
Evolving Vision–Language Inference for Scientific Chart Understanding

Anonymous challenge submission

Abstract

We study code-level inference optimization for scientific-chart question answering with the fixed Qwen3-VL-2B-Instruct and Qwen3-VL-2B-Thinking checkpoints. The system combines two manually designed programs with an AlphaEvolve-style, catalog-constrained program search. Candidate programs are declarative, rendered by a trusted template, evaluated in restricted subprocesses, and selected by accuracy and measured latency. Development used CharXiv descriptive questions, greedy decoding, and one physical H200 GPU. A first formal campaign exposed a censored Thinking-baseline timeout; we retained it as a negative result and pre-registered a Recovery-v2 campaign with per-query atomic journals. External GPU contention later invalidated the whole first Speed32 block. We retained its valid single-query and Dev128 records, then reran all 15 Speed32 program-repeat blocks under an authorized, uncontended GPU0-only replacement. Every final number below is generated from that sealed composition; no old and new speed records are mixed. The held-out challenge test labels remain unavailable, so we make no hidden-test claim. A subsequent portability-only repair removed server-bound resource assumptions and replayed raw evidence independently. Earlier relocated single-query checks reproduced the sealed outputs, but later executable/package changes formally superseded those receipts. They remain historical portability-smoke-only evidence and do not enter any metric or selection.

1 Introduction

Vision–language models (VLMs) can answer questions about scientific figures, but their inference programs expose consequential choices beyond model weights: message construction, image preprocessing, token budgets, and answer extraction. Those choices affect both exact-match accuracy and wall-clock cost. This challenge asks for manual and automatically evolved inference programs for two fixed Qwen3-VL checkpoints. Our objective is therefore not training; it is controlled program search under an immutable model, a fixed metric, and reproducible hardware conditions.

Our main engineering contribution is an evidence-oriented evolution loop inspired by AlphaEvolve [2]. A local Qwen model proposes one option from a pre-registered catalog, while a typed declarative specification and trusted renderer prevent arbitrary generated Python from executing. Population state, lineage, rendered source diffs, Pareto selection, checkpoints, and resume behavior are explicit artifacts. Our empirical contribution separates Phase 2 search evidence from a formal Recovery-v2 evaluation and its whole-set Speed32 replacement. This separation matters: selection-set gains are useful for choosing candidates, but are not substitutes for evaluation on the frozen 128-query development set.

Contributions. We provide (i) naive, manual, and evolved inference programs for both fixed checkpoints; (ii) a restricted, catalog-constrained two-generation search with replay-safe checkpoints; (iii) a formal protocol that measures accuracy once and speed three times with CUDA-boundary timing; and (iv) a hash-linked reproducibility and packaging path whose checks fail closed if formal evidence is absent or modified.

2 Task, Data, and Integrity Constraints

CharXiv contains realistic figures and questions that probe chart understanding [3]. We use the descriptive validation JSON in its original insertion order. Exact match is computed after stripping leading and trailing whitespace and applying case-insensitive comparison; exceptions count as errors and remain in the denominator. Development and final manifests are frozen by query key and image hash. The challenge’s hidden test data and labels are neither present nor inferred.

The model IDs and revisions are fixed for Qwen3-VL-2B-Instruct and Qwen3-VL-2B-Thinking [1]. All evaluated programs use BF16, SDPA, batch size one, and greedy generation (`do_sample=false`, `num_beams=1`, and one returned sequence). No weights are trained or altered. Locked snapshot manifests bind the model configuration, processor, tokenizer, chat template, and weight shards; model and processor resolve from the same source and revision. BF16 is the execution dtype, not a quantized or modified checkpoint. All formal GPU experiments run sequentially with `CUDA_VISIBLE_DEVICES=0`, one visible device, and logical `cuda:0`; GPU 1 is excluded from formal measurement and selection. After all formal results were frozen, an explicitly non-formal portability check used physical GPU 0 for the two Instruct programs and physical GPU 1 for the two Thinking programs. Each lane exposed one logical `cuda:0` and ran its two programs serially. Model loading is excluded from steady-state formal timing.

3 Inference Programs

3.1 Naive baselines

The naive programs preserve the challenge’s direct model/processor structure and serve as reference points for both correctness and speed. They use the checkpoint chat template, one image, one question, greedy decoding, and a conservative answer normalization step. They intentionally avoid search-derived prompt or extraction choices. The Instruct baseline is the accuracy floor used by two of the final selection rules. The Thinking baseline remains part of the six-program single smoke evaluation, but its earlier Dev128 run exceeded 7200 seconds and is explicitly reported as timeout-censored; no accuracy is imputed.

3.2 Manual programs

The manual Instruct program requests a concise, explicitly tagged final answer and uses a conservative tag-aware parser with a 48-token cap. The manual Thinking program preserves native reasoning behavior, requests a tagged final answer, stops on the closing tag when available, and uses the single pre-registered 384-token arm. That arm followed an initial four-query result in which longer reasoning frequently failed to yield a parseable final answer. The arm changed only the token cap; it did not use question IDs, labels, or sample-specific rules.

On the Phase 2 four-query smoke set, manual Instruct answered 3/4 and manual Thinking answered 0/4. These small results were operational gates, not model selection claims. They are included to document that both programs performed real inference and to make the negative Thinking outcome visible. Final comparisons use the formal table generated below.

4 Catalog-Constrained Evolution

4.1 Why declarative candidates

Early provider attempts established an important negative result. Free-form local mutation could return a duplicate or an off-focus edit even after one correction. Those completions were retained as failures and were never re-labeled as children. The production search therefore restricts each proposal to a typed `CANDIDATE` literal. Fields cover prompt template, system instruction, reasoning mode, token budget, preprocessing, and answer extraction, with checkpoint-specific ranges. Model ID, BF16, SDPA, batch size one, and greedy parameters are hard-locked. Paths, imports, calls, attributes, shell commands, network access, query identifiers, and answers are not candidate dimensions.

A trusted renderer maps a validated specification to complete, deterministic `vlm_inference(image_path, question)` source. Lineage and diffs are computed from that rendered source. Semantic canonicalization removes dimensions that would not change runtime behavior. A proposal is valid only when the pre-registered focus changes, all non-focus fields remain permitted, the rendered source passes static/import checks, and restricted single-query evaluation succeeds.

4.2 Local catalog provider

For each request, a fixed local Instruct checkpoint scores a frozen ordered option catalog. The provider uses the exact chat suffix and next-token logits for single-token labels. Before GPU use, labels are verified to be distinct one-token encodings in the exact context. Scores are preserved as float32 bytes, hexadecimal floats, and a full ordinal ranking; ties break only by catalog ordinal. A runtime-invalid top option fails that slot. Only a duplicate already present in the parent or population may advance to the next option in the same stored ranking, without another model call. Network and credential-bearing environment variables are removed, and provider processes exit before target evaluation begins.

Qualification tries a fixed focus order and stops at the first valid receipt. Generation 1 creates two children from the manual seed. Generation 2 creates up to four children from at most two evaluated parents, with at least one valid generation-1 child represented when available. Each model is limited to six children and two generations. No third generation, hidden-test query, or post-hoc search arm is allowed.

4.3 Restricted evaluation and population state

The parent process alone owns ground truth and scoring. A target worker receives an opaque request ID, question text, and image bytes—not a split name, query key, path, expected answer, or correctness bit. Candidate code is never executed directly: only the trusted renderer/runtime is invoked. The subprocess runs with a minimized environment, Evo-local working/cache/temp directories, `shell=false`, a new process group, bounded frames, finite-number checks, and a timeout that kills only its own group. Tests verify that an unrelated sentinel survives timeout handling.

The population tracks parent IDs, generation, proposal request IDs, catalog scores, specifications, source hashes, source diffs, single-query gates, Search12 metrics, and failures. Parent selection is multi-objective: valid candidates enter a Pareto comparison of correctness and latency. Generation 1 is atomically checkpointed; a fresh resume creates a pre-GPU generation 2 journal. Once marked complete, a second resume returns before any provider or GPU call. Crash-injection tests cover checkpoint, export, summary, and completed-marker boundaries.

4.4 Phase 2 search evidence

For Instruct, two generation-1 and four generation-2 proposals were valid. On the fixed Search12 set, scores ranged from 5/12 to 7/12, while the manual seed scored 6/12. Selection32 evaluated the seed and two real children under one identical manifest and warmup. All three scored 17/32; the exported real child had a mean latency 0.024920 seconds lower

than the seed on that development selection set. For Thinking, six children were valid; Search12 scores ranged from 2/12 to 3/12 against a 3/12 seed. On Selection32, the seed scored 7/32 and both finalists scored 12/32. The exported child was 0.017065 seconds slower in mean latency than the seed. These are selection facts, not statistical or held-out claims.

The exact evolved exports are immutable rendered children. We retain the manual seed in the same Selection32 protocol to prevent a child-only advantage. Provider and target network-attempt counts were zero, all target errors in the cited Phase 2 selection runs were zero, and completed resume checks left canonical artifact hashes and mtimes unchanged.

5 Formal Recovery-v2 Evaluation

5.1 Pre-registration and manifests

Recovery-v2 was frozen before inspecting its results. The accuracy manifest is the first 128 descriptive validation queries in source order. Evolution-used queries are recorded separately, and an untouched complement is reported without being used for reselection. Speed32 is deterministically hash-selected, with at most one query per figure where possible; its warmup figure is disjoint. Programs use identical query order, image bytes, dtype, generation semantics, and error handling.

The formal candidate set contains six programs for single-query smoke: two naive, two manual, and two evolved. Dev128 and Speed32 include naive Instruct plus the four optimized programs. Naive Thinking is not silently dropped: its v1 Dev128 event is retained as timeout-censored above 7200 seconds with accuracy unavailable. Recovery-v2 gives each scheduled program a 14400-second wall bound and continues to the next program after a terminal timeout.

5.2 Measurement

Each Dev128 program is evaluated exactly once. Speed32 uses three complete, steady-state repeats in a frozen rotating program order; no fastest repeat is selected. For every query, a non-scored warmup precedes measurement, CUDA is synchronized immediately before and after the timed region, and model/processor load is excluded. The journal records preprocessing, generation, and total seconds, generated-token count, termination reason, peak memory, response hash, and error state. Speed selection first takes the median of three repeats for each query, then aggregates those 32 per-query medians into median, mean, Type-7 p95, and queries per second.

Each query journal record is committed outside the measured interval with atomic no-replace semantics. Resume accepts only a contiguous verified prefix whose source, program, manifest, and row hashes match the pre-registered run. A partial query is not guessed or overwritten. Synthetic tests cover interruption after several queries, duplicate keys, re-ordered rows, numeric tampering, source tampering, and finalization after all 26 scheduled results.

5.3 Whole-set Speed32 replacement

After all 26 original Recovery-v2 runs completed, process telemetry showed that an unrelated process overlapped GPU 0 during part of the speed campaign. We therefore marked the entire original Speed32 block non-qualifying; none of its 15 speed measurements enters selection. The 11 valid single-query and Dev128 records remain immutable. An authorized replacement reran all five programs and all three frozen repeats on hopper25 physical GPU 0, preserving the exact program sources, Speed32 manifest, warmup, rotation, evaluator, and selection rules. A 120-second idle gate preceded the run, and five-second telemetry observed no foreign GPU 0 process throughout the replacement. Validation requires exactly 11 retained records plus 15 replacement records and rejects partial replacement, copied old speed records, or any old/new mix.

Table 1: Recovery-v2 formal validation with authorized whole-set Speed32 replacement. Accuracy is exact match on Dev128; latency is the median of the 32 per-query medians over three steady-state repeats.

Program	Correct/128	Errors	Median (s)	Type-7 p95 (s)
baseline_instruct	51/128	0	14.541101	19.728077
manual_instruct	78/128	0	15.443962	21.594249
manual_thinking	23/128	0	20.043467	27.825051
evolved_instruct	78/128	0	14.071758	19.757527
evolved_thinking	44/128	0	17.912906	26.498611

Table 2: Pre-registered optimized-program selections.

Role	Selected program
best_accuracy	evolved_instruct
best_speed	evolved_instruct
best_overall	evolved_instruct

5.4 Results

The following table is generated by `evaluation/recovery_finalize.py` from the sealed Recovery-v2 composition. Compilation intentionally fails if that generated evidence is absent.

Accuracy is reported as an integer correct count over 128 as well as a fraction. All speed repeats are retained. Paired comparisons, a fixed-seed bootstrap, and McNemar counts are computed from aligned per-query records; the complement analysis is descriptive and never changes the selected programs.

The result pattern also identifies which bundled techniques are associated with the final roles, without pretending that this study is a field-by-field causal ablation. Relative to naive Instruct, both optimized Instruct programs gain 27 exact matches; their principal shared intervention is a concise, tagged-answer prompt. Evolution retained that prompt while changing the manual 48-token, tag-aware path to a 64-token budget with boxed-or-final extraction. The evolved program preserves 78/128 and has the lowest qualifying median latency (14.071758 s), versus 15.443962 s for manual Instruct. For Thinking, the evolved visible-evidence prompt and terminal-punctuation postprocessing accompany a gain from 23/128 to 44/128 and a lower median (17.912906 s versus 20.043467 s), but that program remains below the Instruct accuracy leaders. Because several choices change together, these comparisons support bundle-level attribution only.

5.5 Frozen best-program rules

Only the four optimized programs are eligible for the three `best_*` exports. Best accuracy maximizes integer correct count, then minimizes errors, speed median, generated tokens, and source hash. Best speed requires zero errors and accuracy at least $\max(\text{naive-Instruct correct}, \text{best-accuracy correct} - 4)$; among eligible programs it minimizes median speed, p95, tokens, then maximizes accuracy and uses the source hash. Four correct answers equal 1/32 on the 128-query accuracy set, matching the frozen tolerance.

Best overall is author-defined because the challenge specifies no official joint score. Eligibility requires zero errors and accuracy no worse than naive Instruct. For each eligible program, $A = \text{accuracy}/\text{max accuracy}$ and $S = \min(\text{latency}/\text{latency})$; the score is their harmonic mean. Ties prefer accuracy, then latency, tokens, and source hash. Finalization recomputes all three rules, copies the selected source byte-for-byte, and seals the table, numeric trace, campaign evidence, source identity, and exports in one manifest.

All three rules select `evolved_instruct`; this is one genuine winner serving three roles, not three independently claimed variants. It wins accuracy by the frozen tie-break over the equally accurate manual Instruct program, wins speed because its median is the lowest

among accuracy-eligible programs, and wins overall because it combines normalized maximum accuracy with normalized maximum eligible speed. Consequently, the concise tagged prompt plus robust extraction most directly support the accuracy role, the measured low-latency implementation supports the speed role, and their conjunction supports the overall role.

6 Negative Results and Failure Analysis

Three negative classes shaped the final design. First, unrestricted local proposal generation produced duplicate and off-focus specifications. The system consumed those requests, saved their hashes and reason codes, and did not fabricate children. Catalog scoring made the choice space auditable and bounded. Second, launcher and isolated-bytecode failures occurred before measurement in development. They were retained, verified to have no measurement artifacts, and did not count as successful proposals. Resume then used the same frozen campaign rather than silently restarting it.

Third, formal v1 completed seven results before naive Thinking Dev128 remained healthy but exceeded the 7200-second outer limit. The runner terminated it, recorded no complete raw/summary/journal, released GPU 0, and did not auto-repeat. That observation is right-censored: it establishes only total time greater than 7200 seconds and supplies no accuracy estimate. Recovery-v2 preserves the entire v1 directory byte-for-byte, gives each new program a larger explicit bound, adds per-query journals, and continues other scheduled programs after timeout. The immutable v1 closeout and Recovery-v2 protocol had declared the wrong v1 campaign-plan digest; the verified erratum binds their unchanged bytes to the actual plan SHA-256 `effe1b6a1741885b2ab4c48c71efedc1735825039d837ea0533b79f13258c79`, with neither formal metrics nor selection affected.

Finally, a foreign GPU 0 process overlapped the original Recovery-v2 speed window. We preserved all 26 original artifacts byte-for-byte and recorded an append-only contamination incident rather than selecting from questionable latency data. The authorized whole-set replacement was accepted only after all 15 blocks completed under the frozen protocol with uncontended telemetry. Dev128 accuracy was retained; the original Speed32 block remains permanently non-qualifying.

7 Security, Validity, and Limitations

The restricted worker meaningfully reduces accidental leakage and code-execution risk, but it is not a kernel security boundary. Namespace capability is probed and reported; where unavailable, only trusted catalog-rendered candidates are eligible. The provider is offline and local, candidate inputs omit labels and identifiers, and all cache/temp/history paths remain inside the project root. Symlink audits resolve every link and reject absolute, broken, external, escape, or looping targets. Tests also exercise malformed frames, duplicate JSON keys, NaN/Infinity, credential/proxy stripping, target timeouts, unrelated-process survival, artifact races, and no-overwrite publication.

Internal validity is limited by one formal-measurement GPU, one dtype, batch size one, and a modest development budget. Phase 2 Search12 and Selection32 were used for selection and therefore are optimistic estimates. Dev128 is public validation data rather than the hidden challenge test. It is also not a strictly untouched holdout: exact queries 21_2 and 68_2 occur in the recorded evolution-used set, and eight Dev128 queries share those two figures. Exact match ignores semantic equivalence and may penalize explanatory yet correct responses. Latency on a shared cluster can vary with system load even under synchronization and repeated measurement. We report raw repeats and paired records so readers can inspect that variability.

External validity is also limited: both checkpoints are 2B Qwen3-VL variants and all questions are CharXiv descriptive queries. Improvements may not transfer to reasoning questions, other VLMs, quantized inference, or batched serving. We do not estimate hidden-test

accuracy, claim significance from Phase 2 selection sets, or reinterpret timeout censoring as a measured score.

8 Reproducibility and Artifact Traceability

The repository contains locked dependencies, fixed model revisions and snapshot manifests, data/image manifests, candidate sources, evolution lineage, tests, formal protocols, and a single `reproduce.sh`. With no mode argument, `reproduce.sh` selects `fresh` and invokes the official `evaluate.py` PROGRAM entry in isolated GPU lanes. The explicit `evidence` mode is a standard-library, CPU-only replay that reopens raw records, recomputes correctness, errors, latency aggregates, speed selection, and all three best-program rules, and refuses missing or changed evidence. Fresh inference is therefore the operational default, while evidence replay remains independently available without model loading. The final checker additionally requires the complete Recovery-v2 evidence and authorized whole-set Speed32 replacement, verifies the sealed 11-plus-15 composition, checks the official NeurIPS PDF, and audits the deterministic ZIP. Historical Phase 2 and v1 evidence have separate CPU-only verification modes and cannot satisfy final completion.

Portability closeout. Package resources and managed outputs are physically contained, while explicit model snapshots and evaluator images are validated as read-only regular files instead of being forced beneath the source tree. Model and processor always resolve from the same locked source; an invalid override fails without fallback. The superseded root `BLOCKED_STATUS.json` is retained only in a historical archive and excluded from the current package. Two historical source snapshots were extracted with `git archive` into fresh staging directories, where four Single1 runs reproduced the sealed query, question, image hash, prediction bytes, and correctness with zero errors. Subsequent executable/package changes superseded both receipts as final gates, so they are not presented as proof for the current archive. Current-package portability must instead pass clean-extraction CPU imports, evidence replay, manifest/ZIP verification, and packaged-image path checks; fresh GPU inference remains conditional on the two locked model snapshots and suitable CUDA hardware. None of these checks changes formal accuracy or speed claims.

Final fresh reproduction. After the portability source and evidence gates were frozen, one no-overwrite reproduction ran five Dev128 programs and four Single1 checks through the official entry point. The Dev128 manifest contained 128 ordered queries backed by 128 distinct regular-file paths and inodes. Those paths intentionally duplicate figure bytes when different questions reference the same figure: the manifest discloses 32 unique image-content hashes rather than claiming 128 unique figures. The run used the same locked model snapshots and unchanged weight bytes described above. Its strict receipt closed all nine start/result/end chains, and a copied 42-file evidence set passed independent receipt verification.

This fresh run is labeled final-fresh-reproduction-only. It changes no formal Recovery-v2 metric or best-program selection. In particular, the fresh Single1 check for naive Thinking does not replace or impute the censored Dev128 Thinking baseline. The CPU-only `evidence` mode validates this fresh receipt together with the retained formal evidence.

Every formal number in this paper is traceable through `report/generated/report_number_trace.json`; every row names the source artifact and hash. The final manifest binds both campaign IDs, the contamination audit and replacement authorization, source commit/tree, old and replacement plans, summaries and completed markers, all 26 composed result artifacts, official style hash, table, trace, and byte-identical best exports. The checker reopens and rehashes those files, recomputes metrics and selection, rejects fallback document classes or missing formal evidence, and audits ZIP members for traversal, symlinks, duplicates, oversize payloads, empty required files, and manifest mismatch.

References

- [1] Shuai Bai et al. Qwen3-vl technical report. arXiv preprint arXiv:2511.21631, 2025.

- [2] Google DeepMind. Alphaevolve: A gemini-powered coding agent for designing advanced algorithms. Technical report, 2025.
- [3] Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, Alexis Chevalier, Sanjeev Arora, and Danqi Chen. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. In Advances in Neural Information Processing Systems, 2024.