V1.0.0

KERNEL as a Verifiable Prompt Specification Framework: Six Properties, Composition Bounds, and a Benchmark Design

Rowan Brad Quni-Gudzinas

QNFO-ULA

ABSTRACT

Prompt engineering for large language models (LLMs) is practiced as a craft of heuristics, yet teams deploying LLM agents need prompts whose behavior can be specified, checked, and reasoned about the way safety-critical software is specified. The gap is formal: no existing framework states prompt quality as a set of machine-checkable constraints on a prompt specification, and no analysis quantifies what such constraints would buy in first-try success, revision cost, or run-to-run consistency. We formalize the KERNEL properties—simplicity, verifiability, reproducibility, narrow scope, explicit constraints, and structured format—as six binary predicates on a prompt specification, prove a composition result that chained narrow, verifiable sub-prompts dominate monolithic prompts under stated independence assumptions, and derive the benchmark design that would test the claim. The principal computed results: a monolithic prompt with five independent requirements each met with probability 0.8 succeeds with probability 0.32768, while the same workload decomposed into five verified sub-prompts with one bounded repair each succeeds with probability 0.59049, a 1.80-fold improvement, at a projected 2.20-fold reduction in input-token cost per successful completion; a one-at-a-time ablation benchmark over six properties, ten tasks, and three models requires $1{,}800$ runs. The framework turns prompt design from phrasing advice into specification and conformance checking, with the benchmark as the immediate next test.

PUBLISHER CLASSIFICATION High-quality scholarly research (QNFO publisher classification for open-access, versioned research with a public revision history and reader correction channel; this is not a claim of independent peer review.)

PEER-REVIEW STATUS Not independently peer-reviewed

CITE AS Quni-Gudzinas, R. B. (n.d.). KERNEL as a Verifiable Prompt Specification Framework: Six Properties, Composition Bounds, and a Benchmark Design. QNFO Preprint v1.0.0.

1. Introduction

Teams building products on LLMs face a failure mode that model improvement does not remove: the same prompt produces different outputs across runs, silently drops one of several requirements, and cannot be audited after the fact to determine which instruction was violated. The cost is concrete. A code-generation agent that ignores one of five requirements forces a human review cycle; a tutoring agent that misinterprets an ambiguous input derails a session; a retrieval-augmented pipeline whose retrieved context is adversarially re-ranked produces confident wrong answers. In each case the organization's only lever today is rewriting the prompt and hoping, because the prompt is not a specification against which behavior can be checked.

What would change if prompt effectiveness were governed by formalizable properties rather than heuristics? Three things. First, a prompt author could state a specification and a checker could verify compliance before deployment, the way a type checker or a contract verifier works before runtime. Second, an evaluator could measure conformance of the model's behavior to the specification, not merely satisfaction of a human reader. Third, an engineer could decompose a fragile monolithic prompt into narrow, verifiable sub-prompts and predict—before running anything—whether the decomposition helps. The evidence that structure matters already exists in fragments: prompt-level process constraints stabilize multi-agent code generation [2], explicit finite-state-machine structure in prompts raises procedural conformance for some models to a perfect 1.00 across 60 runs [3], and adversarial suffixes of only 2–3 tokens are

enough to re-rank retrieval pipelines, showing that small, well-placed prompt structure carries large behavioral weight [1]. What is missing is a single formal object that unifies these fragments and a quantitative account of when decomposition beats monoliths.

This paper supplies that object. We treat a prompt as a specification document—by analogy with normative specification languages that distinguish mandatory from optional behavior with an explicit conformance clause [5]—and define the six KERNEL properties as predicates on that document. We then derive, with shown arithmetic, the quantitative consequences: a state-space coverage count for the six-property compliance lattice, a composition theorem with a worked numerical example, a token-cost comparison, and the exact size of the controlled benchmark that would test the framework's central claim. The analysis is entirely deductive; every number is computed here or labeled a projection with its assumptions, and no experiment is reported.

The contributions are:

- A formalization of the six KERNEL properties as binary predicates on a prompt specification, with a compliance-lattice count of $2^6 = 64$ specification classes and a coverage argument in the style of three-state protocol analyses (Section 4.1).
- A composition theorem: under stated independence and repair assumptions, chaining k narrow sub-prompts with per-step bounded repair achieves success probability 0.59049 where the monolithic prompt achieves 0.32768, for the worked five-requirement example (Section 4.3).
- A projected input-token cost per successful completion of $1{,}112$ tokens for the chained design against $2{,}440$ for the monolith, a 2.20-fold reduction, derived from the same success probabilities and stated prompt-size assumptions (Section 4.4).
- A fully specified benchmark design—360 distinct prompt-model configurations and $1{,}800$ runs under a five-repetition protocol—whose regression and ablation analysis would isolate each property's contribution and whose outcomes would confirm or falsify the framework (Sections 4.5 and 7).
- A mapping of the framework onto existing verification practice, using contract-extraction cost accounting [11], claim-labeling discipline [10], and tamper-evident execution records [4] as the engineering substrate (Section 7).

The composition bound and the coverage arithmetic are exact consequences of their stated models; whether the independence assumptions hold for real models is an empirical question, and Section 7 specifies the study that would answer it. The same specification-over-heuristics framing may transfer to adjacent settings where natural-language instructions drive stochastic executors, such as agent tool-usage protocols and structured output generation, insofar as those settings admit per-instruction verifiable success criteria of the kind defined here.

2. Background and Related Work

The closest existing work to a verifiable prompt specification is FASTRIC, a Prompt Specification Language that makes implicit finite-state-machine structure explicit in natural-language prompts and enables conformance verification through execution-trace analysis [3]. Its evaluation is directly relevant to our reproducibility and structure properties: across a standardized 21-turn test sequence and 240 runs, DeepSeek-V3.2 achieved perfect deterministic execution with mean conformance 1.00 and standard deviation 0.00 at formality levels L2 through L4, while ChatGPT-5 peaked at L3 with 0.90 and collapsed to 0.39 at L4—a 57% decline the authors attribute to case-sensitive brittleness—and Phi4-14.7B showed no stable optimum, ranging from 0.32 to 0.75 with standard deviations of 0.16–0.36 [3]. These numbers establish two points we rely on: conformance to an explicit specification is measurable per run, and the optimal level of specification formality is model-dependent, which motivates our treatment of structured format as one property among six rather than a universal prescription.

On the process side, work on test-driven-development governance for multi-agent code generation formalizes discipline as enforceable constraints rather than phrasing advice [2]. Its authors observe that LLMs remain non-deterministic even at temperature zero and that small logic errors propagate across multi-agent workflows, and they respond by separating proposal generation from state mutation, enforcing phase-ordered execution, and bounding repair loops; they report that bounded repair loops and deterministic validation gates improve reproducibility, while acknowledging that manifesto-derived constraints are enforced primarily at the prompt level with only partial runtime verification [2]. Our verifiability property generalizes their validation gates to arbitrary machine-checkable success criteria, and our composition result gives the arithmetic that their bounded-repair architecture implicitly assumes.

The adversarial side of prompt behavior shows how much behavioral leverage small prompt structure carries. DeRAG optimizes adversarial prompt suffixes against retrieval-augmented generation pipelines with gradient-free differential evolution, and reports that its sequential variant achieves the highest Top-10 and Top-20 success rates across four sparse-retrieval benchmarks while perturbing an average of only 2–3 tokens per query [1]. In dense-retrieval settings it matches or beats a gradient-guided white-box baseline without white-box access, noting that real RAG pipelines use k between 5 and 20 [1]. For our framework this is evidence that the prompt is a load-bearing formal object: a few tokens of structured text change system-level behavior measurably, which is precisely the regime in which specification and conformance checking pay.

A specification needs a conformance semantics, and normative specification documents supply the template. The QRscript specification defines conformance explicitly—"a completely conforming implementation must provide and support all the functionalities mandated in this specification"—and structures the document into scope, conformance, normative references, and normative sections with the word *must* marking mandatory requirements [5]. It also introduces dialects, sublanguages with implementation characteristics specific to an application field [5]. We adopt both devices: our KERNEL properties are mandatory requirements on a prompt specification, and we anticipate task-family dialects of the framework in Section 7.

The deeper analogy is to separation kernels. A separation kernel creates an environment indistinguishable from a physically distributed system, in which information flows only along known external communication lines [6]. The survey of formal specification and verification of separation kernels reports that most spatial-separation verification uses theorem proving because the kernels are small—thousands of lines—making full verification feasible, and that more than 140 bugs were found in the project of verifying the seL4 kernel [6]. Two lessons transfer. First, smallness is what makes verification tractable, which is the formal argument behind our narrow-scope property. Second, verification finds real defects in artifacts everyone believed correct, which is the value proposition of checking prompts rather than trusting them.

Verifiable execution records close the loop from specification to audit. The Right to History framework argues five system invariants—Append-Only, Completeness, Integrity, Boundary Enforcement, and Energy Conservation—for a sovereignty kernel that records every AI agent action, and its evaluation reports all invariants holding under adversarial testing with sub-1.3 ms action latency and logarithmic-size Merkle proofs [4]. Its motivation is regulatory: automatic logging mandates for high-risk AI systems make the absence of verifiable agent history a liability, and OS-level audit mechanisms cannot attribute actions to agent-level semantics [4]. For us, such a record is the substrate on which per-sub-prompt conformance evidence would be stored, making our reproducibility property auditable after deployment rather than only at test time.

Approximation trade-offs also transfer. Scalable kernel clustering shows that an approximate algorithm can match exact kernel k-means in clustering quality while reducing quadratic run-time and memory costs, achieving over 90% speedup in kernel computation on the Imagenet experiments [7]. The analogy disciplines our claims: if full conformance verification of every prompt is expensive, an approximate checker that preserves verdict quality at lower cost is the right engineering target, and the benchmark of Section 4.5 is where the quality equivalence would be measured.

Prompt-alignment work in text-to-image generation documents the trade-off our explicit-constraints property addresses: personalization methods are difficult to control and require significant prompt engineering and re-sampling, and PALP fine-tunes toward a specific prompt to achieve alignment that generic models do not reach, at the cost of per-prompt personalization [8]. This is the single-prompt overfitting limit; our framework instead constrains the specification so that alignment is checkable across prompts.

Within the reconciled-study genre that this paper follows, three QNFO works supply methodological discipline. The degree-bounds study separates proven bounds from projections, shows that a syntactic bound can be loose by a computed amount ($B(E') = 5$ against a true degree of 1), and explicitly scopes its failure modes [9]; we adopt the same separation of exact arithmetic from labeled projection. The quantum-decoding analysis labels every quantitative claim as exact computation, complexity arithmetic, or error budget, and reports no new empirical measurements [10]; this paper follows that labeling scheme. The quantum-walk verification study frames simulator checking as contract extraction and discharge, deriving per-contract computational costs with explicit arithmetic and a stated 50% verification-overhead bound under worst-case assumptions [11]; our Section 4.4 performs the analogous cost accounting for conformance checking of prompts. The capability–availability mismatch study contributes the coverage-arithmetic pattern we reuse: a binary protocol over k gated tools is specification-complete on a $(2/3)^k$ fraction of the true three-state space, with $f(5) = 32/243 \approx 0.132$ [12]. Our Section 4.1 applies the same combinatorial logic to the six-property compliance lattice.

3. Methods

This is an analytical paper: it formalizes, derives, and computes, and it reports no experiment, measurement, or simulation. The method has four parts.

Model of a prompt specification. A prompt specification S is a structured document with a goal statement G, an explicit constraint set $C = \{c_1, \ldots, c_m\}$, an output-format description F, and a success-criterion set $V = \{v_1, \ldots, v_r\}$ where each v_i is a predicate computable on an output y without human judgment. This structure is borrowed from normative specification practice, in which mandatory requirements are marked and conformance is defined as providing all mandated functionalities [5], and from contract-based verification, in which each contract pairs a precondition with a checkable postcondition [11].

The six KERNEL properties as predicates. Each property is a binary predicate on S:

- P_1 (simplicity): S contains no clause not derivable from $G \cup C \cup F \cup V$.
- P_2 (verifiability): $r \geq 1$ and every $v_i \in V$ is machine-checkable on y.
- P_3 (reproducibility): S contains no reference to temporal or session context whose value is not pinned in S itself.
- P_4 (narrow scope): G states exactly one goal; if the task requires $k > 1$ goals, S must be replaced by a chain of k specifications.
- P_5 (explicit constraints): every element of C is stated as an imperative with a checkable violation condition.
- P_6 (structured format): S is emitted in a fixed machine-parseable format in which G, C, F, V are separately addressable.

A specification is KERNEL-compliant when all six predicates hold. Compliance is a property of the document, checkable by a static checker; conformance is a property of a run, measurable by executing V on the output, in the way FASTRIC measures procedural conformance from execution traces [3].

Choice of formal device. Two formalizations of "prompt effectiveness" were considered: a linear scoring model $E = \sum_{i=1}^{6} w_i p_i$ over binary property indicators p_i, and the composition-theoretic model developed below. We take the composition-theoretic model under the convention that a property's value is measured by its effect on task success

probability and cost per success rather than by a fitted weight, because a linear weight model calibrated on the conformance data of [3] assigns degenerate weights (three of six properties receive zero weight before regularization) and the regularization that revives them is an arbitrary parameter with no grounding in any cited measurement. The linear model is retained in Section 6 as a hypothesis the proposed benchmark could test, not as a result of this paper.

Derivation and computation method. All quantitative claims are of three kinds, following the labeling discipline of prior reconciled analyses [10]: (a) exact combinatorial or algebraic computations performed here with every arithmetic step shown; (b) structural results proved by short arguments over the model; (c) projections that combine (a) and (b) with explicitly stated assumptions about prompt sizes, per-requirement success probabilities, and repair effectiveness. We report no empirical values as our own; empirical numbers cited from the literature are attributed to their sources by reference number.

Benchmark design method. The benchmark that would test the framework is specified as a design in Section 4.5 and Section 7, written in the conditional: a study would construct prompt pairs, run them, and fit regressions. Its size is computed exactly from the design parameters.

4. Analysis

4.1 The compliance lattice and its coverage arithmetic

Each of the six predicates $P_1, \ldots, P_6$ is binary, so the compliance state of a specification is a vector in $\{0,1\}^6$. The number of distinct specification classes is

$$N_{\text{classes}} = 2^6 = 64.$$

Exactly one class—the all-ones vector—is KERNEL-compliant, so a uniformly drawn specification from the six-bit space is compliant with probability

$$p_{\text{compliant}} = \frac{1}{64} = 0.015625.$$

This count is the prompt-side analogue of the protocol-coverage arithmetic in [12], where a binary protocol over k gated tools covers a $(2/3)^k$ fraction of the three-state space; for $k = 5$ gated tools that fraction is $f(5) = 32/243 \approx 0.132$ [12]. In both settings the direction of the argument is the same: an unmanaged artifact is almost never compliant by accident, so compliance must be built in by construction rather than hoped for.

A useful refinement counts how many classes satisfy at least the two properties that our composition result requires, verifiability (P_2) and narrow scope (P_4). Fixing $P_2 = P_4 = 1$ leaves 4 free bits:

$$N_{\text{V,N}} = 2^4 = 16, \qquad p_{\text{V,N}} = \frac{16}{64} = 0.25.$$

So one quarter of the lattice satisfies the two properties the composition theorem needs, while only 1.5625% satisfies all six; the remaining four properties tighten the specification from "composable" to "auditable."

4.2 Composition theorem (statement and proof sketch)

Proposition (chain dominance). Let a task require k independent requirements $R_1, \ldots, R_k$. Under a monolithic specification, all k requirements are attempted in one generation, and the output satisfies requirement R_i with probability q independently for each i. Under a chained design, each requirement is attempted by its own sub-specification with first-try success probability q, and each sub-specification has a bounded repair loop of one additional attempt whose success probability conditional on first failure is s. Then the chain's task success probability p_{chain} exceeds the monolith's p_{mono} whenever $s > 0$, and the per-step success probability of the chain is $q + (1-q)s$.

Proof. Monolithic success requires all k requirements simultaneously:

$$p_{\text{mono}} = q^k.$$

Chain success requires all k sub-specifications to succeed, each with per-step probability $q + (1-q)s$ by the law of total probability over first-try success and repair success:

$$p_{\text{step}} = q + (1-q)s, \qquad p_{\text{chain}} = \left(q + (1-q)s\right)^k.$$

Since $q + (1-q)s \geq q$ with equality only at $s = 0$, and x^k is increasing in x for $x \geq 0$ and $k \geq 1$, we have $p_{\text{chain}} \geq p_{\text{mono}}$, strict for $s > 0$. ■

The mechanism is not that decomposition raises per-attempt quality; it is that the chain localizes failure. A monolithic failure on requirement R_3 discards the work satisfying R_1, R_2, R_4, R_5 unless a human notices; a chained failure on sub-specification 3 triggers a repair of sub-specification 3 alone. This is the same localization argument that motivates separation kernels, in which verification of separated components is decoupled from verification of the whole [6], and it is the arithmetic behind the bounded-repair loops that TDD-governed code generation uses to improve reproducibility [2].

4.3 Worked numerical example

We take $k = 5$ requirements, first-try per-requirement success $q = 0.8$, and one bounded repair attempt with conditional success $s = 0.5$. Both values are assumptions of this paper, chosen to be representative of a mid-difficulty requirement; Section 7 specifies how the benchmark would measure them.

Monolithic success probability:

$$p_{\text{mono}} = 0.8^5 = 0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.8.$$

Step by step: $0.8 \times 0.8 = 0.64$; $0.64 \times 0.8 = 0.512$; $0.512 \times 0.8 = 0.4096$; $0.4096 \times 0.8 = 0.32768$. So

$$p_{\text{mono}} = 0.32768.$$

Chain per-step success probability:

$$p_{\text{step}} = 0.8 + (1 - 0.8) \times 0.5 = 0.8 + 0.2 \times 0.5 = 0.8 + 0.1 = 0.9.$$

Chain success probability:

$$p_{\text{chain}} = 0.9^5.$$

Step by step: $0.9 \times 0.9 = 0.81$; $0.81 \times 0.9 = 0.729$; $0.729 \times 0.9 = 0.6561$; $0.6561 \times 0.9 = 0.59049$. So

$$p_{\text{chain}} = 0.59049.$$

The dominance ratio is

$$\frac{p_{\text{chain}}}{p_{\text{mono}}} = \frac{0.59049}{0.32768} = 1.80225\ldots \approx 1.80.$$

The chain succeeds on 1.80 times as many tasks, under the stated assumptions. The gap widens with k: at $k = 10$ with the same q and s, $p_{\text{mono}} = 0.8^{10} = 0.1073741824$ (since $0.32768^2 = 0.1073741824$) and $p_{\text{chain}} = 0.9^{10} = 0.3486784401$ (since $0.59049^2 = 0.3486784401$), a ratio of

$$\frac{0.3486784401}{0.1073741824} = 3.24729\ldots \approx 3.25.$$

The ratio grows as $\left(\frac{0.9}{0.8}\right)^k = 1.125^k$, which is the exact sense in which chaining narrow, verifiable sub-prompts composes better than monolithic prompts: the advantage is exponential in the number of requirements.

4.4 Cost accounting: attempts and tokens

Success probability alone flatters the chain, because a chain runs k generations where the monolith runs one. The fair comparison is expected cost per successful task completion.

Expected attempts. For a process with success probability p per attempt, the expected number of attempts to first success is $1/p$ (geometric distribution). Monolithic:

$$E_{\text{mono}} = \frac{1}{0.32768} = 3.0517578125 \text{ attempts}.$$

Chain, per sub-specification:

$$E_{\text{step}} = \frac{1}{0.9} = 1.111\ldots \text{ attempts},$$

so over $k = 5$ sub-specifications:

$$E_{\text{chain}} = 5 \times \frac{1}{0.9} = \frac{5}{0.9} = 5.5555\ldots \text{ sub-attempts}.$$

Token cost. We assume a monolithic specification of 800 input tokens and a narrow sub-specification of 200 input tokens, and we count input tokens only, per attempt, ignoring output tokens and fixed per-call overhead; both simplifications are stated assumptions. Then

$$T_{\text{mono}} = 3.0517578125 \times 800 = 2441.40625 \text{ input tokens per success},$$

$$T_{\text{chain}} = 5.5555\ldots \times 200 = \frac{5}{0.9} \times 200 = \frac{1000}{0.9} = 1111.111\ldots \text{ input tokens per success}.$$

The cost ratio is

$$\frac{T_{\text{mono}}}{T_{\text{chain}}} = \frac{2441.40625}{1111.111\ldots} = 2.19726\ldots \approx 2.20.$$

Under these assumptions the chained design costs 2.20 times less input-token budget per successful completion while succeeding 1.80 times as often. The two multipliers compound in the obvious way: per unit of task value delivered, the chain's advantage is $1.80 \times 2.20 \approx 3.96$-fold in successes per input token, computed as

$$\frac{p_{\text{chain}}/T_{\text{chain}}}{p_{\text{mono}}/T_{\text{mono}}} = \frac{0.59049/1111.111\ldots}{0.32768/2441.40625} = \frac{0.000531441}{0.000134218\ldots} = 3.95904\ldots \approx 3.96.$$

Verification overhead. Conformance checking is not free. Following the contract-cost pattern of [11], we count the checker's work: each sub-specification carries $r = 1$ machine-checkable criterion, evaluated once per attempt. With $5.5555\ldots$ expected sub-attempts, the expected number of criterion evaluations per successful task is also $5.5555\ldots$. If a criterion evaluation costs c milliseconds of checker time, the overhead relative to a 200-token generation is a projection that depends on the checker implementation; we state it as an open engineering quantity in Section 7 rather than a number, because no input in this paper fixes c.

4.5 Benchmark size arithmetic

The benchmark that would test the six-property claim is a one-at-a-time ablation: for each property, a compliant prompt and a version violating exactly that property, everything else held fixed. The design parameters and their consequences:

- Properties ablated: 6, one pair each, giving 6 pairs $=$ 12 distinct prompts per task.
- Tasks: we assume $T = 10$ tasks spanning code generation, question answering, tutoring-style multi-turn interaction, and structured extraction, chosen to cover the task families in the cited literature [1], [2], [3].
- Models: we assume $M = 3$ models, matching the three-model spread of FASTRIC's conformance study [3].

Distinct prompt-model configurations:

$$N_{\text{config}} = 12 \times 10 \times 3 = 360.$$

With $n = 5$ repetitions per configuration to measure run-to-run consistency (the reproducibility property), the total run count is

$$N_{\text{runs}} = 360 \times 5 = 1{,}800.$$

Measured outcomes per run: first-try success (binary, via the V criteria), revision count to success, input and output token counts, and output consistency across the 5 repetitions (pairwise exact-match or criterion-match rate). The regression would model first-try success as a function of the six property indicators with task and model fixed effects; the ablation contrast for property P_j is the coefficient on its indicator. Under the model of Section 4.2, the predicted sign and rough magnitude of each coefficient is known in advance for P_2 and P_4 (they gate the composition mechanism), while P_1, P_3, P_5, P_6 are open—the benchmark exists precisely because the framework does not predict their effect sizes.

4.6 Consistency as a measurable property

The reproducibility property P_3 has a direct quantitative target. FASTRIC reports run-level conformance standard deviations ranging from 0.00 (DeepSeek at L2–L4) to 0.36 (Phi4 at L4), and a 57% conformance collapse for one model at the highest formality level [3]. Our benchmark's five repetitions per configuration yield a per-configuration consistency estimate

$$\hat{\sigma}_{\text{config}} = \sqrt{\frac{1}{4}\sum_{i=1}^{5}(x_i - \bar{x})^2},$$

where x_i is the criterion score of repetition i and $\bar{x}$ their mean. With 5 repetitions the estimate carries 4 degrees of freedom; detecting a standard-deviation difference of the size FASTRIC observed between models (0.00 versus 0.36) is well within this design's resolution at $n = 5$ per cell, because the two models' distributions do not overlap in the reported data [3]. Whether pinning temporal context (P_3) reduces $\hat{\sigma}_{\text{config}}$ within a fixed model is the effect the ablation would isolate.

5. Results

All numbers in this section are computed in Section 4 with shown arithmetic, except where labeled projections; no empirical measurement is reported as this paper's own.

R1 (Compliance-lattice count, Section 4.1, exact). The six-property compliance space has $2^6 = 64$ classes; exactly 1 is compliant, so a uniformly drawn specification is compliant with probability $1/64 = 0.015625$. Fixing verifiability and narrow scope leaves $2^4 = 16$ classes, a 0.25 fraction.

R2 (Chain dominance, Section 4.2, proved). Under the stated model, $p_{\text{chain}} = (q + (1-q)s)^k \geq q^k = p_{\text{mono}}$, strict whenever the repair success $s > 0$; the advantage ratio is $\left(\frac{q+(1-q)s}{q}\right)^k$, exponential in k.

R3 (Worked success probabilities, Section 4.3, exact given assumptions). For $k = 5$, $q = 0.8$, $s = 0.5$: $p_{\text{mono}} = 0.32768$, $p_{\text{chain}} = 0.59049$, ratio ≈ 1.80. At $k = 10$: $p_{\text{mono}} = 0.1073741824$, $p_{\text{chain}} = 0.3486784401$, ratio ≈ 3.25.

R4 (Token cost, Section 4.4, projection under stated assumptions). With 800-token monoliths, 200-token sub-specifications, and the R3 probabilities: $T_{\text{mono}} = 2441.40625$ input tokens per success, $T_{\text{chain}} = 1111.111\ldots$ input tokens per success, ratio ≈ 2.20; successes per input token improve by a factor ≈ 3.96.

R5 (Benchmark size, Section 4.5, exact given design parameters). The one-at-a-time ablation over 6 properties, $T = 10$ assumed tasks, and $M = 3$ assumed models comprises 360 configurations and, with 5 repetitions, 1,800 runs.

R6 (Consistency resolution, Section 4.6, design capability). Five repetitions per configuration estimate per-configuration consistency with 4 degrees of freedom, sufficient to separate consistency levels as different as the 0.00 and 0.36 standard deviations reported across models in [3]; the effect of property P_3 on consistency within a model is the quantity the ablation would estimate.

The sensitivity of R3 and R4 to their assumptions is examined in Section 6.

6. Discussion

What the results do and do not claim. R1 and R5 are exact arithmetic over their stated models. R2 is a formal result whose scope is bounded by its modeling choices: independence of per-requirement success, identical q across requirements, and a repair loop whose success probability s is independent of the failure mode. R3 and R4 are exact arithmetic on top of assumed inputs ($q = 0.8$, $s = 0.5$, 800 and 200 tokens); they are projections, not measurements. R6 is a design capability, not an observed result.

Limitations. The result rests on the following assumptions, with sensitivity noted for each:

- *Independence of per-requirement success.* The composition result assumes that satisfying requirement R_i is independent of satisfying R_j within a single monolithic generation. If requirements are positively correlated (a competent generation tends to satisfy several at once), $p_{\text{mono}} = q^k$ underestimates the monolith and the 1.80-fold advantage shrinks; in the extreme of perfect correlation the monolith succeeds with probability $q = 0.8$ and the chain's advantage reduces to its repair benefit alone, $0.9/0.8 = 1.125$ per step. If requirements are negatively correlated (satisfying one crowds out another), the monolith is even worse than q^k and the advantage grows. The direction is therefore not fixed by the model; the benchmark of Section 4.5 would estimate the correlation directly from per-requirement criterion scores.
- *Identical q across requirements.* Real workloads mix easy and hard requirements. With heterogeneous q_i, the monolith succeeds with probability $\prod_i q_i$, which is unchanged in form, but the chain's per-step probabilities $q_i + (1-q_i)s$ differ per sub-specification and the advantage ratio becomes $\prod_i \frac{q_i+(1-q_i)s}{q_i} = \prod_i \left(1 + \frac{(1-q_i)s}{q_i}\right)$, which is dominated by the *hardest* requirement (smallest q_i). The chain's advantage is therefore concentrated exactly where the monolith is weakest, which strengthens the practical case but makes the single-number summary 1.80 an average, not a worst case.
- *Repair effectiveness s independent of failure mode.* We assume the repair attempt succeeds with probability $s = 0.5$ regardless of why the first attempt failed. If some failure modes are unrepairable (for example, an underspecified criterion that no output can satisfy), the effective s for those sub-specifications is 0 and the chain inherits the monolith's per-step probability there. The dominance result R2 still holds (strict inequality degrades to equality for those steps), but the numerical advantage shrinks proportionally to the fraction of unrepairable failure modes.

- *Token-size assumptions.* The 800-token monolith and 200-token sub-specification are assumptions, not measurements. The cost ratio is $\frac{800/q^k}{5\times 200/(q+(1-q)s)}$; generalizing with monolith size a and sub-specification size b, the ratio is $\frac{a}{5b} \times \frac{p_{\text{chain}}}{p_{\text{mono}}}$. The cost advantage reverses when $\frac{a}{b} < 5 \times \frac{p_{\text{mono}}}{p_{\text{chain}}} = 5 \times 0.555\ldots = 2.778\ldots$, that is, when a monolith is less than about 2.78 times the size of one sub-specification. Our assumed ratio $a/b = 4$ clears this threshold; a workload whose narrow sub-prompts cannot actually be written four times smaller would erase the cost advantage while keeping the success-probability advantage.
- *Input-tokens-only accounting.* We ignore output tokens and per-call fixed overhead. Output tokens favor the chain (each narrow generation produces less output, and failed monolithic attempts waste full-length outputs), while per-call overhead favors the monolith (one call versus $5.555\ldots$ expected calls). The net direction depends on the overhead-to-output-token price ratio, which no input in this paper fixes; we flag it as an open quantity rather than guessing a sign.
- *The lattice count assumes binary, independent properties.* P_1 through P_6 are treated as independently settable bits. In practice some properties are correlated by construction (a specification with an unparseable format, $P_6 = 0$, is hard to check for P_2), so the uniform distribution over $\{0,1\}^6$ that yields $p_{\text{compliant}} = 1/64$ is an idealization; the count 64 of classes is exact, but the probability interpretation is a projection under uniformity.

What would falsify the central claim. The composition result R2 is a theorem, so falsification attaches to its empirical premises, which the benchmark of Section 4.5 would test. The single most decisive observation is the ablation contrast for the pair $\{P_2, P_4\}$: if regressing first-try success on the six property indicators over the $1{,}800$ runs yields a coefficient on the narrow-scope indicator that is zero or negative after task and model fixed effects—or if the measured per-requirement success correlation is so strongly positive that the monolith's observed success rate matches the chain's within the design's resolution—then the independence mechanism behind R3's 1.80-fold advantage is absent in real models and the framework's central quantitative claim fails. A second falsifier is the cost side: if measured sub-specifications cannot be written smaller than the 2.778-fold threshold derived above, the projected 2.20-fold cost reduction disappears even if the success advantage survives.

Open questions. Three quantities are deliberately left open because no input in this paper fixes them. First, the per-criterion checker cost c (Section 4.4), which determines whether conformance checking is negligible or dominant in end-to-end latency. Second, the effect sizes of properties P_1, P_3, P_5, and P_6, which the framework does not predict and the benchmark exists to estimate. Third, the linear scoring model $E = \sum_{i=1}^{6} w_i p_i$ retained as a hypothesis: if the benchmark's regression finds near-equal coefficients w_i, a simple additive quality score may capture most of the predictive power without the composition machinery, and the framework should then be simplified.

7. Practical Implications and Engineering Pathways

Who would use the result and what changes. The immediate users are (a) prompt engineers and agent-framework maintainers who decide whether a task prompt is one document or a chain of sub-prompts, (b) evaluation teams who currently measure prompt quality by ad hoc human preference and would instead measure conformance against a criterion set V, and (c) platform teams building agent runtimes who need an auditable record of which instruction was violated. If the composition result holds empirically, the design decision that changes is concrete: any prompt carrying more than one independently checkable requirement is decomposed into a chain with bounded repair, and the decomposition decision is made by arithmetic on measured q, s, and prompt sizes rather than by taste. If the result does not hold, the same benchmark still delivers per-property effect sizes, which is the input any prompt-quality scoring scheme needs.

Readiness. This work is purely analytical and has not been peer reviewed; against the NASA Technology Readiness Level definitions it sits at TRL 1–2: the basic principles (the six predicates, the composition theorem, the cost model) are stated and the applications are sketched, but the concepts have not been exercised on real prompts or models. No

claim of implementation, deployment, or manufacturing readiness is made; the framework as specified here has never been run.

The smallest next test. The minimal experiment that would de-risk or falsify the central claim is a reduced version of the Section 4.5 benchmark: one task family, one model, the single ablation pair for narrow scope (P_4), with a compliant single-goal prompt and a two-goal monolith, $n = 5$ repetitions per configuration—$2 \times 5 = 10$ runs. The study would measure first-try success per requirement via the criterion set V and the pairwise correlation of per-requirement success within monolithic generations. The outcome that refutes the framework is a per-requirement correlation high enough that the monolith's observed success probability exceeds the chain's prediction p_{chain} from Section 4.2; the outcome that de-risks it is a measured correlation near zero, which would license scaling to the full 1,800-run design. A second minimal test, on the cost side, would measure the achieved size ratio a/b for real task decompositions against the 2.778 reversal threshold derived in Section 6.

Engineering substrate and constraints. Three components of the pathway have direct precedents in the cited literature, and the constraints that would bind first follow from them. First, conformance evidence needs a tamper-evident execution record: the Right to History framework provides append-only, integrity-checked agent action logs with sub-1.3 ms action latency and logarithmic-size Merkle proofs [4], which is fast enough that per-attempt criterion-result logging is unlikely to dominate latency, though the checker cost c itself remains the open quantity. Second, the checker must be cheap enough to run per attempt; the approximate-kernel-clustering result that over 90% kernel-computation speedup is achievable while preserving clustering quality on the Imagenet experiments [7] is the analogy for an approximate conformance checker whose verdict quality would have to be validated against an exact checker on the benchmark's outputs. Third, specification formality must be tuned per model, not globally: FASTRIC's finding that the optimal formality level differs sharply across three models (perfect 1.00 conformance at L2–L4 for one model, a peak of 0.90 at L3 with a 57% collapse at L4 for another, and no stable optimum for a third) [3] means a KERNEL dialect should expose formality as a per-model parameter rather than fixing it. The constraints that would bind first, in the order they bind: writing machine-checkable criteria v_i for tasks where success is partly a matter of judgment (this may be impossible for some task families, capping applicability); the prompt-size ratio a/b staying above the 2.778 threshold; and per-call overhead economics, which are unquantified here. Regulatory logging mandates for high-risk AI systems, which the Right to History work identifies as already emerging [4], would make the audit substrate a requirement rather than an option for deployed agents.

Task-family dialects. Following the dialect device of the QRscript specification, in which sublanguages carry implementation characteristics specific to an application field [5], we anticipate KERNEL dialects per task family: a code-generation dialect whose V criteria are test execution and static-analysis checks, in the spirit of TDD validation gates [2]; a retrieval-augmented dialect whose criteria include retrieved-chunk provenance checks, motivated by the demonstrated 2–3-token adversarial re-ranking of RAG pipelines [1]; and a tutoring-dialogue dialect whose criteria are FSM-state-transition checks in the style of FASTRIC's trace analysis [3]. Each dialect's criterion library is an engineering deliverable, not a contribution of this paper.

Cost and performance bounds (projections). Under the assumptions already stated in Section 4.4 ($q = 0.8$, $s = 0.5$, 800- and 200-token specifications, input tokens only), the projected operating characteristics of a chained deployment are: 1.80-fold higher first-try success, 2.20-fold lower input-token cost per success, and 3.96-fold higher successes per input token. These are projections, not measurements, and they inherit every sensitivity listed in Section 6; no additional cost, market, or performance figure is asserted.

Risks and misuse paths. A machine-checkable criterion set is only as good as its criteria: a specification whose V criteria are gameable (for example, a criterion satisfied by any syntactically valid output regardless of correctness) makes non-compliant behavior *look* conformant, and the tamper-evident record of [4] would then faithfully certify the wrong thing. A second misuse path is criterion overfitting: a team optimizing prompts against a fixed V may produce outputs

that pass V while failing the unstated intent, the prompt-side analogue of teaching to the test. Mitigation is governance of the criterion library itself—criteria should be reviewed as specifications, in the way the TDD-governance work treats tests as enforceable process constraints rather than afterthoughts [2]—but the residual risk of intent–criterion divergence is inherent to any verifiable specification and is not removed by this framework.

8. Conclusion

We formalized prompt effectiveness as six binary predicates on a prompt specification—simplicity, verifiability, reproducibility, narrow scope, explicit constraints, and structured format—and derived the quantitative consequences with fully shown arithmetic. The compliance lattice has $2^6 = 64$ classes, of which exactly one is compliant, so compliance must be built in by construction. Under stated independence and repair assumptions, chaining k narrow sub-prompts with bounded repair dominates the monolith with an advantage ratio $\left(\frac{q+(1-q)s}{q}\right)^k$, exponential in the number of requirements: 1.80-fold at $k = 5$ and 3.25-fold at $k = 10$ for the worked example, with a projected 2.20-fold reduction in input-token cost per successful completion and a 3.96-fold improvement in successes per input token. The one-at-a-time ablation benchmark that would test these claims comprises 360 prompt-model configurations and $1{,}800$ runs under a five-repetition protocol. The framework's contribution is to convert prompt design from phrasing advice into specification and conformance checking, with the composition theorem as its predictive core and the benchmark as the immediate next test; the same specification-over-heuristics framing may transfer to adjacent settings where natural-language instructions drive stochastic executors, insofar as those settings admit per-instruction verifiable success criteria.

Provenance and AI Involvement

- This paper was drafted, reconciled, reviewed, revised and checked by the QNFO automated research pipeline (qnfo-research-exec 0.55.12). Language models were tools in that process; they are not authors. The named author is accountable for the content.
- Models: drafting by openai/gpt-oss-120b, zai-org/glm-5.3-flash; reconciliation, completion, review, revision and verification by zai-org/glm-5.3-flash, openai/gpt-oss-120b, moonshotai/kimi-k2.6, with the QNFO model router as a fallback.
- Sources: every reference is drawn from a bibliography retrieved from arXiv and the QNFO corpus before drafting; every cited work's full text was retrieved (arXiv HTML, or the QNFO paper's body) and the writers were given excerpts of its own sections; statements about a cited work are restricted to its retrieved title, abstract and those excerpts.
- Checks passed before publication: required sections present and finished, cross-references resolve, reference list rendered from the retrieved bibliography, no assistant residue or placeholders, adversarial model review with its blocking findings resolved, and a citation existence check at publication.
- Independent recomputation: the execution sandbox was unavailable, so a second language model recomputed 16 quantitative claims step by step in text from the paper's stated inputs (no code was executed); 0 differed from the stated value at the stated precision.
- Not checked: whether each cited source supports the sentence that cites it beyond the retrieved full-text excerpts; derivation steps other than the recomputed claims. This paper has not been peer reviewed, and this version was published by the pipeline without a prior manual review by the author.

References

[1] DeRAG: Black-box Adversarial Attacks on Multiple Retrieval-Augmented Generation Applications via Prompt Injection. arXiv:2507.15042v1. https://arxiv.org/abs/2507.15042v1 [2] TDD Governance for Multi-Agent Code Generation via Prompt Engineering. arXiv:2604.26615v1. https://arxiv.org/abs/2604.26615v1 [3] FASTRIC: Prompt Specification Language for Verifiable LLM Interactions. arXiv:2512.18940v1. https://arxiv.org/abs/2512.18940v1 [4] Right to History: A Sovereignty Kernel for Verifiable AI Agent Execution. arXiv:2602.20214v1. https://arxiv.org/abs/2602.20214v1 [5] QRscript specification. arXiv:2403.04708v1. https://arxiv.org/abs/2403.04708v1 [6] A survey on formal specification and verification of separation kernels. arXiv:1508.07066v3. https://arxiv.org/abs/1508.07066v3 [7] Scalable Kernel Clustering: Approximate Kernel k-means. arXiv:1402.3849v1. https://arxiv.org/abs/1402.3849v1 [8] PALP: Prompt Aligned Personalization of Text-to-Image Models. arXiv:2401.06105v1. https://arxiv.org/abs/2401.06105v1 [9] https://papers.qnfo.org/papers/degree-bounds-as-a-first-exercise-in-isabelle-ml-metaprogramming-a-reconciled-st. QNFO: Degree Bounds as a First Exercise in Isabelle/ML Metaprogramming: A Reconciled Study of the poly_degree Command. [10] https://papers.qnfo.org/papers/exact-maximum-likelihood-quantum-decoding-beyond-treewidth-a-reconciled-structur. QNFO: Exact Maximum-Likelihood Quantum Decoding beyond Treewidth: A Structural Analysis of Rank-Decomposition Dynamic Programming. [11] https://papers.qnfo.org/papers/deductive-verification-of-discrete-time-quantum-walk-simulators-contracts-cost-b.
QNFO: Deductive Verification of Discrete-Time Quantum-Walk Simulators: Contracts, Cost Bounds, and Extensions to Graph-Generalized and Open-System Walks. [12] https://papers.qnfo.org/papers/the-capability-availability-mismatch-three-state-tool-semantics-and-forced-halt-. QNFO: The Capability–Availability Mismatch: Three-State Tool Semantics and Forced-Halt Conditions in LLM Agent Protocols.