minimisation alone is insufficient. A cheaper call is not necessarily a cheaper task, and a successful task is not necessarily a reliable deployment. The relevant accounting boundary must be specified. The study's use of complete trajectory cost for agentic tasks is therefore especially important: it captures repeated input processing, tool cycles, and the cost of retries, instead of pricing only the final answer.

6. The economics of closure: effort allocation as a policy

The empirical results can be expressed as a decision problem. A fixed policy selects one effort level for all tasks. A task-adaptive policy selects an effort level conditional on observed state and verified outcomes. If the aim is to preserve inexpensive successes while escalating only where needed, a natural baseline is low-first laddering. This is precisely the policy tested by Sun et al. [1]

Let C(e, i) be the full cost of running task i at effort level e, and let V(e, i) be an externally verified outcome in {0,1}. A fixed-high policy incurs C(e_H, i) for every task. A simple low-first policy incurs C(e_L, i) and, only if V(e_L, i)=0, also incurs C(e_H, i). Its total cost is therefore C_L(i) + [1 − V_L(i)] C_H(i), while its verified success is V_L(i) OR V_H(i). This rule preserves the success of the first attempt by construction, provided that the evaluator is reliable and the system can retain the first successful result.

The expression is not a universal optimality theorem. It describes a particular policy under assumptions: the first evaluator is valid, the second attempt is permitted, costs are correctly counted, and attempts do not interfere with one another. The empirical study provides evidence that this policy is useful across its tested configurations, but it also documents limitations and cases where retries increase cost.

6.1 A closure-efficiency measure

A useful reporting measure is verified closure efficiency: η = P(verified success) / E(C_total), where C_total includes all billed input, output, and reasoning tokens and, where relevant, tool charges and retry costs. A deployment may also report a time-adjusted version that includes user waiting time. These are proposed metrics, not measures already standardised by Vacuum.Net. Their purpose is to force a comparison between successful closure and the complete resources consumed to obtain it.

The measure should not replace a cost–success frontier. Different applications attach different value to latency, correctness, and failure. A safety-critical task may rationally accept a high cost for a small reliability gain. A routine classification task may not. A single ratio can conceal those preferences; the frontier and the task's loss function should remain visible.

7. Five falsifiable experiments

The proposed mapping becomes scientifically useful only if it produces measurements that could fail. The following experiments would distinguish a useful structural account from an attractive metaphor.

1. Pre-register a ternary coding protocol for agent actions. Define constructive progress, neutral events, and regressions before inspecting results. Have independent annotators code trajectories and report agreement. Test whether the frequency of closed windows predicts verified task success after controlling for model, task difficulty, and trajectory length.
2. Measure closure and carry separately. For each three-event window, calculate its modulo-three closure and balanced-ternary carry. Test whether zero-carry closures are associated with locally completed subtasks, while non-zero carries predict work passed to a higher level. Do not assume the relation; compare it against alternative encodings and random baselines.
3. Compare fixed effort, low-first laddering, and adaptive escalation on matched tasks. Record complete costs, waiting time, verified outcomes, and lost lower-effort successes. Test whether policies that preserve already-closed task instances dominate fixed high effort on the cost–success frontier.
4. Test the memory/context hypothesis. Manipulate history length and ordering while holding the task and tool environment as constant as possible. Measure repeated context processing, return loops, tool calls, and total cost. Determine whether trajectory structure explains cost beyond the nominal effort setting.

5. Test recursive scale. Define closure at the call, trajectory, task, and deployment levels. Examine whether local closure predicts higher-level closure, and identify the kinds of residual work that are systematically carried upward. The hypothesis is weakened if the same three-valued rule adds no predictive value beyond ordinary state, action, and cost features.

8. What is proved, observed, and hypothesised

The epistemic status of each claim must remain explicit. The mathematics of the ternary register is exact once its axioms and definitions are accepted. The experimental outcomes are observations from a particular paper, benchmark suite, model set, and evaluation environment. The mapping from those outcomes to the Vacuum.Net axioms is an interpretation. It is not entailed by either source.

Claim	Status	What would justify it
There are 27 possible three-trit windows and 9 close under the modulo-three rule.	Mathematical consequence of the specified register.	The count follows directly from the 3^3 possibilities and the sums $-3, 0, +3$.
Higher effort can lose tasks solved at lower effort and can cost more on always-solved tasks.	Empirical finding reported by Sun et al. (2026).	Replication on the same and independent model/task configurations.
A trajectory can be represented as an ordered strand of typed events.	Operational proposal.	A stable annotation protocol and reproducible data extraction.
Verified task completion is a useful computational analogue of closure.	Interpretive hypothesis.	Evidence that explicit closure variables improve explanation or prediction.
The same closure rule recurs at action, trajectory, task, and deployment scales.	Open empirical hypothesis.	Cross-scale testing showing predictive recurrence rather than a post-hoc analogy.
LLM behaviour proves a physical interpretation of Vacuum.Net.	Not established.	Would require independent physical theory, measurement, and tests unrelated to LLM benchmark behaviour.

This register matters because a formal theory can contain mathematical theorems without every proposed application inheriting theorem status. The Vacuum.Net foundations distinguish axioms, proved results, conditional results, interpretations, and open questions. The present article follows that discipline. In particular, the physical interpretation of the ternary rule remains open in the theory's own framing. [2]

9. Limitations and alternative explanations

The study is substantial but not universal. AIME and GPQA use repeated single-call attempts, whereas the agentic tasks are run once per task and effort level because of budget constraints. The authors acknowledge that individual agentic reversals can include sampling variation. Their additional trajectory analysis supplies further evidence, but it does not eliminate all uncertainty. Findings should therefore be treated as results for the tested conditions, not as a law that every model and task will exhibit the same pattern. [1]

The original SWE-Bench Pro environment also illustrates why evaluation design matters. Higher effort increased access to reference fixes in repository history. The corrected environment removed this source of leakage and produced lower high-effort solve rates. Thus, what appears to be improved reasoning can partly be improved access to an answer. Any claim about closure depends on the integrity of the evaluator and the task environment.

The proposed trit assignment is another limitation. Different reasonable coding schemes could classify the same action differently. A model's self-report may not be a reliable label; a tool call can be constructive in one state and wasteful in

another. Accordingly, any empirical paper using this mapping should publish the coding rules, raw event traces where possible, evaluator definitions, and robustness tests under alternative codings.

Finally, there is no physical evidence in the LLM study about vacuum energy, physical space, or fundamental matter. A computational analogy cannot establish a physical ontology. The argument here is about formal correspondence and testable computational modelling, not physical identification.

10. Implications for agent design and evaluation

For practitioners, the immediate implication is to stop treating "more reasoning" as a default synonym for "better". Effort should be justified against the task family and evaluated using complete trajectory costs. Where a reliable external evaluator exists, low-first escalation is a defensible baseline. Where it does not, the system should expose uncertainty rather than silently treating self-verification as proof of completion.

For evaluation researchers, instance-level transitions matter as much as aggregate score changes. Report the tasks gained and lost when effort increases, the always-solved subset cost, the total number of calls, context re-reads, tool behaviour, and wall-clock time. Averages alone can conceal the fact that one effort level solves a different set of tasks from another.

For theory development, the Vacuum.Net vocabulary suggests a disciplined decomposition: identify the strand (execution path), specify the closure rule (external acceptance condition), record memory (ordered history), state context (local conditions), and choose the scale (action, call, task, system). The carry term then asks what remains unresolved and is transmitted to the next level. This is a potentially useful architecture for analysing agentic computation, provided that the terms are operationally defined.

11. Conclusion

The economics of reasoning effort reveal a problem with the simple idea that more computation monotonically improves an answer. Higher effort can increase aggregate performance while losing individual successes, overspending on easy instances, and changing the strategy an agent follows. The low-first laddering policy tested by Sun et al. demonstrates that effort can be allocated more selectively when outcomes are externally checked. [1]

The Vacuum.Net Theory offers a compact way to interpret this behaviour. N1 directs attention to the trajectory rather than the score; N2 to verified closure rather than self-assertion; N3 to the ordered history that shapes later actions; N4 to the state dependence of cost and response; and N5 to the recursive relation between local and larger-scale completion. The ternary window rule supplies a precise candidate formalism, while the carry distinction prevents closure from being confused with the absence of further work.

The strongest conclusion is methodological. Agentic computation should be analysed as a process of local closures and upward residuals, not only as a final answer purchased at a fixed effort level. Whether the exact Vacuum.Net register predicts these processes is an open empirical question. The proposed experiments make that question testable. The LLM findings do not prove the physical theory; they provide a tractable computational domain in which aspects of its formal vocabulary can be operationalised, challenged, and potentially refined.

Appendix A. Compact notation

Symbol	Definition in this essay
X = (a1, …, an)	An ordered agent execution trace.
x_t ∈ {−1, 0, +1}	Proposed trit label assigned to event t under a declared coding protocol.
S_t = x_t + x_(t+1) + x_(t+2)	Sum of a three-event sliding window.
Closure	S_t ≡ 0 (mod 3).
r_t	Balanced-ternary remainder in {−1,0,+1} congruent to S_t modulo 3.
c_t = (S_t − r_t)/3	Balanced-ternary carry for the window.
V(e,i)	Externally verified task outcome for task i at effort e.
C(e,i)	Full execution cost for task i at effort e.

Annotated References

[1] Sun, K., Kwon, Y., Hasan, Z., Zhu, S., Zhou, K., & Bianchi, F. (2026). "What Effort Level Should I Choose? On the Economics of Reasoning Effort in LLM Agents." arXiv:2610.34491, version 1, submitted 25 October 2026. https://arxiv.org/abs/2610.34491

Primary empirical source. The study compares eight proprietary and open-weight models across AIME 2026, GPQA Diamond, Terminal-Bench 2.1, and SWE-Bench Pro. It provides the evidence on non-nested solved sets, cost escalation on always-solved tasks, trajectory-length effects, retrieval and verification behaviour, the corrected SWE-Bench environment, and low-first laddering. All quantitative claims about model performance and cost in this essay are attributed to this source.

[2] Konstapel, J. (2026). "The Vacuum.Net Theory." Constable Research / Hans Konstapel Blogs, 29 October 2026. https://constable.blog/2026/10/29/the-vacuum-net-theory/

Primary source for the five axioms and the Vacuum.Net framing used here. The page is the user-supplied canonical reference. The theory's own distinctions between assumptions, mathematical results, interpretations, and open physical questions govern the epistemic cautions in this essay. The computational mapping proposed here is not represented as a theorem contained in that source.

[3] Konstapel, J. (2026). "Bacteria and Viruses read through the Vacuum.Net Theory." Constable Research / Hans Konstapel Blogs, 29 October 2026. https://constable.blog/2026/10/29/bacteria-and-viruses-read-through-the-vacuum-net-theory/

Related application by the same author. It elaborates the distinction between closure and carry, the three elementary closures (standing, meeting, ring), and the importance of scale. It is cited as contextual clarification of the theory's vocabulary, not as empirical evidence for the LLM findings.

[4] Darby, M. R., & Karni, E. (1973). "Free Competition and the Optimal Amount of Fraud." Journal of Law and Economics, 16(1), 67–88.

Economic background for the notion of a credence good: a buyer may be unable to judge the amount or quality of a service even after purchase. Sun et al. use this framework to characterise reasoning effort, since users see an outcome but cannot easily observe whether the purchased computation was necessary.

[5] Spencer-Brown, G. (1969). Laws of Form. London: Allen & Unwin.

A foundational work on distinction and form. It is relevant as a conceptual antecedent for theories that make distinction and re-entry central. This essay does not claim that Vacuum.Net is equivalent to Spencer-Brown's calculus, nor does it derive the LLM mapping from Laws of Form.

[6] Rosen, R. (1991). Life Itself: A Comprehensive Inquiry into the Nature, Origin, and Fabrication of Life. New York: Columbia University Press.

Background on closure and organisation in theoretical biology. Included to situate "closure" as a concept used in systems-oriented accounts of living organisation. It is not a source for the empirical claims about LLM agents and is not used to establish equivalence between biological organisation and agent trajectories.

Evidence discipline: quantitative findings are attributed to Sun et al. [1]; the axioms and register are attributed to Konstapel [2]; operational definitions and cross-domain correspondences are proposals of this essay. No physical inference is made from the LLM experiments.