Deterministic-Dimension Reduction for Reliable LLM Agents: A Design-Science Study of Spatial Construction and Edge Deployment

Paul Whitten
Rockwell Automation
pcwhitt1@rockwellautomation.com

Sharath Chander Reddy Baddam
Rockwell Automation
sbaddam@rockwellautomation.com

Li-Jen Chen
Rockwell Automation
LiJen.Chen@rockwellautomation.com

Abstract

Reliable language model agents in physically constrained domains often assign output dimensions to the model that are deterministic functions of task state. We introduce deterministic-dimension reduction, a design principle in which such dimensions are computed outside the model by a symbolic executor, reducing the model to language understanding and plan generation while eliminating entire error classes. We instantiate the principle in a 2.5-D spatial construction agent: the model plans in the horizontal plane while a deterministic executor computes vertical placement from column occupancy. Across the Build What I Mean benchmark, a component ablation, cross-model comparison, IGLU transfer, and edge deployment, we find that architectural decomposition improves reliability more than model scaling alone. A small, low-cost model with the decomposition architecture matches or exceeds a frontier model generating coordinates directly. We extract five design implications for reliable, cost-aware, and deployable agentic systems.

Keywords: large language models, reliable AI agents, neuro-symbolic systems, edge AI, spatial reasoning

1. Introduction

Language model agents operating in physically constrained domains face a structural reliability problem. These domains impose constraints that make some output dimensions computable rather than learnable, yet standard agent architectures ask the model to generate all output dimensions equally. The model then routinely produces errors in exactly the dimensions it did not need to generate at all.

Consider a block construction agent that must place colored blocks in a gravity-bound grid. The vertical coordinate of any new block is a deterministic function of column occupancy: a block placed at horizontal position (x, z) always lands at the first unoccupied height in that column. An LLM asked to generate full three-dimensional coordinates must nonetheless compute this value, and off-by-one height errors, duplicate placements, and miscounted stack heights are the predictable result (Bang et al., 2023; Yamada et al., 2024). The same structural issue arises in other constrained domains. In terrain-following path planning, altitude is a function of horizontal position and terrain data. In robotic assembly with geometric constraints, some degrees of freedom are determined by constraint equations once the free dimensions are fixed. In automated code generation, syntax rules and type constraints determine many tokens that language models nonetheless produce incorrectly.

We propose deterministic-dimension reduction, a design principle for agentic systems: identify output dimensions that are deterministic functions of task state and current output, remove them from the language model's generation space, and compute them using a symbolic executor. This reduces the model to language understanding, ambiguity resolution, and structured planning, while eliminating entire classes of output error.

We instantiate the principle in a 2.5-D spatial construction agent and evaluate it on the Build What I Mean (BWIM) benchmark (UvA LTL, 2026), a publicly scored competition requiring block structure construction from natural-language instructions. The agent plans exclusively in the horizontal plane while a deterministic executor computes vertical placement via column occupancy. This 2.5-D decomposition, named by analogy with Marr (2010)'s 2.5-D sketch, reduces the likelihood of y-coordinate errors.

An earlier conference paper presents the 2.5-D spatial construction method and a compact benchmark evaluation (Whitten et al., 2026). The present

paper extends that contribution by developing the deterministic-dimension reduction design principle, adding edge-deployment and latency evidence, IGLU cross-benchmark transfer, and design implications for reliable agentic systems.

This paper addresses three design questions. First, does removing deterministic output dimensions from an LLM agent improve reliability in a constrained construction domain, and which pipeline components contribute most? Second, does a decomposed architecture reduce dependence on large frontier models, and does it remain viable on cost-constrained and edge hardware? Third, does the principle transfer to a structurally different benchmark? The first two questions are addressed through controlled experiments on the BWIM benchmark. The third is addressed through an IGLU cross-benchmark transfer experiment.

The contributions of this paper are: (1) the deterministic-dimension reduction design principle, including an analysis of multiplicative error accumulation showing that, under the stated assumptions, a symbolic executor dominates the language model on deterministic output dimensions for any positive error rate, and a taxonomy of three categories of deterministic operations in agentic systems, (2) a 2.5-D construction agent as a controlled instantiation of the principle, (3) a decision-theoretic framework for underspecification handling under asymmetric scoring, (4) a multi-context evaluation spanning benchmark accuracy, component ablation, model-cost tradeoff, edge deployment, and cross-dataset transfer, and (5) design implications for reliable, cost-aware, and deployable agentic systems.

2. Background

2.1. LLM Spatial Reasoning Limitations

Large language models exhibit systematic weaknesses in spatial and directional reasoning. Yamada et al. (2024) benchmark LLMs on spatial tasks and find that coordinate tracking and relative positioning are persistent error sources. Bang et al. (2023) document directional and chain-reference failures in GPT-family models. LeCun (2022) observes that LLMs lack internal world models for enforcing physical constraints. These weaknesses are especially pronounced in 3D construction tasks, where the model must track block occupancy across three axes and chain height calculations across stacking operations.

2.2. Neuro-Symbolic and Decomposed Approaches

The neuro-symbolic tradition addresses these limitations by separating neural and symbolic computation. Yi et al. (2018) decompose visual question answering into a neural perception module and a symbolic execution engine. Wang et al. (2023) show that separating planning from execution improves zero-shot reasoning. Khot et al. (2023) split complex tasks into sub-problems handled by specialized modules. Prior work also organizes this space with taxonomies. Kautz (2022) distinguishes ways neural and symbolic components integrate, van Harmelen and ten Teije (2019) catalog compositional design patterns for hybrid systems, and Sarker et al. (2021) survey and classify recent work. These taxonomies categorize how neural and symbolic components combine, whereas our taxonomy categorizes deterministic operations by the source of the constraint, a complementary cut. Our approach extends this work by exploiting a dimensional constraint: the planning module operates in 2D while execution operates in 3D, a separation not explored in prior decomposed-prompting work.

2.3. Tool Use and Robotic Grounding

Schick et al. (2023) demonstrate that language models can learn to invoke external tools. Ichter et al. (2023) ground language model plans in robotic affordances, restricting the LLM to skills the robot can physically perform. Liang et al. (2023) have the LLM generate code that calls spatial APIs directly, offloading coordinate arithmetic to deterministic libraries. Huang et al. (2023) close the loop with environment feedback for replanning. Our approach shares the principle of restricting LLM outputs to what deterministic modules can process reliably, but identifies a specific dimensional constraint, the gravity axis, rather than grounding arbitrary skills or generating full executable code.

2.4. Offloading Deterministic Computation from LLMs

A parallel line of work argues that language models should not generate output values that are computable by a deterministic procedure, because delegating computable dimensions to the language model introduces avoidable error. Gao et al. (2023) show that separating language-level reasoning from arithmetic computation and executing the arithmetic in a Python interpreter eliminates computational errors entirely, with improvements of over 10 percentage points on average across numerical benchmarks. W.

Chen et al. (2023) make the same argument under the label "disentangling computation from reasoning," demonstrating that systematic errors in LLM-generated arithmetic are avoidable by design rather than reducible only by scale. Lyu et al. (2023) extend the principle to symbolic reasoning chains: the language model generates a structured plan and a deterministic solver executes it, providing a guarantee of faithful execution that language model sampling cannot provide.

At a broader architectural level, Karpas et al. (2022) propose routing LLM queries to discrete expert modules for reasoning steps that have known-correct computational procedures. Kambhampati et al. (2024) offer a near-formal argument that autoregressive generation cannot achieve correctness guarantees on planning tasks and that symbolic verifiers or executors are required at every constrained step. Dziri et al. (2023) show empirically and theoretically that transformer performance degrades with the compositional depth of a task, consistent with errors accumulating across chained generative steps.

These works collectively support the design intuition underlying deterministic-dimension reduction: removing a computable output dimension from the LLM's generation space eliminates its associated error class rather than reducing it. The present paper extends this line by providing a characterization of when and why this dominance holds, introducing the free-dimension and deterministic-dimension decomposition as an explicit design vocabulary, and applying it to a constrained construction domain with controlled ablation evidence.

2.5. Design Science

Design science research produces artifact knowledge: design principles, methods, and constructs that expand what is achievable in practice (Hevner et al., 2004). The design-science framing is appropriate here because the primary contribution is not a benchmark result but a design principle applicable beyond any single domain or benchmark.

3. Deterministic-Dimension Reduction

3.1. Design Principle

Let the agent's output for a given task instance be a tuple $o = (v_{\text{free}}, v_{\text{det}})$, where $v_{\text{free}} \in D_{\text{free}}$ denotes the *free dimensions* and $v_{\text{det}} \in D_{\text{det}}$ denotes the *deterministic dimensions*. The free dimensions are those that require language understanding, disambiguation, or reasoning about intent: they cannot be computed from task state alone and constitute the genuinely generative part of the task. The deterministic dimensions are those whose correct value is fully determined by task state s and the free dimensions already chosen:

$$v_{\text{det}} = f(s,\, v_{\text{free}})$$

for some computable function f. In block construction, v_{free} is the horizontal position (x, z) and block color c, which require understanding the natural-language instruction. v_{det} is the vertical coordinate y, which follows from column occupancy regardless of instruction content.

In a standard agent architecture the language model generates both v_{free} and v_{det}. This is unnecessary for the deterministic dimensions and costly in practice. A language model sampling v_{det} from its output distribution produces the correct value with probability $1 - \varepsilon$ for some model-specific error rate $\varepsilon > 0$. A symbolic executor computing $f(s, v_{\text{free}})$ produces the correct value with probability 1, provided v_{free} is correct and f is implemented correctly. For any $\varepsilon > 0$ the executor strictly dominates the language model on the deterministic dimensions regardless of dependence structure, since the language model's probability of producing all deterministic values correctly remains strictly below 1.

Under independence of errors across n sequential decisions in a task, the language model's probability of producing all n deterministic values correctly is $(1 - \varepsilon)^n$, which compounds rapidly. A structure requiring tens of placement decisions suffers substantial cumulative error even at modest per-decision ε. Under positive error correlation, which is the expected case when a systematic misunderstanding affects multiple placements, $P(\text{all correct})$ exceeds $(1-\varepsilon)^n$ but remains below 1. The two assumptions underlying this argument are: (i) the symbolic executor f is implemented correctly, which is verifiable by unit testing and holds by construction when f implements a closed-form rule such as column occupancy, and (ii) generating v_{det} jointly with v_{free} does not improve the distribution over v_{free}, an assumption discussed further in the limitations.

Deterministic-dimension reduction restricts the model's output schema to D_{free} and delegates D_{det} to a symbolic executor. This has three further consequences beyond error elimination. The model's output space shrinks, reducing the burden on context and sampling. Smaller models that perform adequately on D_{free} can match larger models that must additionally produce D_{det}. The symbolic executor is independently testable and verifiable, separating evaluation of the language understanding component from evaluation of the execution component.

3.2. Applicability Conditions

The principle applies when two conditions hold: (a) one or more output dimensions are deterministic functions of task state and the remaining output, and (b) the function f is computable in closed form or by a finite algorithm. When both conditions hold, the dominance argument above guarantees that a correct symbolic executor weakly dominates the language model on those dimensions regardless of the model's error rate. The only remaining question is whether the engineering cost of implementing f is justified by the expected error reduction.

In practice, the principle is most valuable when the language model exhibits a non-negligible error rate ε on the deterministic dimensions despite producing correct free-dimension outputs. In gravity-bound construction, y-coordinates satisfy (a) and (b) trivially, and LLMs routinely produce y-errors even when horizontal placement is correct. In UAV altitude control over terrain, altitude satisfies (a) and (b) given terrain data, and altitude errors are a known failure mode in LLM path descriptions. In code generation, bracket matching and type consistency satisfy (a) and (b), and token-level syntax errors in LLM output demonstrate non-negligible ε. The same pattern applies to any system with physical or logical constraints that fix some output dimensions once the free dimensions are determined.

3.3. A Taxonomy of Deterministic Operations

Agent domains exhibit at least three categories of deterministic operations, distinguished by the source of the constraint.

Some constraints follow from laws of the domain. Gravity determines vertical placement in block construction. Terrain data determines altitude in path planning. Kinematic equations limit reachable positions in robotic assembly. These constraints apply globally to every task instance. The LLM adds no value in computing them and produces systematic errors when asked to do so.

Other constraints are codified rules that follow deterministically from observable input properties. In tabular machine learning, the correct handling of a null-valued boolean column is determined by the column's dtype and null statistics, not by language model judgment. In code generation, bracket matching and type-consistency rules are similarly determined by language grammar rather than by content.

A third category consists of protocols with known-correct execution sequences. Git fetch must precede checkout. Build must precede test. Test must precede merge. These sequences are fully determined before the agent runs, require no language model reasoning, and produce omission and sequencing errors when delegated to the LLM. Deterministic execution ordering is long formalized in software engineering, where build systems compute a correct order as a topological sort of a dependency graph (Feldman, 1979; Mokhov et al., 2018) and workflow systems catalog control-flow ordering constraints (van der Aalst et al., 2003).

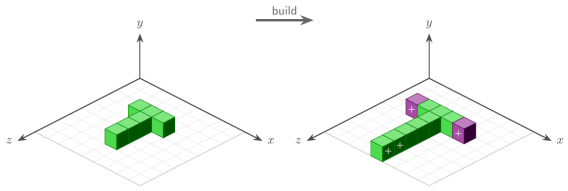

Figure 1. A representative BWIM benchmark round involving T-shape geometry.

4. Artifact: 2.5-D Construction Agent

4.1. Task and Benchmark

The BWIM benchmark defines a block construction task on a discrete grid $\mathcal{G} = \{0, \ldots, 8\} \times \{0, \ldots, 4\} \times \{0, \ldots, 8\}$ where each cell is either empty or occupied by a colored block. In each round, the builder agent receives a natural-language instruction and a starting grid state, and must produce a target grid state. The agent may ask one clarification question before building. Block-level F1 treats each cell in the output grid as a prediction, computing precision and recall over the set of occupied cells with correct color relative to the target grid. The scoring function awards $+10$ for a correct build, -10 for an incorrect build, and -5 per question, creating an expected value of $+5$ for a correct build with one question and -5 for an incorrect build with one question.

4.2. Instantiating the Principle

Figure 1 shows a representative benchmark round. The agent receives a natural-language instruction referencing an existing green T-shape and must identify its stem and arms, extend the stem by two blocks, and place a purple block at each arm endpoint. The 2D plan specifies only horizontal positions and colors. Vertical placement is computed by the deterministic executor from column occupancy.

The BWIM grid is a 2.5-D domain in the sense of

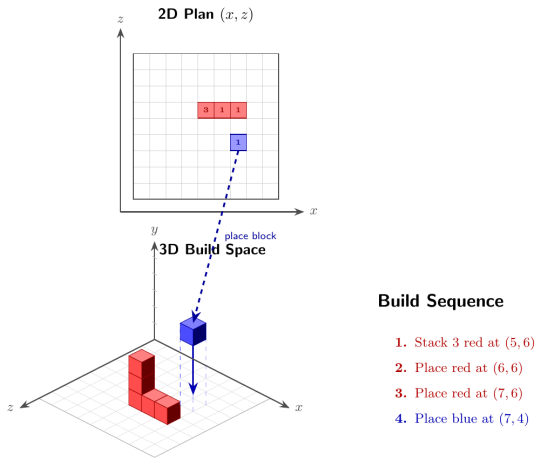

Figure 2. 2.5-D decomposition. The LLM planner generates 2D plans with (x, z) coordinates and action types. The deterministic executor computes vertical placement via column occupancy (Equation 1).

Marr (2010): the vertical axis is not a free variable but a deterministic function of horizontal position and column occupancy. A block at (x, y, z) can only exist if $y = 0$ or a block exists at $(x, y-1, z)$. The vertical coordinate of any new block is therefore:

$$y^*(x, z, G) = \min\{y \in \{0, \ldots, 4\} \mid (x, y, z) \notin \mathrm{dom}(G)\} \quad (1)$$

where $\mathrm{dom}(G)$ is the set of occupied cells. Applying deterministic-dimension reduction assigns y^* to the symbolic executor and restricts the LLM to horizontal positions (x, z) and block colors c.

The LLM produces a plan $P = \langle a_1, \ldots, a_k \rangle$ where each action a_i specifies an action type (stack, place-relative, extend-row), a horizontal position (x_i, z_i), a color c_i, and an optional count n_i. No y-coordinate appears anywhere in the plan format.

4.3. Pipeline

The agent processes each instruction through six stages: parse, analyze, plan (LLM call), verify, execute (deterministic), and format. The structure analyzer pre-computes geometric primitives in the starting grid (rows, stacks, L-shapes, T-shapes) and injects named shape descriptions into the planner prompt, so the LLM reasons over geometry rather than raw coordinate lists. Nine worked examples in the system prompt (Wei et al., 2022) cover chains, L-shapes, T-shapes, and edge placements. No example includes y-coordinates. The spatial executor processes each action, computes y via Equation 1, and chains positional context across steps. No LLM call is involved at execution time.

A rule-based plan verifier runs four deterministic correction passes before execution: direction consistency, endpoint cap correction, T-shape axis correction, and stacking plausibility. These are analogous to peephole optimization (McKeeman, 1965): known-bad output patterns in the LLM plan are rewritten deterministically without a second LLM call. A same-color skip-forward rule in the executor checks whether an `extend_row` start position is already occupied by a block of the same color. If so, it advances the start one grid step in the extension direction, resolving the ambiguity between "extend from this block" and "stack on this block" without an additional model call.

4.4. Underspecification Handling

Many BWIM instructions omit color or block count. The scoring function creates an asymmetric decision problem. Let p denote the probability of a correct unaided guess. The expected values are:

$$\mathrm{EV}_{\mathrm{guess}}(p) = 20p - 10 \quad (2)$$

$$\mathrm{EV}_{\mathrm{ask}} \approx 20 - 15 = 5 \quad (3)$$

assuming the upstream architect, itself an LLM agent with access to the target structure, answers correctly. This assumption is discussed further in Section 6. Setting equal and solving yields indifference threshold $p^* = 3/4$. The agent asks when $p < 0.75$. Color inference achieves approximately 80% on inferable cases (above p^*); count inference achieves approximately 65% (below p^*), so the agent asks whenever a count is underspecified and no question has been used in the current round. Questions are color-specific to allow the upstream architect to identify the correct stack.

4.5. Adaptive Prompt Enrichment

Certain spatial concepts, including chain references, L-shape extensions, and T-shape junctions, produce systematic LLM errors. The agent scans each instruction against 15 pattern-matching rules. When a rule fires, a targeted correction with a worked example is injected into the prompt before the LLM call. The rules are independently testable and compose additively. This methodology is applicable to any domain where LLMs make systematic, input-predictable errors: identify recurring failure modes, classify by input trigger, write targeted corrections with examples, inject dynamically, and validate against regressions.

The benchmark task, including the architect (green) agent and the scoring harness, is publicly available

(UvA LTL, 2026). The extended report (Whitten et al., 2026) describes the pipeline, and the complete source code, all planner prompts, the 15 adaptive enrichment rules, evaluation scripts, and scoring logs are released in the public implementation repository (Whitten, 2026) at tag v1.0.5, supporting independent reproduction and reuse of the method.

5. Evaluation Design

The three design questions are addressed through four controlled conditions. The first is a BWIM 160-round benchmark with GPT-4o-mini, $n = 12$ independent runs, plus component ablation ($n = 7$ per condition) removing one pipeline component at a time from the full system. The second is a head-to-head comparison of GPT-4o-mini and GPT-4o, $n = 12$ and $n = 6$ respectively, on the identical frozen pipeline. All prompt engineering and enrichment rules were developed against GPT-4o-mini due to cost constraints. GPT-4o runs use the same pipeline with no model-specific tuning.

For cross-benchmark transfer, 500 gravity-compatible tasks from the IGLU collaborative building dataset (Kiseleva et al., 2022) compare a bare coordinate-output prompt against the 2.5-D decomposed prompt at temperature 0, with no enrichment rules, plan verifier, or underspecification handling. This isolates the decomposition contribution on an independent benchmark ($11 \times 9 \times 11$ grid, different instructions). For edge deployment, Nemotron-3-Super-120B-A12B-NVFP4 (NVIDIA, 2024) runs locally on an NVIDIA Jetson Thor AGX Developer Kit (NVIDIA, 2025) via vLLM (Kwon et al., 2023) (Blackwell GPU, Arm Neoverse V3AE CPU) under two conditions: full reasoning with 9 examples, and low-effort reasoning with 13 examples designed to exceed the model's 8,320-token prefix cache page size. Error attribution classifies all 1,920 GPT-4o-mini rounds as builder-origin or architect-origin.

All primary runs use a single frozen code commit. Statistical significance uses Welch two-sample t-tests. Run-to-run variance reflects LLM sampling non-determinism rather than development changes.

6. Findings

Table 1 shows the ablation study. Removing 2.5-D decomposition produces the largest accuracy drop (28.7 percentage points, $p < 0.0001$), reducing accuracy to 65.9%. Underspecification questions contribute the second largest effect (-17.9 pp), consistent with the $p^* = 0.75$ threshold. Adaptive prompt enrichment accounts for -11.3 pp and the skip-forward rule for -3.9 pp. The plan verifier shows no statistically significant effect (-0.6 pp, $p = 0.36$), suggesting enrichment rules prevent most errors the verifier would otherwise target.

Table 1. Ablation study ($n=7$ per condition, GPT-4o-mini). Δ is relative to the full system.

Configuration	Mean	σ	Δ	p
Full system	94.6%	0.72%	–	–
− 2.5-D decomp.	65.9%	1.24%	−28.7 pp	<0.0001
− Underspec. qs	76.7%	0.70%	−17.9 pp	<0.0001
− Enrichment	83.3%	1.29%	−11.3 pp	<0.0001
− Skip-forward	90.7%	1.22%	−3.9 pp	<0.0001
− Plan verifier	94.0%	1.44%	−0.6 pp	0.36

Table 2 shows the model comparison. GPT-4o-mini outperforms GPT-4o by 4.3 percentage points ($p < 0.0001$, 95% CI $[3.6, 4.9]$ pp), a result opposite to what model scaling alone would predict. This finding should be interpreted with the caveat that prompt engineering was developed against GPT-4o-mini. Had development been conducted against GPT-4o from the start, GPT-4o results could plausibly be higher. However, GPT-4o with the untuned pipeline reaches 90.3%, well above the best competing system's 76.3% using GPT-4o without a decomposition pipeline (Sandoval Higuera et al., 2026), indicating that the pipeline provides value independent of model-specific tuning.

A cursory check supports cross-model generality. Running the identical untuned pipeline through Azure AI Foundry, DeepSeek-V3.2 reached 95.6 to 96.2% over six runs, Llama-4-Maverick a reproducible 91.2%, and Kimi-K2.5 about 84%, with no model-specific tuning. These are preliminary single-batch trials, and the logs are in the public repository.

GPT-4o incurs approximately 16.7× higher per-token cost than GPT-4o-mini. Assuming comparable token counts across models on the identical pipeline, relative cost is governed by this per-token price ratio, which is dimensionless and independent of any single price sheet. This assumption may not hold in general, since different models emit different numbers of tokens for the same input, especially models that perform explicit reasoning or thinking. As one concrete instantiation, at the time of the experiment a 160-round GPT-4o-mini run cost approximately $0.07 versus $1.10 for GPT-4o, a 16× cost difference for 4.3 percentage points lower accuracy on this pipeline. Normalized by outcome, the GPT-4o-mini pipeline yields approximately 16× more successful builds per unit cost, a cost-effectiveness comparison that does not depend on absolute prices and remains valid when token usage differs across models (L. Chen et al., 2023).

Table 2. Model comparison on the frozen pipeline.

Model	n	Acc.	Score	F1
GPT-4o-mini	12	94.6%	+947	.987
GPT-4o	6	90.3%	+810	.975
Nemotron-3†	6	96.0%	+993	.987

†Local Jetson Thor AGX, Blackwell GPU, full reasoning.

Table 3. Edge latency: Nemotron-3 on Jetson Thor ($n = 6$, seconds).

Configuration	Med.	Mean	P95	Acc.
Full reasoning	48.2	59.0	122.9	96.0%
Low-effort + cache	17.1	19.7	37.0	95.6%

On 500 IGLU tasks, 2.5-D decomposition improved mean block-level F1 from 0.723 to 0.798 (paired $t(499) = 5.76$, $p < 10^{-8}$, $d = 0.26$, 95% CI $[0.050, 0.101]$). The effect size is small ($d = 0.26$), confirming that transfer validates the dimensional reduction principle alone, not the full pipeline. No enrichment rules, plan verifier, or underspecification handling were used. This confirms the principle generalizes to an independent benchmark with a larger grid and different instructions.

The baseline Nemotron-3 evaluation ($n = 6$) used the identical pipeline as GPT-4o-mini with full reasoning enabled, achieving 96.0% accuracy ($\sigma = 0.31\%$, +993 mean score), slightly exceeding the GPT-4o-mini cloud mean ($p < 0.001$). The accuracy advantage may reflect the model's explicit reasoning mode (120B parameters, 12B active via mixture of experts) combined with the elimination of network latency variability in local deployment. This demonstrates that the architecture transfers to a locally hosted open-weight model on edge hardware without prompt modifications.

A follow-on evaluation enabled low-effort reasoning, an inference setting that reduces the number of tokens the model spends on internal deliberation, and expanded the system prompt from 9 to 13 examples, increasing prompt length from approximately 7,400 to 9,140 tokens. This exceeds the model's prefix cache page size (8,320 tokens in vLLM for this model), enabling cache hits that are not possible when the prompt falls short of a full page. The resulting cache hit rate is approximately 85%. Mean per-request latency fell from 59.0 seconds to 19.7 seconds (median from 48.2s to 17.1s), a 3.0× reduction at an accuracy cost of 0.4 percentage points (95.6%).

Across all 1,920 GPT-4o-mini rounds, 104 produce incorrect structures (5.4% failure rate). Of these, 57 (54.8%) originate in the upstream architect agent: 55 cases involve the architect reporting an incorrect color in response to a clarification question, and 2 involve incorrect count information. The remaining 47 failures are builder errors: 25 spatial reasoning mistakes and 22 color-position errors in fully-specified instructions.

Excluding architect-origin failures, the builder pipeline achieves 97.6% accuracy (1,873/1,920 rounds). The 3.0 percentage point gap between raw accuracy (94.6%) and builder-only accuracy (97.6%) quantifies the portion attributable to an upstream LLM dependency that the builder cannot independently detect or verify.

7. Design Implications

The findings converge on design knowledge for reliable agentic systems in constrained domains.

(1) Identify all output dimensions that are deterministic functions of task state, classifying them as geometric constraints, procedural knowledge constraints, or workflow state management constraints. Remove them from the LLM output schema and compute them with a symbolic executor. Eliminating a dimension removes its error class entirely, which is qualitatively different from prompt tuning or model scaling. The 28.7 pp accuracy drop when the 2.5-D decomposition is removed confirms that the architectural boundary is the dominant reliability factor. The IGLU transfer confirms generality on an independent benchmark. The taxonomy in Section 3 distinguishes three categories of such dimensions.

(2) Once deterministic dimensions are removed, assign language understanding and plan generation to the LLM, and assign coordinate arithmetic and constraint satisfaction to deterministic code. Smaller models that handle the free dimensions adequately can then match larger models that must also handle the deterministic dimensions. GPT-4o-mini with the decomposition architecture matches or exceeds GPT-4o generating coordinates directly at 16× lower cost. Nemotron-3 on edge hardware reaches 96.0% accuracy.

(3) Architectural decomposition can provide a more favorable cost-accuracy tradeoff than model scaling on constrained tasks. In this evaluation, the decomposed pipeline costs $0.07 per run at 94.6% accuracy versus $1.10 at 90.3% for the frontier model baseline.

(4) When underspecification is possible, derive the indifference threshold from the scoring function rather than using a fixed heuristic. Unmodeled underspecification leads to systematic guessing losses. The $p^* = 0.75$ threshold derived from expected-value analysis correctly identifies when asking dominates guessing on BWIM. The threshold depends on task-specific costs and must be rederived for new domains.

(5) In multi-agent pipelines, measure and attribute each LLM component's contribution to failure separately during evaluation. Raw accuracy conflates component quality. In this evaluation, 54.8% of failures originate in the architect LLM. Raw accuracy underestimates builder pipeline quality by 3.0 pp.

8. Limitations

The primary evaluation uses a single 160-round benchmark. The IGLU transfer confirms the dimensional reduction principle on an independent dataset, but the full pipeline has been validated on only one benchmark. The 2.5-D decomposition depends on a gravity constraint that makes vertical coordinates computable. Tasks with free-form 3D placement or rotation degrees of freedom require a different decomposition.

A central limitation is that all enrichment rules and prompt examples were developed against GPT-4o-mini, while GPT-4o ran on the same pipeline with no model-specific tuning. The comparison therefore cannot separate the effect of the architecture from the effect of prompt optimization, and the GPT-4o result may be understated. A fair test would tune each model pipeline to a similar degree. The plan verifier null result ($p = 0.36$, $n = 7$) may reflect insufficient statistical power rather than a true null effect.

The 15 enrichment rules were developed by analyzing BWIM failure modes. Their coverage of failure modes in other instruction distributions is unknown.

The strict dominance argument assumes the free and deterministic dimensions are independent in the sense that generating v_{det} jointly with v_{free} does not improve the distribution over v_{free}. Whether joint generation of all output dimensions could benefit the free dimensions in some settings is an open question not addressed by this work or by prior literature.

9. Conclusion

We presented deterministic-dimension reduction as a design principle for reliable LLM agents: identify output dimensions that are deterministic functions of task state, remove them from the LLM's output space, and compute them symbolically. We instantiated the principle in a 2.5-D spatial construction agent and evaluated it across benchmark accuracy, component ablation, cross-model comparison, IGLU transfer, and edge deployment.

The central finding is that architectural decomposition improves reliability more than model scaling on constrained tasks. A small, low-cost model with the decomposition architecture matches or exceeds a frontier model generating coordinates directly. The principle transfers to edge hardware and to a structurally different benchmark. Error analysis shows that a substantial fraction of residual failures originate in upstream LLM dependencies, a finding with direct implications for multi-agent system design.

The evidence here, one primary domain with a controlled ablation and one transfer benchmark, supports the conjecture that the design principle applies to other physically or logically constrained domains with deterministically computable output dimensions, including terrain-following navigation, robotic assembly, and code generation under structural constraints, where injecting deterministic structure into code agents has independently been shown to reduce run-to-run variance and improve reliability (Lin et al., 2026). We do not claim general applicability beyond what these two benchmarks show. The boundary between deterministic and reasoning dimensions is not always sharp, and the principle breaks down when its governing assumption does not hold. In the IGLU transfer, some target structures contained floating blocks whose height gravity does not fix, so the evaluation used only gravity-compatible tasks, and recognizing where a constraint fails is itself a reasoning task. The location of the boundary also depends on model capability. As base models improve and make fewer errors on a computable dimension, the reliability gained by delegating that dimension to a deterministic executor narrows, so the value of the decomposition is largest for smaller and lower-cost models, while a frontier model may reach comparable accuracy at substantially higher token cost. The taxonomy of three categories, geometric constraints, procedural knowledge constraints, and workflow state management constraints, provides a structure for identifying decomposition opportunities in new domains. The primary design lever is not model selection but responsibility boundary placement.

References

Bang, Y., Cahyawijaya, S., Lee, N., Dai, W., Su, D., Wilie, B., Lovenia, H., Ji, Z., Yu, T., Chung, W., et al. (2023). A multitask, multilingual, multimodal evaluation of ChatGPT on reasoning, hallucination, and interactivity. *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association*

for Computational Linguistics, 675–718. https://doi.org/10.18653/v1/2023.ijcnlp-main.45

Chen, L., Zaharia, M., & Zou, J. (2023). FrugalGPT: How to use large language models while reducing cost and improving performance.

Chen, W., Ma, X., Wang, X., & Cohen, W. W. (2023). Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=YfZ4ZPt8zd

Dziri, N., Lu, X., Sclar, M., Li, X. L., Jiang, L., Lin, B. Y., Welleck, S., West, P., Bhagavatula, C., Le Bras, R., Hwang, J. D., Sanyal, S., Ren, X., Ettinger, A., Harchaoui, Z., & Choi, Y. (2023). Faith and fate: Limits of transformers on compositionality. *Advances in Neural Information Processing Systems*, *36*. https://proceedings.neurips.cc/paper_files/paper/2023/hash/deb3c28192f979302c157cb653c15e90-Abstract-Conference.html

Feldman, S. I. (1979). Make — a program for maintaining computer programs. *Software: Practice and Experience*, *9*(4), 255–265. https://doi.org/10.1002/spe.4380090402

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., & Neubig, G. (2023). PAL: Program-aided language models. *Proceedings of the 40th International Conference on Machine Learning*, 10764–10799. https://proceedings.mlr.press/v202/gao23f.html

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, *28*(1), 75–105. https://doi.org/10.2307/25148625

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., et al. (2023). Inner monologue: Embodied reasoning through planning with language models. *Proceedings of the 6th Conference on Robot Learning*, *205*, 1769–1782.

Ichter, B., Brohan, A., Chebotar, Y., Finn, C., Hausman, K., Herzog, A., Ho, D., Ibarz, J., Irpan, A., Jang, E., Julian, R., et al. (2023). Do as I can, not as I say: Grounding language in robotic affordances. *Proceedings of the 6th Conference on Robot Learning*, *205*, 287–318.

Kambhampati, S., Valmeekam, K., Guan, L., Verma, M., Stechly, K., Bhambri, S., Saldyt, L., & Murthy, A. (2024). Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. *Proceedings of the 41st International Conference on Machine Learning*, 22895–22907.

Karpas, E., Abend, O., Belinkov, Y., Lenz, B., Lieber, O., Ratner, N., Shoham, Y., Bata, H., Levine, Y., Leyton-Brown, K., Muhlgay, D., Rozen, N., Schwartz, E., Shachaf, G., Shalev-Shwartz, S., Shashua, A., & Tennenholtz, M. (2022). MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning.

Kautz, H. A. (2022). The third AI summer: AAAI robert s. engelmore memorial lecture. *AI Magazine*, *43*(1), 105–125. https://doi.org/10.1002/aaai.12036

Khot, T., Trivedi, H., Finlayson, M., Fu, Y., Richardson, K., Clark, P., & Sabharwal, A. (2023). Decomposed prompting: A modular approach for solving complex tasks. *International Conference on Learning Representations*.

Kiseleva, J., Li, Z., Aliannejadi, M., Mohanty, S., ter Hoeve, M., Burtsev, M., Skrynnik, A., Zholus, A., Panov, A., Srinet, K., et al. (2022). Interactive grounded language understanding in a collaborative environment: IGLU 2021. *NeurIPS 2021 Competitions and Demonstrations Track*, 146–161.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., & Stoica, I. (2023). Efficient memory management for large language model serving with PagedAttention. *Proceedings of the 29th Symposium on Operating Systems Principles*, 611–626. https://doi.org/10.1145/3600006.3613165

LeCun, Y. (2022). *A path towards autonomous machine intelligence* (tech. rep.). New York University and Meta. https://openreview.net/forum?id=BZ5a1r-kVsf

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., & Zeng, A. (2023). Code as policies: Language model programs for embodied control. *2023 IEEE International Conference on Robotics and Automation*, 9493–9500. https://doi.org/10.1109/ICRA48891.2023.10160591

Lin, Z., Zhou, M., Yang, Y., & Li, L. (2026). How much static structure do code agents need? A study of deterministic anchoring. *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*.

Lyu, Q., Havaldar, S., Stein, A., Zhang, L., Rao, D., Wong, E., Apidianaki, M., & Callison-Burch, C. (2023). Faithful chain-of-thought reasoning. *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, 305–329. https://doi.org/10.18653/v1/2023.ijcnlp-main.20

Marr, D. (2010). *Vision: A computational investigation into the human representation and processing of visual information*. MIT Press.

McKeeman, W. M. (1965). Peephole optimization. *Communications of the ACM*, *8*(7), 443–444. https://doi.org/10.1145/364995.365000

Mokhov, A., Mitchell, N., & Peyton Jones, S. (2018). Build systems à la carte. *Proceedings of the ACM on Programming Languages*, *2*(ICFP), 1–29. https://doi.org/10.1145/3236774

NVIDIA. (2024). *Nemotron-3-super-120b-a12b* [Model card and weights]. Retrieved June 10, 2026, from https://huggingface.co/nvidia/NVIDIA-Nemotron-3-Super-120B-A12B-NVFP4

NVIDIA. (2025). *Jetson thor* [Product specification]. Retrieved June 10, 2026, from https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-thor/

Sandoval Higuera, D. S., Romero Mahecha, S. A., Henao Garcia, J. A., & Garcia Sanchez, A. F. (2026). *Purple builder agent: BWIM competition agent* [Team Manada Werewolve, AgentBeats Phase 2]. Retrieved June 10, 2026, from https://github.com/hisandan/Purple-Agent-Beats-build-what-i-mean

Sarker, M. K., Zhou, L., Eberhart, A., & Hitzler, P. (2021). Neuro-symbolic artificial intelligence: Current trends. *AI Communications*, *34*(3), 197–209. https://doi.org/10.3233/AIC-210084

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, *36*. https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html

UvA LTL. (2026). *Build what I mean* [Interactive grounded language understanding benchmark]. Retrieved June 10, 2026, from https://github.com/ltl-uva/build_what_i_mean

van Harmelen, F., & ten Teije, A. (2019). A boxology of design patterns for hybrid learning and reasoning systems. *Journal of Web Engineering*, *18*(1-3), 97–123. https://doi.org/10.13052/jwe1540-9589.18133

van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., & Barros, A. P. (2003). Workflow patterns. *Distributed and Parallel Databases*, *14*(1), 5–51. https://doi.org/10.1023/A:1022883727209

Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R. K.-W., & Lim, E.-P. (2023). Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2609–2634. https://doi.org/10.18653/v1/2023.acl-long.147

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, *35*, 24824–24837. https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html

Whitten, P. (2026). *AgentWhetters-bwim: 2.5-D construction agent implementation* [Version tag v1.0.5]. Retrieved September 13, 2026, from https://github.com/paulwhitten/AgentWhetters-bwim

Whitten, P., Chen, L.-J., & Baddam, S. (2026). 2.5-D decomposition for LLM-based spatial construction. *NAECON 2026 - IEEE National Aerospace and Electronics Conference*, 114–119. https://doi.org/10.1109/NAECON70028.2026.11674959

Yamada, Y., Bao, Y., Lampinen, A. K., Kasai, J., & Yildirim, I. (2024). Evaluating spatial understanding of large language models. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=xkiflfKCw3

Yi, K., Wu, J., Gan, C., Torralba, A., Kohli, P., & Tenenbaum, J. B. (2018). Neural-symbolic VQA: Disentangling reasoning from vision and language understanding. *Advances in Neural Information Processing Systems*, *31*. https://proceedings.neurips.cc/paper_files/paper/2018/hash/5e388103a391daabe3de1d76a6739ccd-Abstract.html