# ASM-RISE: LEARNING TO INVESTIGATE ASSEMBLY UNDER AN EVIDENCE BUDGET

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Assembly semantics is often distributed across control flow, call relations, and data dependencies rather than contained in the visible instruction sequence. Resolving a question therefore requires not only interpreting assembly, but also deciding which program evidence to inspect next. We introduce ASM-RISE, a framework that turns assembly understanding into budgeted program investigation. A structured environment exposes complementary static views through scoped actions with explicit evidence prices, while trajectory supervised fine-tuning teaches the interaction protocol and agentic reinforcement learning jointly improves evidence acquisition and semantic answering. We evaluate the learned policies on binary code similarity retrieval, tool-free program semantics, semantic answering under four evidence budgets, and general code generation. Across three model families, RL consistently improves retrieval and budgeted reasoning over the corresponding supervised initialization. On Qwen2.5-Coder-7B, adaptive RL yields a 52.92% relative improvement in Recall@1 over SFT and outperforms both static-plan and no-tool RL on the evidence-dependent endpoints. Policy rankings differ between evidence-dependent tasks and tool-free semantics, showing that investigation quality and direct-answer proficiency are not interchangeable. ASM-RISE demonstrates that models can learn not only what assembly means, but how to investigate it.

## 1 INTRODUCTION

Understanding assembly requires more than interpreting the visible instructions. A short sequence can implement different behaviors depending on its control-flow context and the provenance of its operands. The decisive evidence may lie in a distant basic block, an external call, or a register dependency. Which view resolves the ambiguity depends on the question, and an observation can change what the model should inspect next.

Existing assembly models strengthen representations of supplied code (Su et al., 2024; Jiang et al., 2025; Wang et al., 2025a), while studies of binary summarization and code understanding show the difficulty of recovering semantics from low-level representations (Jin et al., 2023; Shang et al., 2024). A complementary problem is how to acquire the facts needed for a particular question. Providing a longer listing does not itself direct attention to the decisive locations. Structured tools expose these facts, but effective use requires choosing the evidence type, its scope, and when to stop as the investigation unfolds.

We introduce ASM-RISE (ASM for assembly and RISE for Reasoning through Interactive Structured Evidence), which turns assembly understanding into budgeted program investigation (Figure 1). Starting from a question and a bounded assembly view, the policy requests scoped static evidence or submits an answer. Requests have explicit prices, and later choices condition on previous observations. Trajectory supervised fine-tuning (SFT) first teaches the model to issue valid evidence requests and produce evidence-grounded answers. Agentic reinforcement learning (RL) builds on this initialization to jointly optimize acquisition and answering using semantic quality, evidence use, and cost. This makes the investigation process itself a learning target, beyond interpreting a fixed input.

Across Qwen2.5-Coder, Qwen3, and DeepSeek-Coder, RL improves binary code similarity retrieval and budgeted semantic reasoning over the corresponding SFT models.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

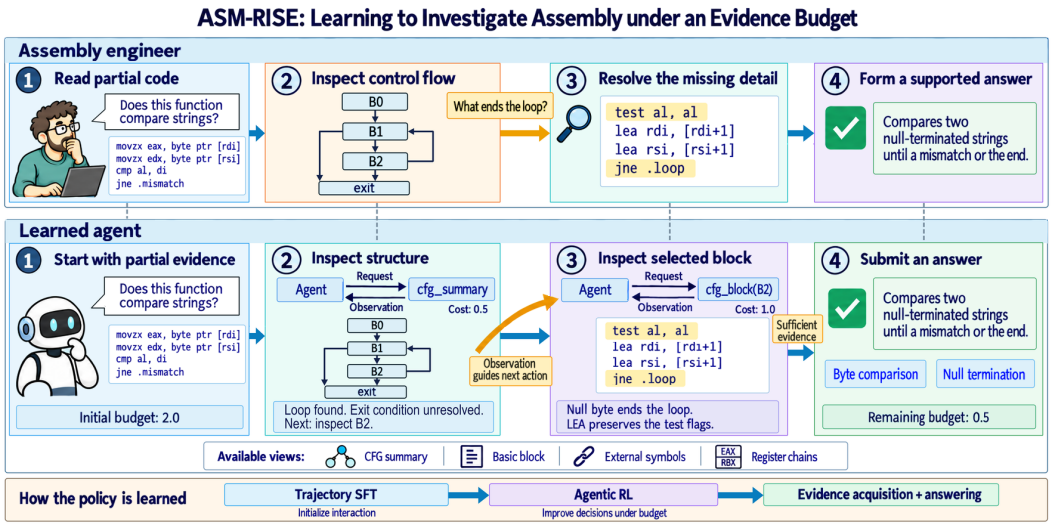


Figure 1: Learning to investigate assembly: an illustrative workflow. An engineer and a learned agent resolve a semantic information gap through targeted inspection. In the agent example, a control-flow graph (CFG) summary motivates inspecting block B2, whose instructions reveal the loop's null-termination condition. The agent submits an evidence-supported answer using 1.5 of its 2.0 budget.

In the shared-initialization Qwen2.5-Coder-7B comparison, adaptive RL outperforms static-plan and no-tool RL on both endpoints, with a 52.92% relative improvement in Recall@1 over SFT. Policy rankings differ on tool-free semantics and HumanEval, showing that investigation, direct prediction, and general coding have distinct strengths.

Our contributions are threefold:

1. We formulate assembly understanding as budgeted acquisition of question-relevant program evidence, making selection, scope, and stopping explicit decisions.
2. We introduce a structured evidence environment and an SFT-to-RL framework that jointly learns investigation and semantic answering.
3. We demonstrate gains across three model families and compare adaptive, static-plan, and no-tool policies, identifying the value of adaptive investigation and the task-dependent strengths of each policy.

## 2 Related Work

**Assembly representation and semantic modeling.** Representation learning supports binary similarity analysis using assembly context, control flow, and execution signals (Ding et al., 2019; Li et al., 2021; Wang et al., 2022; Pei et al., 2020; Xu et al., 2023). Language supervision connects assembly representations to textual descriptions (Wang et al., 2024), while Asm2Seq and CP-BCS generate binary function summaries (Taviss et al., 2024; Ye et al., 2023). BinT5 and HexT5 recover semantic information from decompiled code (Al-Kaswan et al., 2023; Xiong et al., 2023), while generative models produce higher-level program forms (Jiang et al., 2025; Tan et al., 2024). ASMA-Tune aligns a structure-aware assembly encoder with an instruction-following language model (Wang et al., 2025a). Recent systems explore obfuscation-aware self-improvement, retrieval of similar code, and function-call-aware decompilation (Wang et al., 2026; Karbab, 2026; Feng et al., 2025). These methods improve representations or transformations of supplied programs. ASM-RISE instead learns which structured evidence to acquire for a question, at what scope, and when to stop.

**Tool use and adaptive evidence acquisition.** ReAct and Toolformer connect language-model reasoning with external actions (Yao et al., 2022; Schick et al., 2023). Adaptive-RAG and Self-RAG

condition retrieval on task difficulty or evolving information needs (Jeong et al., 2024; Asai et al., 2024). SPARKLE learns a separate retrieval policy through RL (Fang et al., 2026), while AutoSearch trains agents to adapt search depth using intermediate answers (Sun et al., 2026). SWE-agent structures interactions with a software environment through a dedicated interface (Yang et al., 2024). Assembly investigation requires selecting heterogeneous program views rather than documents. ASM-RISE exposes these views as typed, scoped, and priced actions, enabling question-conditioned acquisition under an explicit budget.

**Agentic learning for tool-mediated reasoning.** AgentTuning develops agent capabilities through instruction tuning on interaction trajectories (Zeng et al., 2024). Reinforcement learning further supports search, strategic tool use, web interaction, and long-horizon policies (Jin et al., 2025; Feng et al., 2026; Qi et al., 2025; Xi et al., 2025; Wang et al., 2025b). CodeGym trains end-to-end tool use in synthetic coding environments, Context-Folding learns trajectory context management, and Agent Lightning separates execution from RL optimization (Du et al., 2026; Sun et al., 2025; He et al., 2026). ASM-RISE applies group relative policy optimization (GRPO) (Shao et al., 2024) after trajectory supervision to jointly learn assembly evidence acquisition and semantic answering.

# 3 ASM-RISE

An observation can change which program view is most useful next, making assembly investigation a sequential decision problem. ASM-RISE learns these decisions through trajectory supervision and agentic RL in a budgeted evidence environment (Figure 2).

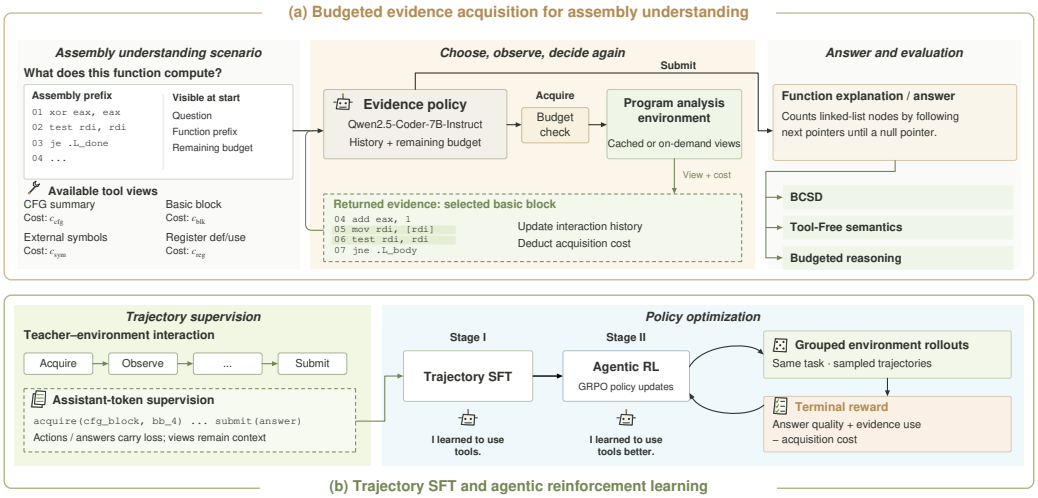


Figure 2: ASM-RISE environment and training. (a) The policy selects scoped static evidence through a budget-checked interface or submits an answer. Returned observations update its history and remaining allowance. (b) Teacher trajectories initialize the policy through assistant-token SFT. Grouped environment rollouts then support agentic RL, combining answer quality, evidence use, and acquisition cost.

## 3.1 Budgeted evidence environment

Given normalized assembly $x$ and question $q$, the initial observation $o_0$ contains $q$, a bounded assembly prefix, an evidence directory, operator schemas and prices, and budget $B$. Reference answers are withheld from the policy. An operator $k$ returns a static view $V_k(x, s)$ at scope $s$.

The complementary views in Table 1 are computed deterministically from normalized assembly, using heuristic analysis for register views (Appendix C). The program remains fixed while observations expand the policy's information. At step $t$, the policy acts from its history:

Table 1: Evidence operators and acquisition prices. Prices measure evidence access. Register chains provide heuristic definition/use summaries.

| Operator | Scope | Cost | Returned evidence |
|---|---|---|---|
| cfg_summary | Function | 0.5 | Global CFG statistics and calls |
| cfg_block | Selected block | 1.0 | Local instructions and control flow |
| external_symbols | Function or call site | 0.5 | Symbols, calls, and immediates |
| register_chains | Function | 0.7 | Heuristic register definition/use |

$$h_t = (o_0, a_0, o_1, \ldots, a_{t-1}, o_t), \qquad a_t \sim \pi_\theta(\cdot \mid h_t). \tag{1}$$

An action requests evidence, $a_t = \mathrm{acquire}(k_t, s_t)$, or submits an answer, $a_t = \mathrm{submit}(\hat{y})$. A valid request with price $c(k_t)$ within the remaining budget $b_t$ returns evidence and updates the allowance:

$$e_{t+1} = V_{k_t}(x, s_t), \qquad b_{t+1} = b_t - c(k_t), \qquad b_0 = B. \tag{2}$$

Each $o_{t+1}$ contains the response and remaining-budget feedback. Over-budget requests are rejected before execution, enforcing the following bound for executed acquisitions $\mathcal{I}(\tau)$ in trajectory $\tau$:

$$C(\tau) = \sum_{t \in \mathcal{I}(\tau)} c(k_t) \le B. \tag{3}$$

Episodes allow at most six actions and five acquisitions. Tool-enabled protocols require at least one acquisition and allow submission before the budget is exhausted. Direct mode uses no tools. Adaptive investigation thus includes deciding when to stop. Within the hard budget limit, acquisition cost also enters the training reward (Section 3.3).

### 3.2 Trajectory supervision

Trajectory SFT teaches executable evidence requests and grounded answers. Tasks draw on ASMA-Tune's assembly-question resources (Wang et al., 2025a), primarily ordinary and obfuscated Juliet programs (Boland & Black, 2012). A teacher uses reference summaries to generate adaptive demonstrations in the same environment. The filtered set contains 6,938 trajectories for Qwen2.5-Coder-7B-Instruct. Appendix E specifies the teacher, filtering criteria, and training-data composition.

We minimize next-token negative log-likelihood on assistant-generated requests and final answers. Turn-level masks exclude system instructions, user inputs, and tool observations, so observations condition later decisions without becoming prediction targets. This checkpoint initializes all three Qwen RL policies, which then train on a larger prompt pool.

### 3.3 Cost-aware policy optimization

RL connects acquisition decisions to semantic outcomes through a trajectory-level reward. Answer correctness supplies the primary signal. Grounding measures whether available program facts support important answer claims. Evidence utility measures whether acquired observations are used and provide sufficient, complementary support.

For a valid answer, the judge's quality and grounding grades map to $r_{\mathrm{qual}}(\tau)$ and $r_{\mathrm{ground}}(\tau)$, while $u_\tau$ measures evidence utility:

$$R(\tau) = \mathrm{clip}_{[-1,1]}\Big(r_{\mathrm{qual}}(\tau) + r_{\mathrm{ground}}(\tau) + 0.10\frac{u_\tau}{4} - p_{\mathrm{cost}}(C(\tau)) - P(\tau)\Big). \tag{4}$$

Here, $p_{\mathrm{cost}}$ is a progressive cost penalty, and $P$ penalizes invalid, duplicate, premature, unsupported, or contradictory behavior. Failed interactions receive deterministic negative rewards. Appendix E specifies grade mappings, cost penalties, and terminal-failure rewards.

For each task, we sample four trajectories and optimize the policy using GRPO with group-normalized terminal rewards (Shao et al., 2024). The clipped policy objective and Kullback–Leibler (KL) regularization apply to model-generated response tokens. Tool observations condition the policy but are masked from the loss. Training uses budget 6.0 and the action limits in Section 3.1. Following Agent Lightning (Luo et al., 2025), we separate execution from optimization. Appendix E provides the rollout and optimizer settings.

### 3.4 Acquisition policies and illustrative example

The core Qwen comparison branches from one SFT checkpoint. **Adaptive RL (A-RL)** selects evidence type, scope, and submission time from the evolving history. **Static-plan RL (S-RL)** follows a prescribed plan whose types and scopes are fixed before observations arrive and are not selected or revised by the model. **No-tool RL (N-RL)** learns direct answering with evidence tools disabled. A-RL and S-RL initially receive the first 40 assembly lines and an evidence directory. N-RL instead receives up to 1,200 lines from the head and tail. Appendix E describes plan construction and its training and evaluation execution. Figure 3 illustrates the three acquisition protocols on a function in Common Weakness Enumeration (CWE) category 78.

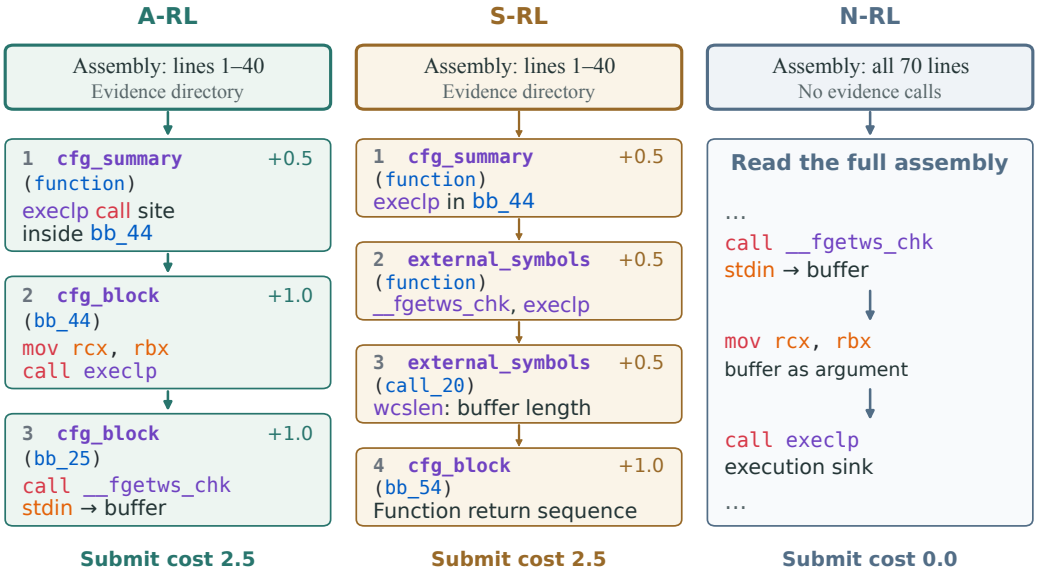


Figure 3: Three acquisition protocols on a CWE-78 function (budget 3.0). Sequences illustrate the protocols using authentic environment observations. Arrows connect acquisitions for A-RL and S-RL, and program facts for N-RL. + values denote request costs.

A-RL uses returned evidence to connect the input source to the execution sink. S-RL observes the same execution clue but continues with its prescribed scopes. Appendix F provides the complete sequences and budget accounting.

## 4 Experimental Design

### 4.1 Evaluation questions and model comparisons

We assess learning gains, adaptivity, budget sensitivity, and complementary capabilities. The core comparison uses Qwen2.5-Coder-7B-Instruct (Hui et al., 2024), with A-RL, S-RL, and N-RL initialized from the same trajectory-SFT checkpoint and evaluated under the protocols in Section 3.4. The SFT-to-RL comparison tests learning beyond imitation, while comparisons among the three Qwen policies assess the value of adaptive acquisition. SFT/RL pairs based on Qwen3-8B (Yang et al., 2025) and DeepSeek-Coder-6.7B (Guo et al., 2024) test learning gains across model families. Base

and Instruct models provide additional references. In both tool-enabled evaluations, A-RL, SFT, and initial models use adaptive acquisition (Appendix C).

### 4.2 Evidence-dependent evaluations

**Binary code similarity retrieval (BCSD).** BCSD tests whether generated explanations preserve semantics useful for identifying matching functions. Following ASMA-Tune (Wang et al., 2025a), each model generates summaries of at most 96 words for query and candidate functions. Contriever-MSMARCO (Izacard et al., 2021) encodes these summaries for similarity ranking. Generation follows Section 3.4, with budget 6 for tool-enabled models. The SFT, A-RL, S-RL, and N-RL comparison varies the summary generator while fixing candidate pools, function correspondences, and the retriever. We report Recall@1 with 500 candidates on the seven projects in Table 2. Recall@1 is the fraction of queries whose matching function ranks first. AT denotes published ASMA-Tune results.

**Budgeted Assembly Reasoning.** We evaluate semantic answering under a fixed evidence allowance. This evaluation and Tool-Free Assembly Understanding use a separately constructed Juliet C/C++ benchmark (Boland & Black, 2012), with source functions partitioned before task generation. They share a confirmation pool of 100 source functions, yielding 200 compiled-variant tasks for budgeted reasoning. Each model answers the same tasks at budgets 1, 2, 3, and 5, following Section 3.4, the operator prices in Table 1, and limits of six actions and five acquisitions. S-RL executes its complete budget-specific frozen plan before answering (Appendix E).

Qwen3.5-122B-A10B-FP8 grades semantic answer quality from 0 to 4. The normalized score is the grade divided by four, with a range of [0,1]. We report the mean normalized score at each budget and an equally weighted average across budgets 1, 2, 3, and 5.

### 4.3 Complementary capability evaluations

**Tool-Free Assembly Understanding.** This evaluation measures direct semantic prediction from the supplied assembly. With tools disabled, models answer 700 tasks over normalized x86-64 assembly from the shared confirmation pool: 200 CWE classifications, 200 direct-call target sets, 200 conditional-branch mnemonic sets, and 100 O2-to-O0 source-function matches. References derive from source labels or deterministic static analysis and are withheld from models. Classification and matching use accuracy, and call-target and branch prediction use set F1. We report each subtask score and the task-count-weighted composite:

$$S_{\mathrm{TF}} = \frac{2S_{\mathrm{CWE}} + 2S_{\mathrm{call}} + 2S_{\mathrm{branch}} + S_{\mathrm{map}}}{7}. \tag{5}$$

**General code generation.** HumanEval (Chen et al., 2021) tests whether assembly specialization preserves general coding ability through function completion checked by unit tests. We report Pass@1 (%) for each SFT/RL pair and the Qwen policies. Differences are expressed in percentage points, with full results provided in Appendix B.

Assembly scores are reported as fractions. Generation and scoring protocols are detailed in Appendix C, with aggregation, rounding, and effect-size conventions in Appendix D.

## 5 Experimental Results

### 5.1 Cross-family learning gains

**Agentic RL improves retrieval and budgeted reasoning over SFT in all three model families.** On BCSD (Table 2), Qwen2.5-Coder-7B improves from 0.2194 to 0.3355 Recall@1 (52.92% relative). The corresponding relative gains for Qwen3 and DeepSeek are 45.78% and 43.17%, respectively. Gains span all seven projects for both Qwen families and six projects with one tie for DeepSeek. The Qwen adaptive policy also exceeds the published ASMA-Tune aggregate of 0.2570 by 30.54% relative.

Table 2: BCSD results (Recall@1, 500 candidates). Bold indicates the best performance within each model family.

| Model | Binutils | Coreutils | Curl | ImageMagick | SQLite | OpenSSL | Putty | Average |
|---|---|---|---|---|---|---|---|---|
| *General-purpose reference* | | | | | | | | |
| GPT-4-Turbo | 0.1900 | 0.1750 | 0.1910 | 0.1620 | 0.1690 | 0.2320 | 0.1740 | 0.1850 |
| *Qwen2.5-Coder* | | | | | | | | |
| Base 7B | 0.0720 | 0.0760 | 0.0780 | 0.0720 | 0.0650 | 0.1010 | 0.0700 | 0.0760 |
| Instruct 7B | 0.1550 | 0.1059 | 0.1191 | 0.1347 | 0.0906 | 0.1469 | 0.1188 | 0.1244 |
| Instruct 14B | 0.2272 | 0.1792 | 0.2240 | 0.2258 | 0.1717 | 0.2464 | 0.2119 | 0.2123 |
| AT 7.2B | 0.2250 | 0.1938 | 0.2435 | 0.2448 | 0.1888 | 0.2676 | 0.2390 | 0.2570 |
| SFT 7B | 0.2260 | 0.1940 | 0.2300 | 0.2220 | 0.1940 | 0.2520 | 0.2180 | 0.2194 |
| **A-RL 7B** | **0.3472** | **0.2728** | **0.3417** | **0.3701** | **0.3411** | **0.4191** | **0.2564** | **0.3355** |
| *Qwen3* | | | | | | | | |
| Base 8B | 0.0700 | 0.0620 | 0.0740 | 0.0470 | 0.0690 | 0.0950 | 0.0640 | 0.0687 |
| SFT 8B | 0.1490 | 0.1430 | 0.1520 | 0.1130 | 0.1340 | 0.1880 | 0.1420 | 0.1459 |
| **RL 8B** | **0.2250** | **0.1650** | **0.2270** | **0.2050** | **0.1920** | **0.2970** | **0.1780** | **0.2127** |
| *DeepSeek-Coder* | | | | | | | | |
| Base 6.7B | 0.0500 | 0.0530 | 0.0520 | 0.0740 | 0.0370 | 0.0670 | 0.0470 | 0.0540 |
| Instruct 6.7B | 0.0748 | 0.0696 | 0.0833 | 0.0837 | 0.0656 | 0.0945 | 0.0747 | 0.0780 |
| V2-Lite Instruct 16B | 0.1075 | 0.0859 | 0.1088 | 0.1186 | 0.0862 | 0.1412 | 0.1070 | 0.1079 |
| AT 6.9B | 0.1143 | 0.1210 | 0.1090 | 0.1362 | 0.1010 | **0.1518** | 0.1628 | 0.1280 |
| SFT 6.7B | 0.1200 | 0.0904 | **0.1100** | 0.1016 | 0.0904 | 0.1376 | 0.1108 | 0.1087 |
| **RL 6.7B** | **0.1570** | **0.1690** | **0.1100** | **0.1710** | **0.1380** | 0.1430 | **0.2010** | **0.1556** |

Mean Budgeted Assembly Reasoning scores increase from 0.4391 to 0.6353 for Qwen2.5, from 0.3219 to 0.4784 for Qwen3, and from 0.0634 to 0.4222 for DeepSeek (Appendix A, Table 6). These improvements complement the retrieval gains, showing that RL improves semantic answering as well as the retrieval utility of generated summaries.

### 5.2 Performance across evidence budgets

**On Budgeted Assembly Reasoning, RL at budget 1 exceeds SFT at budget 5 in every family.** The comparisons are 0.6225 versus 0.4875 for Qwen2.5, 0.4638 versus 0.3375 for Qwen3, and 0.4138 versus 0.1413 for DeepSeek (Appendix A, Table 6). Increasing the supervised policy's allowance alone therefore does not close the gap to RL.

Figure 4 shows that larger allowances do not consistently improve quality. Qwen2.5 A-RL peaks at budget 2 and Qwen3 RL at budget 3. DeepSeek RL improves throughout the evaluated range.

### 5.3 Tool-free assembly understanding

**RL also improves tool-free semantic prediction across model families.** The composite increases from SFT to adaptive RL in all three families (Table 3). A-RL attains the highest source-function mapping score, 0.6800, among all evaluated models.

### 5.4 General coding performance

**General coding performance is largely retained relative to SFT.** HumanEval Pass@1 changes by -0.61 and -1.35 percentage points for Qwen2.5 and Qwen3, respectively, while DeepSeek remains at 71.95%. Appendix B provides the full results.

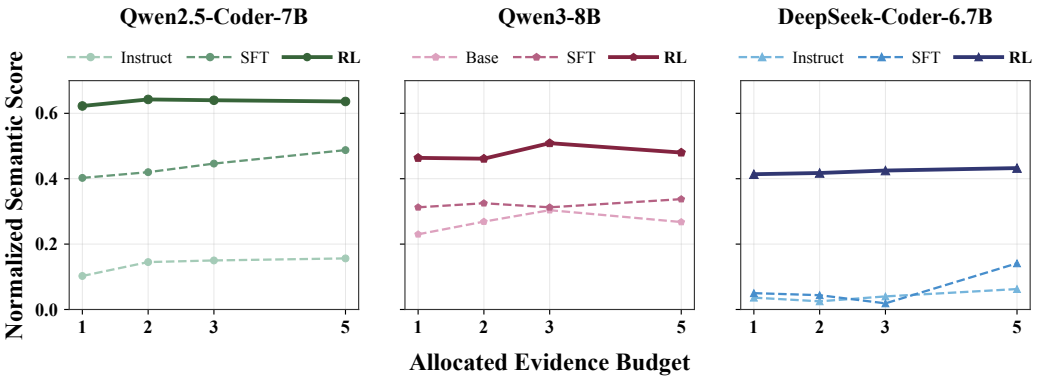


Figure 4: Budgeted Assembly Reasoning across three model families. Solid lines show RL (A-RL for Qwen2.5), and dashed lines show Base/Instruct and SFT. Scores are normalized semantic ratings. The budget axis represents evidence allowances rather than realized execution costs.

Table 3: Tool-Free Assembly Understanding. Bold indicates the best performance within each model family. Weighted reports the task-count-weighted composite, and Map Acc. reports O2-to-O0 source-function matching accuracy.

| Model | CWE Acc. | Call F1 | Branch F1 | Map Acc. | Weighted |
|---|---|---|---|---|---|
| *Qwen2.5-Coder* | | | | | |
| Instruct 7B | 0.2400 | 0.8775 | 0.8533 | 0.3700 | 0.6160 |
| SFT 7B | 0.2900 | 0.8313 | 0.8228 | 0.5700 | 0.6369 |
| **A-RL 7B** | 0.3400 | 0.8877 | 0.8131 | **0.6800** | 0.6802 |
| S-RL 7B | 0.3100 | 0.8192 | 0.8335 | 0.5900 | 0.6451 |
| N-RL 7B | **0.4400** | **0.8956** | **0.9115** | 0.6500 | **0.7349** |
| *Qwen3* | | | | | |
| Base 8B | 0.1350 | 0.8939 | 0.8415 | 0.5500 | 0.6130 |
| SFT 8B | 0.2300 | **0.9137** | 0.8850 | 0.5900 | 0.6639 |
| **RL 8B** | **0.2900** | 0.8802 | **0.9256** | **0.6200** | **0.6874** |
| *DeepSeek-Coder* | | | | | |
| Instruct 6.7B | 0.1400 | 0.7499 | 0.6466 | 0.2200 | 0.4704 |
| SFT 6.7B | **0.1800** | 0.8576 | 0.7163 | **0.2800** | 0.5411 |
| **RL 6.7B** | **0.1800** | **0.8952** | **0.7575** | 0.2500 | **0.5593** |

### 5.5 Adaptive Evidence Acquisition

**Adaptive acquisition leads the Qwen policy comparison on evidence-dependent tasks.** In Table 4, BCSD Recall@1 is 0.3355 for A-RL, 0.2541 for S-RL, and 0.1520 for N-RL. Mean budgeted reasoning follows the same ordering: A-RL reaches 0.6353, exceeding S-RL by 0.0578 and N-RL by 0.1003. A-RL leads both alternatives on every BCSD project and at every evaluated budget (Appendix A).

A-RL's advantage over S-RL, which also accesses the evidence interface, supports allowing later inspections to respond to earlier observations rather than following a fixed plan.

N-RL achieves the highest tool-free composite among the Qwen policies, 0.7349, compared with A-RL's 0.6802, and leads CWE accuracy. S-RL and N-RL also exceed A-RL on HumanEval. These rankings show that direct-answer and general coding proficiency do not fully characterize a model's ability to investigate a program.

Table 4: Qwen policy comparison across complementary endpoints. Bold indicates the best value in each column. Tool-Free and Budgeted report the composite scores defined in Section 4.

| | Assembly ↑ | | | Code ↑ |
|---|---|---|---|---|
| **Policy** | **BCSD R@1** | **Tool-Free** | **Budgeted** | **HumanEval (%)** |
| **A-RL** | **0.3355** | 0.6802 | **0.6353** | 75.61 |
| S-RL | 0.2541 | 0.6451 | 0.5775 | **81.71** |
| N-RL | 0.1520 | **0.7349** | 0.5350 | 78.66 |
| SFT | 0.2194 | 0.6369 | 0.4391 | 76.22 |

## 6 Discussion and Limitations

ASM-RISE shows that organizing program evidence is a useful learning target for assembly understanding. Improving a model's interpretation of a supplied program and learning which parts of that program to inspect address complementary problems. The gains in retrieval and budgeted answering support training the investigation process jointly with the semantic answer. This complements structural–semantic alignment approaches such as ASMA-Tune (Wang et al., 2025a) by making the selection of program views part of the learned policy: each observation can inform both the answer and the next inspection.

The advantage over a frozen plan and the strong performance at small budgets highlight the value of learning evidence allocation under explicit constraints. Acquisition prices shape the training objective, while the budget bounds the evidence available during inference. Our evaluation characterizes answer quality under these allocated allowances. Relating them to measured tool use, token expenditure, and latency would connect the learned allocation policy to the quality–resource trade-off in deployed analyzers.

The changing policy rankings also clarify what should be optimized. Direct semantic prediction, investigation, and general code generation place different demands on a model. N-RL's stronger tool-free composite alongside A-RL's evidence-dependent gains shows that direct-answer proficiency is an incomplete proxy for investigation quality. This motivates matching training and evaluation to the intended evidence interface, while tracking general coding as a complementary capability.

The present conclusions concern the evaluated model families and program collections with the four static evidence operators. Budgeted Assembly Reasoning also relies on model-based semantic grading. Evaluating additional compilation and analyzer settings, alongside expert validation of semantic ratings, would test how the learned investigation policies transfer to broader program-analysis workflows.

## 7 Conclusion

We introduced ASM-RISE, a framework for learning to acquire and interpret program evidence under an explicit budget. Its structured environment lets a policy choose which program views to inspect, at what scope, and when to answer. Trajectory SFT and agentic RL train these decisions jointly with semantic answering. Across three model families, RL improves both binary code similarity retrieval and budgeted reasoning over SFT. Within the Qwen policy comparison, adaptive acquisition outperforms fixed-plan and no-tool alternatives on both tasks. Together, these findings establish evidence acquisition as a valuable learning objective for assembly models, connecting semantic understanding with decisions about what to inspect and when to answer.

### AI USE STATEMENT

Large Language Models (LLMs) were used to polish the language of a limited number of passages in the manuscript, including minor sentence rephrasing and grammar corrections. LLMs were also used to generate teacher trajectories and provide training rewards and semantic evaluation scores, as described in Section 3 and Appendices C and E. The authors reviewed these LLM-generated outputs before using them in the study. The authors take full responsibility for the content of the manuscript, including any text polished with LLM assistance.

### ETHICS STATEMENT

This work studies assembly-code understanding using existing program-analysis resources, including the Juliet C/C++ test suite. The research is intended to support program comprehension and defensive security analysis. We recognize that improved binary-code understanding can also facilitate vulnerability discovery with potential offensive applications, and emphasize its use within authorized software analysis.

### REPRODUCIBILITY STATEMENT

The method and evaluation protocols are described in Sections 3 and 4. Appendix C specifies task construction, acquisition protocols, and scoring procedures. Appendix E reports training-data composition, optimization settings, static-plan construction, and reward mappings. Appendix F illustrates evidence acquisition and budget accounting. The code and datasets used in this study will be made publicly available in a GitHub repository.

## REFERENCES

Ali Al-Kaswan, Toufique Ahmed, Maliheh Izadi, Anand Ashok Sawant, Premkumar Devanbu, and Arie van Deursen. Extending source code pre-trained language models to summarise decompiled binaries. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 260–271. IEEE, 2023.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, pp. 9112–9141, 2024.

Tim Boland and Paul E Black. Juliet 1.1 c/c++ and java test suite. *Computer*, 45(10):88–90, 2012.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Steven HH Ding, Benjamin CM Fung, and Philippe Charland. Asm2Vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In *2019 ieee symposium on security and privacy (sp)*, pp. 472–489. IEEE, 2019.

Weihua Du, Hailei Gong, Zhan Ling, Kang Liu, Lingfeng Shen, Xuesong Yao, Yufei Xu, Dingyuan Shi, Yiming Yang, and Jiecao Chen. Generalizable end-to-end tool-use rl with synthetic codegym. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 17733–17756, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/1e4322fddd833f83c855660ac65e428d-Paper-Conference.pdf.

Jinyuan Fang, Zaiqiao Meng, and Craig Macdonald. SPARKLE: A structured and plug-and-play agentic retrieval policy for adaptive RAG models. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 38699–38719, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1793. URL https://aclanthology.org/2026.acl-long.1793/.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. In *International Conference on Learning Representations*, volume 2026, pp. 37909–37926, 2026.

Yunlong Feng, Bohan Li, Xiaoming Shi, Qingfu Zhu, and Wanxiang Che. Ref decompile: relabeling and function call enhanced decompile. *arXiv preprint arXiv:2502.12221*, 2025.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, et al. Deepseek-coder: when the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Zhiyuan He, Siwei Zhang, Zhiwen Zhou, Yuqing Yang, Yu Kang, Yuge Zhang, Luna K Qiu, Tin Yan Tsui, Jiahang Xu, and Chong Luo. Agent lightning v1. 0: Towards harnessed agentic rl. *arXiv preprint arXiv:2608.17528*, 2026.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 conference of the north american chapter of the association for computational linguistics: Human language technologies (volume 1: Long papers)*, pp. 7036–7050, 2024.

Nan Jiang, Chengxiao Wang, Kevin Liu, Xiangzhe Xu, Lin Tan, Xiangyu Zhang, and Petr Babkin. Nova: Generative language models for assembly code with hierarchical attention and contrastive learning. In *International Conference on Learning Representations*, volume 2025, pp. 95905–95926, 2025.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Xin Jin, Jonathan Larson, Weiwei Yang, and Zhiqiang Lin. Binary code summarization: Benchmarking chatgpt/gpt-4 and other large language models. *arXiv preprint arXiv:2312.09601*, 2023.

ElMouatez Billah Karbab. Asmrag: Llm-driven malware detection by retrieving functionally similar assembly code. *arXiv preprint arXiv:2604.23196*, 2026.

Xuezixiang Li, Yu Qu, and Heng Yin. Palmtree: Learning an assembly language model for instruction embedding. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pp. 3236–3251, 2021.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K Qiu, and Yuqing Yang. Agent lightning: Train any ai agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025.

Kexin Pei, Zhou Xuan, Junfeng Yang, Suman Jana, and Baishakhi Ray. Trex: Learning execution semantics from micro-traces for binary similarity. *arXiv preprint arXiv:2012.08680*, 2020.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 79791–79821, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Xiuwei Shang, Shaoyin Cheng, Guoqiang Chen, Yanming Zhang, Li Hu, Xiao Yu, Gangyang Li, Weiming Zhang, and Nenghai Yu. How far have we gone in binary code understanding using large language models. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 1–12. IEEE, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Zian Su, Xiangzhe Xu, Ziyang Huang, Zhuo Zhang, Yapeng Ye, Jianjun Huang, and Xiangyu Zhang. Codeart: Better code models by attention regularization when symbols are lacking. *Proceedings of the ACM on Software Engineering*, 1(FSE):562–585, 2024.

Jingbo Sun, Wenyue Chong, Songjun Tu, Qichao Zhang, Yaocheng Zhang, Jiajun Chai, Xiaohan Wang, Wei Lin, Guojun Yin, and Dongbin Zhao. AutoSearch: Adaptive search depth for efficient agentic RAG via reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 28059–28079, 2026.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*, 2025.

Hanzhuo Tan, Qi Luo, Jing Li, and Yuqun Zhang. Llm4decompile: Decompiling binary code with large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3473–3487, 2024.

Scarlett Taviss, Steven HH Ding, Mohammad Zulkernine, Philippe Charland, and Sudipta Acharya. Asm2Seq: Explainable assembly code functional summary generation for reverse engineering and vulnerability analysis. *Digital Threats: Research and Practice*, 5(1):1–25, 2024.

Hao Wang, Wenjie Qu, Gilad Katz, Wenyu Zhu, Zeyu Gao, Han Qiu, Jianwei Zhuge, and Chao Zhang. Jtrans: Jump-aware transformer for binary code similarity detection. In *Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis*, pp. 1–13, 2022.

Hao Wang, Zeyu Gao, Chao Zhang, Zihan Sha, Mingyang Sun, Yuchen Zhou, Wenyu Zhu, Wenju Sun, Han Qiu, and Xi Xiao. Clap: Learning transferable binary code representations with natural language supervision. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 503–515, 2024.

Xinyi Wang, Jiashui Wang, Jinbo Su, Ke Wang, Peng Chen, Yanming Liu, Long Liu, Xiang Li, Yangdong Wang, Qiyuan Chen, et al. Asma-tune: Unlocking llms' assembly code comprehension via structural-semantic instruction tuning. In *ECAI 2025: 28th European Conference on Artificial Intelligence 25-30 October 2025, Bologna, Italy-Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025) Proceedings*, pp. 3968–3975. SAGE Publications 1 Oliver's Yard, 55 City Road, London, EC1Y 1SP, 2025a.

Xinyi Wang, Rongze Chen, Ke Wang, Qiyuan Chen, Yanming Liu, Xiang Li, and Chunfu Jia. Oasif: An efficient obfuscation-aware self-improving framework for llm-based assembly code instruction following and comprehension. *arXiv preprint arXiv:2606.29155*, 2026.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025b.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, et al. Agentgym-rl: Training llm agents for long-horizon decision making through multi-turn reinforcement learning. *arXiv preprint arXiv:2509.08755*, 2025.

Jiaqi Xiong, Guoqiang Chen, Kejiang Chen, Han Gao, Shaoyin Cheng, and Weiming Zhang. HexT5: Unified pre-training for stripped binary code information inference. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 774–786. IEEE, 2023.

Xiangzhe Xu, Zhou Xuan, Shiwei Feng, Siyuan Cheng, Yapeng Ye, Qingkai Shi, Guanhong Tao, Le Yu, Zhuo Zhang, and Xiangyu Zhang. Pem: Representing binary program semantics for similarity analysis via a probabilistic execution model. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 401–412, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Tong Ye, Lingfei Wu, Tengfei Ma, Xuhong Zhang, Yangkai Du, Peiyu Liu, Shouling Ji, and Wenhai Wang. Cp-bcs: Binary code summarization guided by control flow graph and pseudo code. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14740–14752, 2023.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agent-Tuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024.

# A Detailed retrieval and budgeted reasoning results

## A.1 Project-level acquisition-policy comparison

Table 5 breaks down the shared-initialization Qwen2.5-Coder-7B comparison by project. A-RL exceeds SFT, S-RL, and N-RL on each of the seven projects, extending the aggregate policy comparison in Section 5.5. All four policies use the summary-generation protocol in Appendix C.2.

Table 5: BCSD policy comparison on Qwen2.5-Coder-7B-Instruct (Recall@1, 500 candidates). Bold indicates the best value in each column.

| Policy | Binutils | Coreutils | Curl | ImageMagick | SQLite | OpenSSL | Putty | Average |
|---|---|---|---|---|---|---|---|---|
| **A-RL** | **0.3472** | **0.2728** | **0.3417** | **0.3701** | **0.3411** | **0.4191** | **0.2564** | **0.3355** |
| S-RL | 0.2434 | 0.2002 | 0.2689 | 0.2464 | 0.2428 | 0.3742 | 0.2032 | 0.2541 |
| N-RL | 0.1463 | 0.1099 | 0.1541 | 0.1171 | 0.1577 | 0.2323 | 0.1469 | 0.1520 |
| SFT | 0.2260 | 0.1940 | 0.2300 | 0.2220 | 0.1940 | 0.2520 | 0.2180 | 0.2194 |

## A.2 Budgeted Assembly Reasoning

Table 6 reports all eleven models and policies at evidence budgets 1, 2, 3, and 5, together with their equal-weight mean. It complements the model-family curves in Figure 4 and includes S-RL and N-RL at every budget. All conditions are evaluated on the same 200 tasks. N-RL scores 0.5350 at each of the four budgets.

Table 6: Budgeted Assembly Reasoning. Bold indicates the best performance within each model family. Mean reports the average across budgets 1, 2, 3, and 5. N-RL is evaluated without tools.

| | Allocated evidence budget | | | | Equal weight |
|---|---|---|---|---|---|
| **Model** | **1** | **2** | **3** | **5** | **Mean** |
| *Qwen2.5-Coder* | | | | | |
| Instruct 7B | 0.1025 | 0.1450 | 0.1500 | 0.1563 | 0.1384 |
| SFT 7B | 0.4025 | 0.4200 | 0.4463 | 0.4875 | 0.4391 |
| **A-RL 7B** | **0.6225** | **0.6425** | **0.6400** | **0.6363** | **0.6353** |
| S-RL 7B | 0.5663 | 0.5775 | 0.5838 | 0.5825 | 0.5775 |
| N-RL 7B | 0.5350 | 0.5350 | 0.5350 | 0.5350 | 0.5350 |
| *Qwen3* | | | | | |
| Base 8B | 0.2300 | 0.2688 | 0.3038 | 0.2675 | 0.2675 |
| SFT 8B | 0.3125 | 0.3250 | 0.3125 | 0.3375 | 0.3219 |
| **RL 8B** | **0.4638** | **0.4613** | **0.5088** | **0.4800** | **0.4784** |
| *DeepSeek-Coder* | | | | | |
| Instruct 6.7B | 0.0363 | 0.0250 | 0.0400 | 0.0625 | 0.0409 |
| SFT 6.7B | 0.0500 | 0.0438 | 0.0188 | 0.1413 | 0.0634 |
| **RL 6.7B** | **0.4138** | **0.4175** | **0.4250** | **0.4325** | **0.4222** |

# B HUMANEVAL RESULTS

Table 7 reports HumanEval Pass@1 as percentages for the three model families and the Qwen policy variants. We compare corresponding SFT and RL models within each family and include the larger Instruct models and ASMA-Tune (AT) models as additional references. The shared-initialization Qwen policy comparison appears in Section 5.5.

Table 7: HumanEval Pass@1 (%). Bold indicates the best performance within each model family. AT denotes the ASMA-Tune variant.

| **Model / variant** | **Pass@1 (%)** ↑ |
|---|---|
| *Qwen2.5-Coder* | |
| Instruct 7B | 79.88 |
| Instruct 14B | **89.60** |
| AT 7.2B | 80.50 |
| Trajectory SFT 7B | 76.22 |
| A-RL 7B | 75.61 |
| S-RL 7B | 81.71 |
| N-RL 7B | 78.66 |
| *Qwen3* | |
| Base 8B | **79.88** |
| SFT 8B | 78.66 |
| RL 8B | 77.31 |
| *DeepSeek-Coder* | |
| Instruct 6.7B | 78.66 |
| V2-Lite Instruct 16B | **81.10** |
| AT 6.9B | 61.00 |
| SFT 6.7B | 71.95 |
| RL 6.7B | 71.95 |

# C TASK CONSTRUCTION AND EVALUATION PROTOCOLS

## C.1 CONTROLLED ASSEMBLY BENCHMARK

Tool-Free Assembly Understanding and Budgeted Assembly Reasoning use a separately constructed Juliet C/C++ benchmark. After excluding candidates matching historical training sources, we partition source functions into disjoint training, development, and confirmation sets before generating tasks. The two evaluations share a confirmation pool of 100 source functions, yielding 200 compiled-variant tasks for budgeted reasoning and 700 direct semantic tasks for tool-free understanding.

The tool-free tasks use normalized x86-64 assembly: 200 CWE classifications, 200 direct-call target sets, 200 conditional-branch mnemonic sets, and 100 O2-to-O0 source-function matches. References derive from source labels or deterministic static analysis. Models answer from the supplied assembly and task prompts, with tools disabled. Reference labels, source identities, and mapping answers are reserved for scoring.

## C.2 EVIDENCE ACCESS AND SUMMARY GENERATION

**Acquisition interface.** BCSD and Budgeted Assembly Reasoning use adaptive acquisition for A-RL, SFT, and initial models, a predetermined evidence plan for S-RL, and direct answering for N-RL. Tool-enabled models receive an assembly prefix and evidence directory, with at most six actions and five acquisitions. They complete at least one valid acquisition before submission. Section 3.4 defines the policy-specific initial views, and Appendix E.4 specifies the static plan.

**BCSD summaries.** Each model generates a semantic summary of at most 96 words for each query and candidate function. Tool-enabled generation uses budget 6 for each function. Query and candidate functions, their correspondences, the 500-candidate pools, and Contriever-MSMARCO are fixed across the policy comparison. The retriever encodes the summaries and ranks candidates by similarity.

**Evidence environment.** Requests cost 0.5 for a CFG summary, 1.0 for a selected block, 0.5 for external symbols, and 0.7 for a register definition/use summary. Prices are denominated in evidence units. An unaffordable request is rejected before tool execution. Both precomputed and on-demand backends return static program views, with register views obtained through heuristic definition/use analysis.

### C.3 Generation and scoring

**Tool-Free Assembly Understanding.** Generation uses temperature zero, disables the interface's explicit thinking mode, and requires fixed-format JSON. CWE classification and source-function matching use accuracy. Call-target and branch prediction use per-task set F1, averaged within each subtask. Set F1 is one when both the prediction and reference are empty. Generation failures and invalid or unparseable outputs receive zero and remain in the task denominator.

**Budgeted Assembly Reasoning.** The same 200 tasks are evaluated at each budget in $\{1, 2, 3, 5\}$, with temperature zero and explicit thinking mode disabled. Qwen3.5-122B-A10B-FP8 grades final-answer semantic quality on a 0–4 scale, using the question, reference answer, final answer, acquired evidence, and initially visible assembly. The task score is the grade divided by four. Operational errors, missing or invalid final answers, protocol failures, and invalid judge outputs receive zero and remain in the denominator.

## D Aggregation and reporting conventions

**Composite scores.** Tool-Free Assembly Understanding uses task-count weights $2/7$, $2/7$, $2/7$, and $1/7$ for CWE, call targets, branches, and source-function matching, respectively (Section 4.3). Budgeted Assembly Reasoning averages scores over the same 200 tasks at each budget, then takes the equal-weight mean of the four budget scores.

**BCSD averages.** The Qwen AT average of 0.257 is taken from ASMA-Tune. The DeepSeek RL average of 0.1556 is the arithmetic mean over the seven projects.

**Precision and comparisons.** Half-up rounding gives four decimal places for assembly scores and two for HumanEval percentages. Effect sizes use unrounded values: assembly differences are in score units, relative gains in percent of the baseline, and HumanEval differences in percentage points.

## E Training and policy implementation

### E.1 Training data and teacher trajectories

Training tasks draw on ASMA-Tune's assembly-question resources, primarily ordinary and obfuscated Juliet programs. Questions address function behavior, control and data semantics, and security properties. Qwen3.5-397B-A17B-FP8 uses the student's evidence environment and reference summaries to generate adaptive demonstrations. Filtering for termination, interaction format, acquisition count, and answer length retains 6,938 trajectories.

The 52,440 unique prompt IDs are split with seed 42 into 47,196 RL-training and 5,244 validation prompts. Of the SFT demonstration IDs drawn from this pool, 6,239 belong to the RL-training partition and 699 to validation. Final Tool-Free Assembly Understanding and Budgeted Assembly Reasoning results use the separate confirmation benchmark in Appendix C.1.

### E.2 TRAJECTORY SUPERVISION AND MODEL INITIALIZATION

The core policy base is Qwen2.5-Coder-7B-Instruct. Its trajectory-SFT checkpoint is trained on the 6,938 demonstrations for two epochs and 110 optimizer updates. Each serialized trajectory contains a message sequence and a turn-level supervision mask. The loss applies to assistant-generated evidence requests and final answers. Tokens from system instructions, user inputs, and tool observations are masked from the loss and serve as conditioning context.

A-RL, S-RL, and N-RL all start from this shared Qwen checkpoint. Qwen3-8B and DeepSeek-Coder-6.7B each have a corresponding SFT/RL pair, with adaptive acquisition in their RL stage. The initial reference models are Qwen2.5-Coder-7B-Instruct, Qwen3-8B Base, and DeepSeek-Coder-6.7B-Instruct, used in both the assembly and code evaluations.

### E.3 RL OPTIMIZATION AND INTERACTION SETTINGS

**Optimization.** The five RL runs use seed 11, with one run per policy or model-family condition. Each run uses 80 rollout iterations, 64 prompts per rollout batch, and four sampled trajectories per prompt. The global optimization batch size is 64, and each run records 320 optimizer updates. GRPO uses a clipping threshold of 0.2 and a KL-loss coefficient of 0.02. Adam uses learning rate $5 \times 10^{-6}$, cosine decay to zero, a 3% warmup fraction, weight decay 0.1, and $(\beta_1, \beta_2) = (0.9, 0.98)$.

**Rollouts and resources.** Rollout temperature is 0.8, with maximum context and response lengths of 6,144 and 1,024 tokens. Each run uses four actor GPUs and four rollout GPUs. Qwen3.5-122B-A10B-FP8 provides the training grades used by the reward in Appendix E.5.

**Training interaction.** RL uses evidence budget 6.0, with at most six actions and five successful acquisitions per episode. Adaptive and static-plan protocols expose the first 40 assembly lines and an evidence directory, require at least one acquisition, and include no worked example in the prompt. N-RL uses Direct mode with up to 1,200 assembly lines selected from the head and tail, with evidence tools disabled.

### E.4 STATIC-PLAN CONSTRUCTION AND EXECUTION

The static strategy, `ranked_v1`, constructs a deterministic plan from the current assembly and its static evidence directory before any observations arrive. Candidate requests are ordered as follows:

1. A function-level CFG summary.
2. Function-level external symbols.
3. External symbols at the first call site, if one is present.
4. The middle basic block in directory order, if present.
5. The final basic block, if distinct from the middle block.
6. Function-level register chains.

The planner traverses this list in order, selecting each candidate that fits the remaining budget until it reaches five acquisitions or the end of the list. It skips unaffordable candidates and continues to consider later entries.

During training, the plan is stored in sample metadata and shown in the system prompt. S-RL generates the prescribed requests and the final answer, while the environment checks each request against the plan. During Budgeted Assembly Reasoning evaluation, a controller executes the complete plan for the current budget before requesting the final answer. In both stages, evidence types and scopes remain fixed independently of returned observations.

### E.5 SEMANTIC GRADING AND REWARD COMPUTATION

**Judge inputs.** The training judge receives the question, reference answer, model answer, and evidence available under the policy interface: acquired observations for tool policies and visible assembly for Direct mode. Its instructions prioritize program evidence when that evidence is more complete than the reference. In addition to answer quality, it grades grounding and evidence utility on a 0–4 scale.

**Grounding.** This grade measures support for important answer claims. Grade 4 requires direct support for all such claims; 3 allows limited cautious inference; 2 mixes supported facts with unsupported guesses; 1 indicates that most key claims lack support; and 0 indicates a direct conflict with the evidence.

**Evidence utility.** This grade measures whether acquired observations contribute to the answer. Grade 4 requires multiple necessary, complementary observations supporting the conclusion; 3 resolves the core functionality; 2 supports only part of the answer; 1 largely repeats information already apparent in the prompt; and 0 indicates no acquisition, unused evidence, or irrelevant evidence. In Direct mode, quality and grounding are graded from the visible assembly. Evidence utility and acquisition cost are zero.

**Grade mappings.** Quality grades 0 through 4 map, respectively, to rewards of $-1.00$, $-0.60$, $-0.10$, $0.55$, and $1.00$. Grounding grades use adjustments of $-0.30$, $-0.20$, $-0.05$, $0.05$, and $0.10$, in the same order. Evidence utility contributes $0.10u_\tau/4$. For total acquisition cost $C$, the progressive penalty is

$$p_{\text{cost}}(C) = \min\Big\{0.15,\ 0.04\,[\max(C - 0.5, 0)]^{1.5}\Big\}. \tag{6}$$

All acquisitions consume the environment budget. The reward cost penalty applies above 0.5 evidence units and grows to a cap of 0.15. The combined reward for a valid submitted answer is clipped to $[-1, 1]$.

**Terminal failures.** Truncation, consecutive illegal actions, and other invalid-answer terminations receive $-1.0$. The terminal reward is $-0.8$ when the budget or action limit is exhausted without a submitted answer, and $-0.5$ when the judge fails.

# F ILLUSTRATIVE ACQUISITION SEQUENCES

Figure 3 presents illustrative sequences constructed from static-analysis observations for the same Juliet CWE-78 task. The x86-64 function is compiled at O2 and contains 70 instructions. The task asks the model to summarize its security-relevant behavior. A-RL and S-RL initially receive the first 40 lines and an evidence directory. N-RL instead receives all 70 lines. The allocated evidence budget is 3.0.

**Adaptive acquisition.** A function-level CFG summary costs 0.5 and reports 13 basic blocks, 17 edges, and six call sites. It locates the `execlp` call at line 50 within `bb_44`, which spans lines 44–53. Inspecting `bb_44` costs 1.0 and reveals that `rbx` is moved into `rcx` before the call, exposing the buffer's role in the arguments. Inspecting `bb_25` costs another 1.0 and shows that `r12` is derived from `rbx` plus the current wide-string offset. In this block, `stdin` is loaded at line 34 and `__fgetws_chk` writes through `r12` at line 36. The remaining budgets are 2.5, 1.5, and 0.5. Answer submission ends the four-action sequence at a total evidence cost of 2.5.

**Static-plan acquisition.** The frozen sequence requests a function-level CFG summary, function-level external symbols, external symbols at `call_20`, and block `bb_54`, followed by answer submission. The acquisition prices are 0.5, 0.5, 0.5, and 1.0, leaving budgets of 2.5, 2.0, 1.5, and 0.5. The function-level views expose the input and execution calls, `__fgetws_chk` and `execlp`. The predetermined local requests inspect `wcslen` at `call_20` and the return sequence in `bb_54`. The adaptive sequence instead follows the CFG observation to inspect `bb_44`.

**Direct answering.** N-RL receives all 70 assembly lines, including the input call at line 36 and execution call at line 50. The source-to-sink evidence is available in its initial context, so direct answering incurs zero acquisition cost.

**Program interpretation.** The function appends input from `stdin` to a wide-character buffer and passes the resulting buffer to `execlp` without sanitization. The input-producing and execution blocks establish this source-to-sink relation. The three sequences illustrate how adaptive inspection, a fixed evidence plan, and direct reading expose the program facts used to answer the same question.