What Makes a Virtual ISA Agent-Programmable?

A Controlled Study of Agent-Facing Documentation for StableHLO

Luxin Zhang
Independent Researcher
luxin.zhang92@outlook.com

***Abstract*—AI agents increasingly write code at the virtual instruction-set architecture (ISA) layer, while benchmarks score the agent, not the *interface* the ISA exposes. Isolating that interface is hard because documentation variants must carry the same facts, examples and length. We fix one intermediate representation (IR), StableHLO, with a pinned verifier and executor, the tasks and a harness-guided generate-and-repair loop, and vary only how documentation reaches the agent: none, syntax only, prose, a length-matched JSON contract with the same facts and examples, or retrieval through a tool. With a protocol frozen beforehand, we evaluated 267 held-out tasks (114 synthetic, 153 from KernelBench) with independent oracles and three GPT-4.1 snapshots (5,005 pre-registered trials; the largest model on 91 tasks). Syntax-only documentation added 16 to 33 percentage points (pp) of first-attempt correctness over none. With matched content, the tested JSON contract scored below prose for every GPT-4.1 model (-6.0, -7.9, -12.1 pp; significant for the two larger), reversing our earlier, unmatched comparison. On-demand access trailed upfront delivery by 4–32 pp: the strongest model never called the tool unless forced, and partial upfront documentation suppressed lookup for nano. Over 99% of observed failures were rejected statically: well-formedness was the observed bottleneck. Post-hoc checks under frozen addenda (14,002 trials) showed that the presence effect held for six of seven models and that curated facts beat the raw 91k-token specification by 12 to 26 pp, whereas the JSON penalty and the uptake failure were large only for GPT-4.1 mini and 4.1.**

***Index Terms*—virtual ISA, StableHLO, MLIR, LLM agents, agent-facing interfaces, documentation, code generation, benchmarking**

I. Introduction

A virtual ISA decouples programs from rapidly changing hardware: front ends target the virtual layer and back ends absorb the churn [1], [2]. Increasingly, the producer at that layer is an AI agent that writes, repairs and lowers code, which makes the presentation of the ISA to a non-human programmer a design dimension in its own right. Existing benchmarks score models on GPU kernels [3]–[5], LLVM IR optimization [6] or unfamiliar accelerator ISAs [7]; they answer "how good is the agent?" for a given interface. We ask the complementary question: *for a fixed agent and a fixed IR, which properties of the agent-facing interface change whether the agent writes correct programs?*

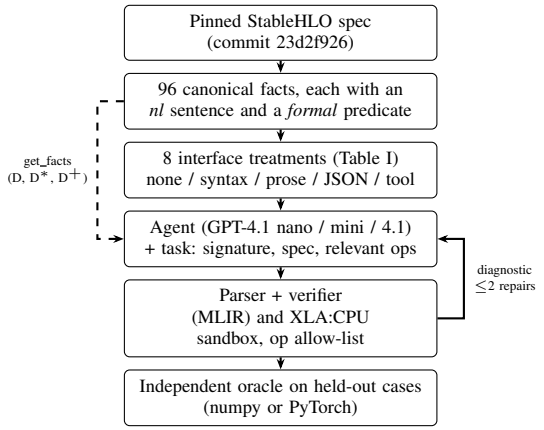
Isolating the interface is hard, for three reasons. First, documentation variants must be information-equivalent: if one rendering carries a concrete example and another only a schema, or one is longer, the comparison measures that difference rather than the representation. Our own first study fell into this trap and reached a conclusion that later reversed (§IV). Second, correctness must be judged by oracles that are independent of the IR, including for real-world code lowered from ML frameworks. Third, making documentation available is not the same as having it used: an agent with a retrieval tool may never call it.

We address these challenges with a fact-level design on StableHLO [8], a versioned, specification-backed dialect of MLIR (Multi-Level IR) [2] that is a portability layer between ML frameworks and compilers (Fig. 1). A canonical inventory anchored to the pinned specification is rendered as length-matched prose, as a formal contract with the same examples, or behind a retrieval tool whose uptake we measure and, in one condition, force. Correctness is judged by numpy or PyTorch oracles on held-out inputs. In our data, failures were dominated by *well-formedness* errors: agents rarely wrote programs that verified but computed the wrong values. Post-hoc checks show that documentation presence helped six of seven models, whereas the format and uptake effects were large only for GPT-4.1 mini and 4.1 (§V). Our claims therefore concern LLM program generation with harness-guided repair against one pinned StableHLO toolchain, not semantic reasoning, autonomous agents, or ISA design beyond its documentation.

Contributions. (i) A controlled methodology and harness that vary agent-facing ISA documentation while holding IR, toolchain, tasks and agent loop fixed, with an automated equivalence audit (§III). (ii) A 328-task StableHLO suite: synthetic tasks plus 176 KernelBench modules lowered and validated end-to-end against PyTorch oracles (§III-C). (iii) Two studies: an exploratory one that exposed confounds in our own treatments, and a confirmatory one with a frozen protocol (5,005 trials) that removes them (§IV), plus post-hoc checks with four further models and the raw specification (§V). (iv) Design implications for agent-facing virtual ISAs (§VI).

II. Related Work

Kernel and IR benchmarks fix the interface and compare agents. KernelBench [3] and TritonBench [4] evaluated LLM generation of GPU kernels and Triton operators, and TritonGym [5] added agentic tool use to separate model ability from workflow design. CIRBench [6] evaluated LLMs on analyzing, repairing and optimizing LLVM IR, and KernelCraft [7] benchmarked agents writing kernels for emerging accelerator ISAs. Schema-derived constrained decoding addressed natural-language-to-MLIR generation, including Sta-



Task (`bias_add_col`): $out[i,j] = x[i,j] + b[i]$, `@main(%x: tensor<4x2xf32>, %b: tensor<4xf32>)`.

Fact MOD.SYN.3, prose (B): "One-dimensional integer-list attributes (dimensions, permutation, ...) are written like `array<i64: 0, 1>`, never in the older `dense<[0, 1]> : tensor<2xi64>` form; ..."

Same fact, structured (C): `{"id":"MOD.SYN.3","rule":"int_list_attr := 'array<i64: d0, d1, ...>' (NOT 'dense<[...]> : tensor<Nxi64>'); ...; example: 'array<i64: 0, 1>'"}`

Agent (mini, D^+: this rule shown upfront), first attempt:
`%0 = "stablehlo.broadcast_in_dim"(%b) {broadcast_dimensions = dense<0> : tensor<1xi64>} ...`

Verifier: `'stablehlo.broadcast_in_dim' op attribute 'broadcast_dimensions' failed to satisfy constraint: i64 dense array attribute`

Repair (accepted by verifier and oracle):
`%0 = "stablehlo.broadcast_in_dim"(%b) {broadcast_dimensions = array<i64: 0>} ...`

Fig. 1. Study 2 design (left) and a running example from its logs (right). The same fact is rendered as prose or as a formal JSON rule; the agent's stale prior (legacy `dense` encoding) overrides the rule shown upfront, and the verifier's diagnostic, not the oracle, drives the repair (F3, F4).

bleHLO, from the decoder side [9], which is complementary to changing the interface.

Interface and documentation studies vary what the model sees. SWE-agent [10] showed that the design of the *agent-computer interface* changes agent success; its interface is a set of tools and actions, whereas ours is the ISA's documentation. DocPrompting [11] showed that retrieved documentation helps code generation for unseen APIs, and retrieving it selectively, mainly for rarely used APIs, reduced API hallucinations [12]. For the PDDL planning language, documentation alone did not help, but pipeline-driven retrieval with modular generation and error refinement did [13]; in our study the *agent* drives retrieval, and the outcome hinges on uptake and on which facts it requests. Code models carry stale knowledge of evolving APIs [14], the mechanism behind the legacy attribute encodings that StableHLO's verifier rejects in our logs.

Prompt-format studies showed that semantically equivalent formatting changes can swing LLM accuracy widely, with weak correlation across models [15], and that plain-text, Markdown, JSON and YAML templates change GPT-model results, with larger models more robust [16]. ConCodeEval [17] compared how LLMs handle code constraints written in JSON, YAML, XML, Python and natural language. Our F2 is consistent with format sensitivity: the penalty did not shrink across GPT-4.1 sizes, and it was small for models of other families (§V). What our design adds is that matching facts, examples and length reversed the sign of a format comparison, and that on-demand documentation failed through uptake rather than content.

III. Study Design

We ran two studies (Fig. 1, left). **Study 1** (exploratory; 114 synthetic tasks, 2,052 trials) surfaced two confounds in our own treatments and an uptake failure in on-demand documentation. **Study 2**, which Sections III–IV report, was preregistered internally after Study 1: its protocol, task manifest and code were frozen and hashed in version control before any evaluation trial. It fixes those confounds, adds real-world tasks derived from KernelBench, and turns the Study 1 post-hoc conditions into planned ones.

A. Fixed IR, toolchain and agent loop

All conditions share one toolchain and one agent loop. Programs are StableHLO modules with a single `@main`. We parse and verify them with the MLIR/StableHLO bindings in jaxlib 0.11.2 (StableHLO 1.20.0) and execute them on the XLA compiler's CPU backend (XLA:CPU) in a sandboxed subprocess with a 60 s timeout. An operation allow-list admits StableHLO and three element-wise operations from CHLO, the companion "client" dialect, and excludes `custom_call`. The task fixes the signature exactly. The agent receives a system prompt, the documentation (if any) and the task, and replies with a program. Failures are returned verbatim (the verifier diagnostic, or the first mismatching oracle element), with up to two repairs. Documentation stays in context, and in the interactive conditions the retrieval tool remains available for up to 6 calls. Each trial is a fresh conversation at temperature 0, one per task and condition.

B. Canonical facts and interface treatments

The treatments differ only in how one fact inventory reaches the agent. The inventory has 96 facts over 23 operation groups (25 syntax, 16 type, 32 shape, 23 semantics). Each fact is anchored to a clause of the pinned specification [8] and authored twice: as a natural-language sentence and as a formal predicate in specification notation (Fig. 1, right). Table I lists the eight treatments and the post-hoc conditions C′, C″ and R.

Two changes remove the Study 1 confounds. First, every syntax fact now carries the same general schema and the same concrete instance in both renderings (e.g. both `array<i64: ...>` and `array<i64: 1>`); in Study 1, prose carried only the instance and the formal rendering only the schema. Second, the structured contract is compact JSON, so its length is on average $0.98\times$ that of prose per task over all 328 tasks ($1.01\times$ in mean tokens, Table I; Study 1: $1.31\times$), measured with `o200k_base`, the GPT-4.1 tokenizer. An automated audit

TABLE I
INTERFACE TREATMENTS (STUDY 2). TOK.: MEAN DOCUMENTATION TOKENS (o200k_base TOKENIZER) OVER 328 TASKS. †POST-HOC (SECS. IV AND V).

	Condition	Facts upfront	Rendering	Tok.
–	NONE	none	–	0
A	MINIMAL	syntax	prose	951
N	SYNTAX-ONLY STR.	syntax	JSON	850
B	PROSE	all	prose	1455
C	STRUCTURED	all	JSON + IDs	1469
C′	JSON-PROSE†	all	JSON, prose	1807
D	INTERACTIVE	none (on demand)	JSON, tool	0
D*	INT. FORCED	as D, 1st call forced	JSON, tool	0
D^+	INT. CORE	module; rest on demand	JSON, tool	362
C″	JSON, RAW CODE†	all	JSON + code	1616
R	RAW SPEC†	whole pinned spec	Markdown	91.3k

checks every task: identical fact-ID sequences, each rendering present verbatim, and no IDs leaking into prose. All 29 code examples verify with the pinned toolchain. A task's fact set is the cross-cutting `module` facts (program structure, types, the integer-list attribute encoding, no implicit broadcasting) plus the facts of the operation groups the task lists as relevant. Every condition receives the identical task text, signature and list of relevant groups.

C. Tasks and independent oracles

Synthetic (114 tasks). We wrote 38 parametric templates over broadcasting, reductions, contractions, layout, slicing and padding, masking and indexing, conversion, and composites (e.g. softmax). Each template is instantiated three times with distinct dimension sizes, and one additional instance per template forms a development split. Each task is specified mathematically. Its oracle is a numpy reference evaluated on three held-out cases that include semantic traps: all-negative rows for max-reductions, exact zeros for strict comparisons, and out-of-range `dynamic_slice` starts.

KernelBench-derived (153 tasks). We converted all 200 level-1 and level-2 tasks of KernelBench [3] (MIT license). They cover single operators, normalizations, pooling, convolutions, losses, and fused operator chains. The pipeline (i) shrinks integer size literals deterministically, preserving divisibility where required; (ii) exposes all module parameters as explicit `@main` arguments, keeping the training-mode semantics under which KernelBench evaluates; (iii) uses PyTorch eager outputs on three cases, each with freshly initialized parameters and inputs, as the oracle (rtol=10^{-3}); and (iv) produces a reference StableHLO program via `torch.export` and `torchax`, a PyTorch-to-JAX bridge. The reference, never shown to agents, shows that the task is expressible and that oracle and toolchain agree. The agent sees the PyTorch module source, the argument mapping and the signature. We excluded 24 tasks with logged reasons (18 too large after shrinking; 6 dropout, non-tensor inputs, broken shape couplings or a failed reference). Of the remaining 176, a 10% hash split (23) forms the development set, leaving 153 evaluation tasks (267 in total).

Validation (gate G1). All 328 reference programs (152 synthetic, 176 KernelBench) parse, verify, execute and match their oracles. Mutating the reference programs (operator swaps, constant and attribute perturbations) showed that the oracle rejected all 132 non-equivalent executing mutants on synthetic tasks, but only 322 of 432 executing mutants on KernelBench tasks, whose original inputs are non-negative. We therefore built a stronger oracle: five new cases per task, judged by PyTorch, with signed, wide-range, zero-heavy, outlier and all-negative inputs (non-negative for one task whose operation requires it); two tasks whose reference fails these cases were set aside. It killed 9 further mutants; the remaining 95 also survive it and are likely equivalent. Re-grading all 1,069 accepted first attempts and 1,282 accepted final attempts on the remaining 265 tasks changed 1 and 3 labels, and none of the 24 pre-registered contrasts by more than 0.5 pp. We report the pre-registered grading.

D. Models, budget and statistics

We used three dated snapshots of one model family, `gpt-4.1-nano`, `gpt-4.1-mini` and `gpt-4.1` (all `2025-04-14`), called *nano*, *mini* and *4.1* below. A local Qwen2.5-7B solved 0 of 12 development trials and was not used (§V evaluates the larger Qwen3-Coder). The API budget for the pre-registered runs was capped at $35 and enforced by the runner. Trials were ordered task-blocked (tasks shuffled, conditions shuffled within each task), so an outcome-blind budget stop leaves complete, paired tasks.

The primary endpoint is first-attempt semantic correctness, and the primary contrast is C−B. Eight paired contrasts were pre-specified (Table III). For each model we report the mean paired difference in percentage points (pp) with a 95% cluster-bootstrap CI (10k resamples; clusters are templates for synthetic tasks and individual tasks for KernelBench), together with exact McNemar tests, Holm-corrected within each model. Per-suite results are secondary and Holm-corrected within each suite. Because 4.1 covers a task-blocked random subset, we checked that subset on nano and mini: it has the same suite mix (57% KernelBench), is slightly harder (prose 36% vs. 41% for mini), and keeps the sign of their C−B (-3.3, -6.6 pp). The failure taxonomy (assembly syntax, attribute encoding, literal, shape/type, unknown op, wrong values) was fixed before the runs.

IV. RESULTS

Study 2 comprises 5,005 pre-registered trials; post-hoc ablations and repeats are reported here and the other post-hoc runs in §V. Nano and mini completed all pre-registered trials (267 tasks each). The 4.1 run reached its share of the budget after 733 trials; under the frozen task-blocked rule it is analyzed on its 91 complete tasks (39 synthetic, 52 KernelBench). The evaluation runs cost $34.52 (plus $0.04 for a pilot) and the post-hoc ablation $4.76. Tables II and III give overall rates, cost and the pre-specified contrasts; per-suite results are in the artifact.

TABLE II

DOCUMENTATION DOMINATES, AND PROSE IS THE MOST ACCURATE AND CHEAPEST CONDITION. CORRECTNESS (%) AT FIRST ATTEMPT (1ST) AND AFTER $\leq$2 REPAIRS (REP.); ¢: LIST-PRICE CENTS PER SOLVED TASK; n = 267, 267, 91.

	nano			mini			4.1		
Condition	1st	rep.	¢	1st	rep.	¢	1st	rep.	¢
None	2.6	3.4	3.3	0.4	0.7	54	14.3	19.8	12
Minimal (A)	18.4	19.9	0.7	33.0	40.4	1.2	41.8	48.4	5.8
Syntax-only str. (N)	15.0	17.2	0.8	32.6	40.8	1.2	39.6	47.3	5.7
Prose (B)	22.8	24.7	0.6	41.2	48.7	1.1	56.0	60.4	4.5
Structured (C)	16.9	23.2	0.7	33.3	39.7	1.4	44.0	53.8	5.6
Interactive (D)	12.7	13.5	0.9	19.9	21.7	2.5	12.1	16.5	16
Int. forced (D*)	14.2	15.7	1.0	22.1	23.6	2.6	29.7	31.9	10
Int. core (D+)	3.7	4.5	2.8	20.6	34.8	1.7	29.7	37.4	7.5

TABLE III

SYNTAX FACTS HELP EVERY GPT-4.1 MODEL; THE CONTRACT TRAILS PROSE AND ON-DEMAND ACCESS TRAILS UPFRONT DELIVERY. PRE-SPECIFIED CONTRASTS, FIRST ATTEMPT: Δ (PP), 95% CLUSTER CI, DISC. (SOLVED ONLY BY FIRST / SECOND), HOLM WITHIN MODEL.

Model	Contrast	Δ (pp)	95% CI	disc.	p_{Holm}
nano	Structured – Prose	-6.0	$[-12.5, +0.0]$	12/28	0.083
	Interactive – Structured	-4.1	$[-9.0, +0.7]$	11/22	0.240
	Int. forced – Interactive	$+1.5$	$[-0.8, +4.1]$	9/5	0.424
	Int. core – Interactive	-9.0	$[-14.7, -3.8]$	3/27	<0.001
	Int. core – Structured	-13.1	$[-19.8, -6.6]$	5/40	<0.001
	Prose – Minimal	$+4.5$	$[+0.0, +9.1]$	18/6	0.091
	Minimal – None	$+15.7$	$[+10.1, +21.8]$	43/1	<0.001
	Syntax-only struct. – Minimal	-3.4	$[-8.6, +1.5]$	11/20	0.299
mini	Structured – Prose	-7.9	$[-12.6, -3.7]$	4/25	<0.001
	Interactive – Structured	-13.5	$[-19.3, -8.0]$	4/40	<0.001
	Int. forced – Interactive	$+2.2$	$[-1.1, +5.5]$	14/8	0.859
	Int. core – Interactive	$+0.7$	$[-5.8, +7.0]$	22/20	1.000
	Int. core – Structured	-12.7	$[-18.6, -7.2]$	4/38	<0.001
	Prose – Minimal	$+8.2$	$[+3.7, +13.3]$	26/4	<0.001
	Minimal – None	$+32.6$	$[+24.3, +40.5]$	87/0	<0.001
	Syntax-only struct. – Minimal	-0.4	$[-4.2, +3.2]$	8/9	1.000
4.1	Structured – Prose	-12.1	$[-19.1, -5.7]$	0/11	0.004
	Interactive – Structured	-31.9	$[-44.2, -19.0]$	2/31	<0.001
	Int. forced – Interactive	$+17.6$	$[+4.5, +30.6]$	21/5	0.007
	Int. core – Interactive	$+17.6$	$[+7.2, +28.7]$	17/1	<0.001
	Int. core – Structured	-14.3	$[-24.4, -5.4]$	2/15	0.007
	Prose – Minimal	$+14.3$	$[+7.5, +21.8]$	13/0	0.001
	Minimal – None	$+27.5$	$[+15.2, +39.6]$	27/2	<0.001
	Syntax-only struct. – Minimal	-2.2	$[-6.8, +2.1]$	1/3	0.625

F1: Documentation was the dominant lever. Syntax facts alone, the smallest documentation condition (A−None), added $+15.7$, $+32.6$ and $+27.5$ pp (all Holm $p < 0.001$). Without documentation, mini solved 1 of 267 tasks. Semantic and shape facts on top of syntax (B−A) added $+4.5$ (n.s.), $+8.2$ and $+14.3$ pp, growing with capability.

F2: With matched content, the tested structured contract scored below prose. The primary contrast C−B was negative for every GPT-4.1 model: -6.0 pp $[-12.5, 0.0]$ for nano (Holm $p = 0.083$, n.s.), -7.9 pp $[-12.6, -3.7]$ for mini ($p < 0.001$) and -12.1 pp $[-19.1, -5.7]$ for 4.1 ($p = 0.004$, 11 vs. 0 discordant tasks).

For 4.1 the penalty fell on KernelBench (30.8% vs. 11.5%, Holm $p = 0.016$ within the suite; synthetic 89.7% vs. 87.2%), for nano and mini on synthetic tasks (-13.2, -11.4 pp; KernelBench -0.7, -5.2 pp). The format of syntax facts alone made no difference (N−A: -3.4, -0.4, -2.2 pp, all n.s.). C differs from B in its JSON wrapper, fact IDs, grouping and formal predicates; the ablation below separates these. For 4.1 the extra structured failures were mostly shape/type verifier errors (32 vs. 13 first attempts), and for mini mostly assembly and attribute errors (62 vs. 35).

TABLE IV

MATCHING INSTANCES AND LENGTH SHRINKS NANO'S PENALTY, FLIPS 4.1'S NON-SIGNIFICANT ESTIMATE AND LEAVES MINI'S UNCHANGED. C−B ON THE SAME 114 SYNTHETIC TASKS, FIRST ATTEMPT (PP, 95% CI). STUDY 1: SCHEMA-ONLY SYNTAX, 1.31× LONGER; STUDY 2: SAME INSTANCES, 0.98×.

Model	Study 1 (unmatched)	Study 2 (matched)
nano	-39.5 $[-52.6, -27.2]$	-13.2 $[-27.2, 0.0]$
mini	-13.2 $[-25.4, -1.7]$	-11.4 $[-21.1, -3.5]$
4.1	$+2.6$ $[-7.0, +12.3]$	-2.6 $[-8.1, 0.0]$

Ablation: wrapper vs. predicates. To separate these factors, C′ keeps C's JSON wrapper, IDs and grouping but holds the prose sentence in each rule (post-hoc, 625 trials, addendum frozen beforehand). C′ scored below prose (-5.6 $[-9.9, -1.9]$, -4.1 $[-7.8, -0.4]$ and -6.6 pp $[-13.5, 0.0]$ for C′−B; significant only for nano, Holm $p = 0.008$), and C scored a further 0.4, 3.7 and 5.5 pp below C′ (n.s.). C′ is also 24% longer than prose (1,807 vs. 1,455 tokens), so C′−B mixes the wrapper and grouping with length. The ablation therefore suggests that the wrapper, grouping or added length, and possibly formal predicates, contribute to the penalty; it does not isolate a single cause, and our conclusion concerns the tested presentation, not machine-readable contracts in general.

Escaped code. A second post-hoc variant, C″, moved every quoted code span out of the JSON strings into a plain code block, so that no example is escaped (625 trials). It did not close the gap (C″−C: -6.4, -3.0, $+1.1$ pp; nano's loss, Holm $p = 0.003$, came mostly from more assembly-syntax errors, 122 vs. 102), and prose still beat it by 11–12 pp (all $p < 0.01$). Since C″ also separates code from its rule and is 10% longer, this shows only that removing escaping does not recover the gap; copied artifacts are ruled out (no first attempt contained escaped quotes or fact IDs).

Confound check. Matching instances and length changed Study 1's conclusions (Table IV). Nano's penalty shrank from -39.5 to -13.2 pp, so most of the Study 1 collapse came from the unmatched instances and length; mini's was unchanged (-13.2 vs. -11.4 pp), so a format effect remains; and 4.1 moved from $+2.6$ to -2.6 pp (both n.s.), so "structure helps the strongest model" did not survive. The comparison is between studies, not randomized.

F3: On-demand access trailed upfront delivery. D−C was -4.1 (n.s.), -13.5 and -31.9 pp. *Uptake:* 4.1 never called the retrieval tool (0 of 91 tasks), as in Study 1; nano and mini averaged 1.3 and 2.4 calls. Forcing a first-turn call (D*) helped only the model that would not ask: it lifted 4.1 by $+17.6$ pp ($p = 0.007$), which still trailed C by 14 pp. *Suppressed lookup:* giving the cross-cutting rules upfront (D+) helped 4.1

(+17.6 pp over D, $p < 0.001$), but nano then almost stopped retrieving (0.02 calls per trial, down from 1.34) and lost 9.0 pp ($p < 0.001$). For every GPT-4.1 model, D^+ remained 13–14 pp below C. Qwen3-Coder, Haiku 4.5 and GPT-5.4-mini did call the tool (§V).

Which facts are fetched. Retrieving agents that fetched the cross-cutting `module` facts (with the attribute-encoding rule) were correct 62%, 51% and 57% (4.1, forced) of the time, against 7%, 2% and 16% otherwise, but only 24–52% fetched them (observational; Study 1: 88% vs. 22%).

F4: The observed bottleneck was well-formedness. Across the GPT-4.1 models, 99.2–99.8% of failed first attempts were rejected by the parser, the verifier or the signature check. This held on KernelBench too: at most 0.5% of its failed first attempts verified and then produced wrong values. Programs that fail verification never reach semantic testing, so this describes where agents failed, not whether semantic reasoning matters. The failure mix was dominated by invalid assembly syntax (33–44%), shape/type verifier errors (14–27%), malformed literals (16–22%) and attribute encoding (13–16%). Fig. 1 (right) shows a typical case: with the encoding rule upfront (D^+), mini still wrote the legacy `dense` form and repaired it after the diagnostic. Up to two repairs added 0.3–14.2 pp, with the largest gain for mini under D^+.

Repeat runs. Two further first-attempt runs for nano and mini (post-hoc) gave, over three runs, C−B of −5.2 pp [−10.5, −0.4] and −6.4 pp [−10.7, −2.6], A−None of +16.9 and +33.1 pp, and D−C of −5.2 pp [−9.9, −0.6] and −12.7 pp. Between runs 0–12% of cells changed label, but every sign held in every run.

Cost. Prose was also the cheapest condition per solved task for every GPT-4.1 model (Table II); C cost 1.05–1.33× as much and the on-demand conditions 1.4–4.5×. Each retrieval adds a model round trip (D^*: 3.7–3.9 model calls per trial against 1.9–2.5 for B), and documentation is resent with every repair, so failures multiply its cost; None still used 3.2–3.3k input tokens per trial, against 6.0–7.3k for B. Median trial times (6–14 s, including local verification with six concurrent workers) indicate latency only roughly.

V. Beyond GPT-4.1 and Curated Facts

These runs were added after review, each under an addendum frozen and hashed before its first trial; they report first-attempt correctness on all 267 tasks.

Raw specification. Given the whole pinned specification instead of curated facts (91.7k tokens per trial, 98–99% served from the prompt cache), nano and mini scored 10.9% and 15.0%: −12.0 pp [−18.0, −6.5] and −26.2 pp [−34.1, −18.5] below prose (both $p < 0.001$), below even syntax-only facts (−7.5, −18.0 pp), and above None (+8.2, +14.6 pp). Its examples use the current attribute encoding, but 51% of mini's attempts failed on it. A first attempt cost 5.6–6.2× a prose one with caching, about 20× without.

Seven models. We added Claude Haiku 5.5 (all eight conditions, full protocol; default sampling, since it rejects temperature 0; thinking off). To separate vendor from release date, we also added the open-weights Qwen3-Coder (OpenRouter, host pinned), Claude Haiku 4.5 and GPT-5.4-mini (a reasoning model, at its lowest effort), on None, A, B, C and D at temperature 0 where supported (Table V). Syntax facts helped six of seven models (+15.0 to +34.8 pp, all $p < 0.001$). Haiku 4.5 behaved like GPT-4.1, which suggests that the one exception, Haiku 5.5, reflects its prior (89.5% without documentation) rather than its vendor: syntax facts lowered its accuracy by 7.9 pp, apparently by steering it from the pretty assembly form it wrote reliably to the generic form the facts teach (observational).

TABLE V
Documentation presence helps six of seven models; format and uptake penalties are large only for GPT-4.1 mini and 4.1. First attempt: None (%), contrasts (pp), tool calls per trial under D; *Holm $p < 0.05$; Rel.: yy-mm; rows below GPT-4.1 are post-hoc.

Model	Rel.	n	None	A−None	B−A	C−B	D−C	Calls
GPT-4.1 nano	25-04	267	2.6	+15.7*	+4.5	−6.0	−4.1	1.3
GPT-4.1 mini	25-04	267	0.4	+32.6*	+8.2*	−7.9*	−13.5*	2.4
GPT-4.1	25-04	91	14.3	+27.5*	+14.3*	−12.1*	−31.9*	0.0
Qwen3-Coder	25-07	267	15.0	+31.5*	+3.4	−1.1	+0.4	4.0
Haiku 4.5	25-10	267	27.0	+34.8*	+8.2*	−1.9	−2.2	3.6
GPT-5.4-mini	26-03	267	31.5	+15.0*	+2.6	−0.4	−3.8	3.8
Haiku 5.5	26-10	267	89.5	−7.9*	+3.0	+1.9	−1.5	0.2

The GPT-4.1 format and uptake effects were small elsewhere. C−B ranged from −1.9 to +1.9 pp (n.s.; Qwen3-Coder −3.4 pp in a second, full-protocol run); it was negative for all six models with weak or moderate priors, but within ±5 pp by equivalence tests (90% CI) for Haiku 5.5, GPT-5.4-mini and Qwen3-Coder. The large GPT-4.1 penalty may thus reflect that family's training distribution rather than JSON as such. The three added models called the tool 3.6–4.0 times per trial, and D−C was −3.8 to +0.4 pp (n.s.). For every model, 93–99.8% of failures were static.

VI. Implications for Agent-Facing Virtual ISAs

These are hypotheses for one IR; §V tests them on four more models. **Validate interface claims with fact-equivalent, length-matched treatments** (Table IV). **Curate the specification rather than supplying it whole**: for nano and mini, the raw specification scored below even syntax-only facts (§V). **Test contract formats before adopting them.** The tested JSON contract scored below prose for GPT-4.1 but not markedly for other models, and one prose and one JSON rendering do not show that specifications should be prose. An ISA can generate several renderings from one fact inventory and should measure which one its agents use best. **Do not rely on agents to ask.** 4.1 skipped retrieval, partially documented nano stopped retrieving, and retrieving agents often missed the cross-cutting rule. For self-describing virtual ISAs, introspection mechanisms therefore need uptake design, such as a required first query or diagnostics that cite the violated fact. Three of the added models asked unprompted (§V), so uptake must be measured per agent. **Make well-formedness learnable.** Most failures are static, so a normative,

version-pinned surface syntax with concrete instances, plus diagnostics that name the violated rule, targets the actual bottleneck.

VII. Threats to Validity

Models. The pre-registered study used three snapshots of one family (GPT-4.1, April 2025) at temperature 0, with one trial per cell; repeat runs (§IV) show that 0–12% of cells change label between runs. Never calling the tool appears specific to 4.1 (§V). The four added models ran post-hoc, mostly first attempt only, with model-specific settings, and the ± 5 pp equivalence margin was chosen after their results were seen. *Tasks.* Synthetic tasks are small and fully specified, and KernelBench tasks were shrunk and exclude large 3D transposed convolutions. The stronger oracle re-grade (§III-C) moved no pre-registered contrast by more than 0.5 pp. KernelBench's PyTorch sources, and StableHLO code for newer models, may be in training data; only the StableHLO lowering is new. *Treatments.* The fact inventory and audit were authored with AI assistance (see the disclosure below); beyond the automated audit, an independent human audit of a stratified 24-fact sample (audit sheet released) is pending. Prose and contract differ in notation by design; the ablations narrow but do not identify the cause of GPT-4.1's contract penalty. The retrieval tool indexes 96 short facts that would fit in context, so D, D^* and D^+ test uptake in a favorable setting, not retrieval at specification scale; the uptake failures arose even there. *Agent loop.* Repairs are harness-driven. A self-directed agent track is implemented but was not evaluated. *Procedure.* Post-hoc runs took place one to six days after the main runs, so serving-side drift cannot be excluded. Two robustness fixes during Study 2 changed no prompt, grading or analysis (dated addenda).

VIII. Conclusion

We have shown that, holding the IR, toolchain, tasks and agent fixed, the agent-facing interface of a virtual ISA changed GPT-4.1's correctness by tens of percentage points. Documentation presence dominated, prose scored at or above the matched JSON contract, and on-demand access trailed upfront delivery even when the first lookup was forced. Post-hoc checks showed that the presence effect held for six of seven models while the format and uptake effects were large only for GPT-4.1 mini and 4.1, and that curated facts beat the raw specification for nano and mini. We have also shown that unmatched documentation comparisons can reverse these conclusions. For this toolchain, the results support measuring agent-facing virtual ISAs with matched controlled experiments on the agents that will use them; uptake interventions, retrieval over the full specification, harder tasks, other IRs and a closed-loop agent are our next steps.

Artifact Availability

Code, tasks, oracles, the frozen protocol with its addenda and hashes, and the logs of every reported trial are public at https://github.com/luxinlabs/VISA2026; post-hoc transcripts are attached to its `camera-ready` release.

AI-Use Disclosure

The author used an AI coding assistant (Anthropic Claude, via Claude Code) throughout this work. It reviewed the initial scaffold; implemented the harness, evaluator, task generator, KernelBench converter and analysis scripts; drafted the fact inventory from the pinned specification; ran the experiments (Study 2 and all post-hoc runs within budgets the author approved; the Study 1 spend of $6.80 was incurred before a budget was set); and drafted and revised the manuscript text, tables and figures. All citations were checked against publisher, proceedings or arXiv records. The evaluated models are objects of study, not authoring tools. The author directed the study, made the design and budget decisions, and takes full responsibility for the content.

References

[1] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proc. Int. Symp. Code Gener. Optim. (CGO)*, 2004, pp. 75–86.

[2] C. Lattner *et al.*, "MLIR: Scaling compiler infrastructure for domain specific computation," in *Proc. IEEE/ACM Int. Symp. Code Gener. Optim. (CGO)*, 2021, pp. 2–14.

[3] A. Ouyang *et al.*, "KernelBench: Can LLMs write efficient GPU kernels?" in *Proc. 42nd Int. Conf. Mach. Learn. (ICML), PMLR 267*, 2025, pp. 47 356–47 415.

[4] J. Li *et al.*, "TritonBench: Benchmarking large language model capabilities for generating Triton operators," in *Findings Assoc. Comput. Linguistics: ACL 2025*, 2025, pp. 23 053–23 066.

[5] Y. Guan *et al.*, "TritonGym: A benchmark for agentic LLM workflows in Triton GPU code generation," in *Proc. 43rd Int. Conf. Mach. Learn. (ICML), PMLR 306*, 2026, pp. 37 457–37 473.

[6] Z. Yang *et al.*, "CIRBench: Evaluating large language models as LLVM IR optimizers," in *Proc. 43rd Int. Conf. Mach. Learn. (ICML), PMLR 306*, 2026, pp. 144 491–144 544.

[7] J. Nie *et al.*, "KernelCraft: Benchmarking for agentic close-to-metal kernel generation on emerging hardware," in *Proc. 43rd Int. Conf. Mach. Learn. (ICML), PMLR 306*, 2026, pp. 93 070–93 121.

[8] OpenXLA, "StableHLO specification," https://github.com/openxla/stablehlo/blob/23d2f926/docs/spec.md, commit 23d2f926, accessed 2026-10-13.

[9] P. K. Rath, "Cross-dialect generalization without retraining: Benchmarks and evaluation of schema-derived constrained decoding for MLIR," 2026, arXiv:2607.18254.

[10] J. Yang *et al.*, "SWE-agent: Agent-computer interfaces enable automated software engineering," in *Adv. Neural Inf. Process. Syst. 37 (NeurIPS)*, 2024, pp. 50 528–50 652.

[11] S. Zhou, U. Alon, F. F. Xu, Z. Wang, Z. Jiang, and G. Neubig, "DocPrompting: Generating code by retrieving the docs," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[12] N. Jain, R. Kwiatkowski, B. Ray, M. K. Ramanathan, and V. Kumar, "On mitigating code LLM hallucinations with API documentation," in *Proc. IEEE/ACM ICSE-SEIP*, 2025, pp. 237–248.

[13] R. Wang and L. Zhang, "Documentation retrieval improves planning language generation," in *Proc. 14th Int. Joint Conf. Nat. Lang. Process. (IJCNLP-AACL)*, 2025, pp. 147–158.

[14] A. N. Ashik, S. Wang, T.-H. Chen, M. Asaduzzaman, and Y. Tian, "When LLMs lag behind: Knowledge conflicts from evolving APIs in code generation," 2026, arXiv:2604.09515.

[15] M. Sclar, Y. Choi, Y. Tsvetkov, and A. Suhr, "Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2024.

[16] J. He, M. Rungta, D. Koleczek, A. Sekhon, F. X. Wang, and S. Hasan, "Does prompt formatting have any impact on LLM performance?" 2024, arXiv:2411.10541.

[17] M. Kammakomati, S. Pimparkhede, S. G. Tamilselvam, P. Kumar, and P. Bhattacharyya, "ConCodeEval: Evaluating large language models for code constraints in domain-specific languages," in *Proc. ACL 2025 (Industry Track)*, 2025, pp. 1466–1479.