CONTEXT-GROUNDED DATA CURATION AND GATE-FIRST RL REWARD DESIGN FOR SQL AUTOCOMPLETE

Anonymous authors
Paper under double-blind review

ABSTRACT

SQL autocomplete models trained on user editing logs suffer from a supervision mismatch: the recorded completion often depends on information absent at the cursor, such as a table not yet referenced, a literal not in the session, or a clause whose intent emerges only with further typing. We address three coupled problems: what constitutes a valid training target, how to measure both suggestion quality and firing-abstention behavior, and how to train the model to balance useful completions with calibrated silence. Raw editing logs contain content the model could not have inferred at the cursor; we enforce context grounding by trimming each SFT target to its longest inferable prefix and retaining empty targets as silence labels, raising precision by nearly 13 points. To further improve suggestion quality and calibrate the firing decision, we apply RL with deterministic abstention and grounding gates before a graded LLM-judge reward, which prevents the silent collapse observed with a judge-only reward and raises pass@1 on predictable triggers from 35.8% to 43.4%. Since public benchmarks do not score silence, we reconstruct intended completions from real editing sessions and design a selective evaluation framework that jointly measures suggestion quality and firing-abstention behavior. In matched seven-day deployment windows, the 4B dense model, trained on no customer data, raises live acceptance from 17.8% to 26.35% over the previous 30B-A3B MoE and reduces median latency by 71%.

1 INTRODUCTION

User editing sessions are a natural source of autocomplete supervision: they capture how each query was actually completed after the cursor. Yet this supervision is observed with hindsight. A later edit may reference a schema element the user had not yet mentioned, introduce a filter value absent from the session context, or extend the query in a direction that only becomes predictable after additional typing. Training a model on such targets, or rewarding it for reproducing them, teaches it to generate content that *happened to follow the cursor* rather than content that *could have been inferred* at the cursor. The resulting model learns to guess, and its suggestions become unreliable at the moments that matter most: when the user's intent is only half-expressed and an incorrect interruption is costly.

This problem is compounded by the demands of production autocomplete. A suggestion must arrive within a tight latency budget, respect the target SQL dialect[1] and available schema, and infer an intent that may be only partially expressed. Often, the correct output is empty because the query is already complete or the context supports no reliable continuation. Production autocomplete therefore requires two coupled decisions: *whether to suggest anything* at all and, if so, *what to suggest*. Inference-time filtering can suppress some unreliable suggestions, but it does not correct the supervision that taught the model to produce them. Common remedies target model capacity, dataset size, or inference-time context rather than the training signal. A larger fill-in-the-middle (FIM) code model (Bavarian et al., 2022; Lozhkov et al., 2024; Hui et al., 2024), more interaction data, or additional cross-file retrieval (Zhang et al., 2023; Ding et al., 2023) can improve coverage but do not change what the model is supervised to produce. Other work checks whether a generation is supported by its

[1]To preserve double-blind anonymity, the specific SQL dialect, data platform, and internal system and product names are anonymized throughout this submission.

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.

context (Jacovi et al., 2025; Tang et al., 2024) or uses model confidence to abstain (Ren et al., 2023; Kadavath et al., 2022). These checks operate at inference or evaluation time and leave the underlying mismatch intact: a valid edit observed later in a session may not have been inferable at the earlier cursor, and the model has already been trained to reproduce it.

We address this mismatch at its source with *context grounding*: every post-training signal must be supported by the information available at the cursor. This moves grounding from an inference-time or evaluation-time check into training itself and provides a common constraint across the transition from supervised to reinforcement learning. During SFT, we remove unsupported content from the targets and teach the model to stop when support ends. During RL, hard gates prevent unsupported or miscalibrated outputs from receiving a graded reward. With the backbone and training recipe fixed, replacing raw-edit targets with context-grounded references produces a 12.9-point precision gain, and the resulting 4B model exceeds the previous 30B-A3B system in shown-precision. In the reward ablation, the gates also prevent the silent collapse observed with a judge-only reward.

We test this approach in a deployed SQL autocomplete feature serving up to 70,000 users and two million triggers per day, following prior work on production code and SQL assistants (Svyatkovskiy et al., 2020; Murali et al., 2024; Maddila et al., 2025). The previous model was fine-tuned from Qwen3-Coder-30B-A3B-Instruct, mostly on synthetic examples. We replace it with a 4B dense model initialized from Qwen3-4B-Instruct-2507 (Yang et al., 2025). The deployed 4B model reaches 38.6% offline shown-precision, compared with 22.0% for the previous system, while reducing both parameter count and latency (Table 3). The comparison motivates training explicitly for dialect-specific grounding and calibrated silence.

Measuring what we optimize. Existing benchmarks do not directly measure this behavior. Code-completion and text-to-SQL benchmarks (Gong et al., 2024; Yu et al., 2018; Lei et al., 2025) typically assume a well-formed context and a known or executable target. Mid-edit SQL may not parse, has no explicit user request, and sometimes calls for an empty output. We therefore reconstruct intended completions from *real editing sessions* (Fig. 1). A two-stage, LLM-assisted reconstruction first retrieves and validates a later-executed query and extracts its raw cursor edit, then trims that edit to a context-grounded reference. Evaluation is separate: the reference is frozen once, and an offline evaluator scores every model output against it (Section 2.1).

Contributions. We present a complete pipeline for production autocomplete: we formalize the problem as selective generation with calibrated silence, construct training and evaluation datasets from real editing sessions through context-grounded reconstruction, design evaluation metrics that jointly measure suggestion quality and firing-abstention behavior, and develop a post-training recipe spanning SFT and gated-reward RL. Specifically:

- **Context-grounded supervision.** We keep only the longest context-inferable prefix of each target and use an empty target when nothing is supported. Precision rises by nearly 13 points (21.0% → 33.9%). The ablation separates the benefit of trimming unsupported content from the benefit of explicit silence labels (Sections 2.2 and 3.3).
- **Gate-first RL for suggestion and silence.** To our knowledge, this is the first application of deterministic abstention and grounding gates before an LLM-judge reward in autocomplete RL. The gates teach the policy both useful completions and calibrated abstention, preventing the silent collapse observed with a judge-only reward. Suggest pass@1 rises from 35.8% to 43.4%, while the decline rate changes from 51.2% to 49.9%. The gated reward is effective under both GRPO and SAO (Sections 2.3, 3.4 and L).
- **Selective autocomplete evaluation.** We provide a systematic pipeline to reconstruct intended completions from real editing sessions and an evaluation framework that jointly measures conditional suggestion quality and firing-abstention behavior. In production, the resulting 4B model has higher shown-precision and acceptance and lower latency than the previous 30B-A3B system (Sections 2.1 and 3.5).

2 METHOD

We formulate autocomplete as fill-in-the-middle (FIM) generation. At each cursor, the model receives the current query's SQL *prefix* and *suffix*, plus grounding context: relevant data definition language (DDL) statements for the schema, recent SQL history and executed queries, and neighboring

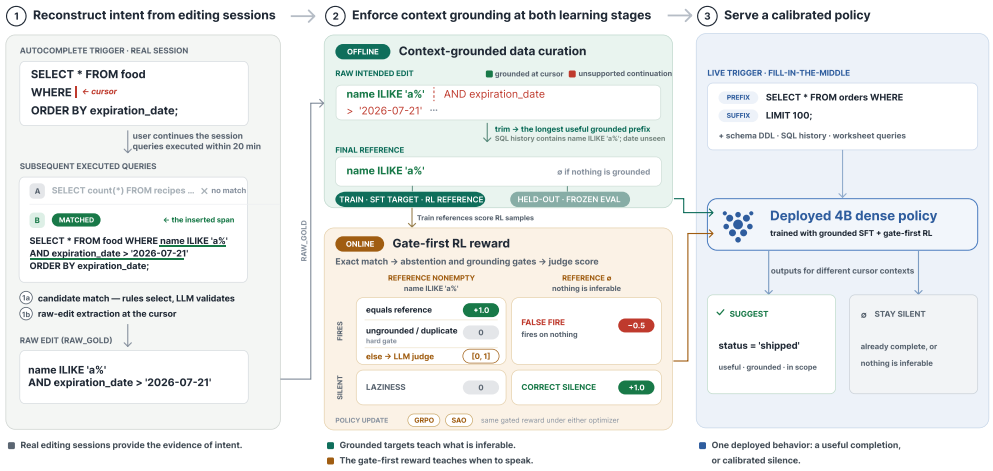


Figure 1: **Context grounding across the autocomplete pipeline.** The constraint *context grounding*, which means never learning from or rewarding content not inferable at the cursor, is enforced at both learning stages. Intended completions are reconstructed from real editing sessions **(1)**; the constraint is then applied *offline* by trimming supervised targets and *online* through a gate-first RL reward whose hard gates precede a graded LLM-judge score, under GRPO or SAO **(2)**. The deployed 4B policy is trained to produce a grounded suggestion or calibrated silence.

worksheet queries. It returns the fragment inserted between prefix and suffix. The combined SQL should be natural and either valid or a useful step toward a valid query. A completion may stop before the full intended edit, and an empty completion is often correct.

Building a production autocomplete system requires solving three problems together: (1) what counts as a valid training target, given that user editing logs contain completions the model could not have inferred at the cursor (Sections 2.1 and 2.2); (2) how to measure model performance across both suggestion quality and firing-abstention behavior (Section 2.1.2); and (3) how to train the model to balance useful suggestions with calibrated silence (Sections 2.2 and 2.3).

2.1 Evaluation from Real Editing Sessions

To measure completion quality and abstention at realistic cursors, we build frozen evaluation sets from internal autocomplete interactions, sampling from two distinct traffic periods.

2.1.1 Reconstructing the Intended Completion

Each logged trigger contains the serving context and several queries the user executed *after* the trigger. These later queries provide evidence of intent, but the evidence is noisy: the user may switch worksheets, execute an unrelated query, or edit both sides of the cursor. A direct text diff can therefore assign the wrong span or the wrong query, and would leave out a large distribution of noisy production data that cannot be diffed accurately yet still requires autocomplete.

We use **two-stage reference reconstruction** (Fig. 6 in Section B). **(1) Stage 1** first applies fixed time and text-alignment rules to select one candidate from the queries that the same user successfully executes within 20 minutes after the trigger. Claude Sonnet 4.6 then validates whether the candidate evolved from the trigger and extracts the cursor-level *raw edit*, stored as RAW_GOLD. It also assigns a coarse category: *match*, *already complete*, *blank slate*, or *no match*. **(2) Stage 2** keeps the longest prefix of a matched raw edit that is context-grounded by the prefix, suffix, schema, or session history; this becomes the *final reference* used for both evaluation and SFT. It labels the result *fully predictable* (kept intact), *partially predictable* (trimmed), *unpredictable* (emptied), *structurally unreasonable* (not a plausible inline insertion and therefore emptied; Table 4), or *already complete*. The offline evaluator uses this category to distinguish incorrect content from appropriate silence; during RL, whether the final reference is empty determines which abstention gate activates.

Reconstruction runs once, so every model is scored against the same fixed reference. Development spot checks identified and corrected systematic matching, extraction, and trimming errors (Section B).

The resulting evaluation package contains the frozen reference, its gold category, and the serving context; Section 3.1 describes the two-step protocol that scores model outputs against it.

2.1.2 MEASURING USEFUL AUTOCOMPLETE

To separate quality conditional on firing from the decision to remain silent, each completion receives $r \in \{0, 1, 2\}$ (0: incorrect; 1: safe/partial; 2: fully useful) and is fired (shown to the user) if nonempty. With k=3 completions per trigger, we report three metric groups.

Output quality. **Shown-precision** (*precision*) is the fraction of fired completions with r=2; **pass@1** is the fraction of draws with r=2; and **pass@3** is the fraction of triggers with at least one r=2 among three draws. **Mean** r averages all valid draws, giving rating-1 partial completions and safe empty outputs intermediate credit. Precision conditions on firing; the others cover the full output distribution.

Gold-aware decision. Because the preceding metrics do not credit correct silence, we also split by reference. A *suggest* draw (nonempty reference) passes only if it fires with r=2; a *decline* draw (empty reference) passes if empty or if a fired completion earns r=2. Suggest and decline pass@1 average within their respective subsets, and **blended pass@1** averages over all draws.

Behavior and length. The **decline rate** is the fraction of empty draws; we also report mean length for fired and rating-2 completions, because a model can achieve high precision by producing very short completions that rarely risk an error but provide limited utility to the user. Ablations additionally use **laziness** (empty output at a predictable, non-typo cursor) and **correct silence** (empty output at an already-complete or blank-slate cursor). Full formulas are in Section G.

2.2 CONTEXT-GROUNDED TRAINING DATA CURATION

We next apply the reconstructed, context-grounded references to SFT. We construct training examples with the same two-stage pipeline used for the evaluation sets (Fig. 6), but use Qwen3.6-27B-FP8 for all LLM-assisted steps to run it at scale. On a disjoint multi-week window of internal editing sessions, we discard rows without a reliable match and empty targets whose cursor context would make silence a misleading label. After the decontamination and anonymization described below, approximately 351k examples remain (Section K).

Use final references, not raw edits, as targets. The raw edit records what the user later inserted, but some of that content may have been impossible to predict at trigger time. Examples include an aggregation function or a filter value never mentioned in the session. Training on raw edits would encourage guessing. We instead train on the final reference from Stage 2: a partially supported edit contributes only its supported prefix, while a wholly unsupported edit contributes an empty target. SFT therefore teaches the model to predict only what the context supports. RL applies the same principle to policy samples through its grounding gates and judge (Sections 2.3 and D).

Decontamination and anonymization. To prevent evaluation overlap from being mistaken for a training gain, we decontaminate against both frozen evaluation sets. We remove a training row if it falls on the same date as any evaluation sample, if its normalized matched query equals an evaluation query, or if its parsed context has at least 60% SQL-statement Jaccard overlap with any evaluation sample. We anonymize identifiers and sensitive values consistently across prefix, target, and suffix to protect privacy and increase schema diversity.

2.3 GATE-FIRST REWARD DESIGN FOR REINFORCEMENT LEARNING

Target trimming controls the offline supervision, but RL introduces new outputs sampled from the current policy. Context-grounded SFT teaches the model *what* is inferable at the cursor; RL additionally teaches *how much* useful progress to make on each suggestion and *when* to stay silent. We enforce both through a gate-first reward: deterministic abstention and grounding gates decide eligibility, while a graded LLM-judge score drives completion quality. To verify that this design is effective independent of the optimizer, we test it under two distinct policy optimization schemes: the group-relative GRPO and the single-rollout SAO (Sections 2.3.1 and 2.3.2).

2.3.1 REWARD: HARD GATES, THEN A GRADED SCORE

The reward follows a fixed order: exact match, hard gates, then a graded score. Exact matches receive full credit immediately. Any completion that violates a gate receives zero or a fixed penalty and

never reaches the LLM judge (Section E). This ordering separates eligibility for reward from graded completion quality.

1. **Exact match.** If the completion equals the final reference, it scores 1.0 immediately; this includes the empty-reference, empty-completion case, so correct silence earns full credit.
2. **False-fire gate.** If the reference is empty (the query is complete or no completion is predictable) but the model fires, the reward is -0.5. This is how RL teaches the model to stay silent.
3. **Laziness gate.** The mirror case: a nonempty reference with an empty completion scores 0, which together with the false-fire gate prevents collapse to an always-empty policy.
4. **Context-grounding gate.** We tokenize the combined SQL and isolate table and column identifiers introduced by the completion, excluding keywords, aliases, common table expression (CTE) names, and function calls. Each identifier must have a word-boundary match in the generation context. An ungrounded identifier, or surrounding-text duplication that differs from the reference, zeroes the reward (Section D).

Only completions that pass every gate proceed to scoring. We consider two approaches: a rule-based overlap metric that provides a cheap, deterministic signal, and an LLM judge that evaluates multiple dimensions of completion quality. We compare both in Section 3.4.

SQL-BLEU: a rule-based baseline score. A natural first choice is deterministic reference overlap. We design SQL-BLEU as a structure-aware variant of BLEU (Papineni et al., 2002) for mid-edit SQL, where many queries do not parse. Its first-order term uses AST edit distance when both fragments parse, a hybrid of AST similarity and token matching on partial parses, or `SequenceMatcher` block precision as a fallback. Higher-order n-grams use NIST smoothing, and the final score includes a brevity penalty and a case-sensitivity discount (Section F). It is cheap and CPU-only, but a reference-overlap score cannot recognize every valid alternative edit.

LLM judge: a graded usefulness score. To credit valid alternatives that do not overlap the reference closely, the deployed reward uses an LLM judge to score semantic usefulness. It evaluates syntax, intent alignment, semantic correctness, completeness, and boundary coherence, then normalizes the weighted total to $[0, 1]$. Intent uses the widest range ($0/1/5$), and semantic correctness receives double weight (Section E). This reward judge is Qwen3-Coder-Next-FP8, distinct from the Qwen3.6-27B-FP8 offline evaluator. The signal can credit valid alternatives and partial progress, but running an LLM during training creates a reward-hacking surface. The hard gates therefore always run first.

2.3.2 Policy Optimization: GRPO and SAO

To test the same reward under group-relative and single-rollout optimization, we compare Group Relative Policy Optimization (GRPO) (Shao et al., 2024) with Single-Rollout Asynchronous Optimization (SAO) (Hou et al., 2026). Both optimizers sample from the current policy, score each completion with R, and update the policy. We reuse the SFT inputs and final references, but references affect the reward only through exact match, the gates, and the judge; they are never supervised targets. GRPO maximizes a PPO-style (Schulman et al., 2017) clipped surrogate with token-level importance ratio $\rho_t(\theta) = \pi_\theta(o_t \mid q, o_{<t}) / \pi_{\theta_{\text{old}}}(o_t \mid q, o_{<t})$ and clip range ϵ:

$$\mathcal{J}(\theta) = \mathbb{E}_{q,\, o \sim \pi_{\theta_{\text{old}}}} \left[\frac{1}{|o|} \sum_{t=1}^{|o|} \min\Big(\rho_t(\theta)\, \hat{A}_t,\ \text{clip}\big(\rho_t(\theta), 1-\epsilon, 1+\epsilon\big)\, \hat{A}_t \Big) \right]. \tag{1}$$

SAO retains importance weighting but masks tokens outside a double-sided trust band instead of clamping them. It computes ρ_t against rollout-engine log-probabilities, whereas GRPO recomputes $\pi_{\theta_{\text{old}}}$ with the training engine. The two optimizers also use different advantage estimators:

$$\text{(GRPO, } G \text{ rollouts)} \quad \hat{A}_i^{\text{GRPO}} = \frac{R(c_i) - \text{mean}\left(\{R(c_j)\}_{j=1}^{G}\right)}{\text{std}\left(\{R(c_j)\}_{j=1}^{G}\right)}, \tag{2}$$

$$\text{(SAO, single rollout)} \quad \hat{A}_t^{\text{SAO}} = \sum_{l \geq 0} (\gamma\lambda)^l\, \delta_{t+l}, \quad \delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t).$$

GRPO assigns one group-normalized advantage per completion and needs no value model. SAO draws a single rollout (G=1) and estimates token-level advantages with a learned GAE critic (Schulman

	Model / policy	Output quality				Gold-aware decision			Behavior / length		
		Prec.	p@1	p@3	Mean r	S@1	D@1	Bl@1	Decl.	Fired	R2
Public	Seed-Coder-8B-Instruct (ByteDance Seed, 2025)	2.2	2.1	3.7	0.100	4.7	2.2	3.3	1.9	479	131
	Qwen3-Coder-Next-80B-A3B (Cao et al., 2026)	5.8	4.5	6.5	0.364	10.1	33.9	24.1	22.8	488	16.4
	Qwen3.6-35B-A3B (Qwen Team, 2026)	11.8	6.6	9.2	0.525	14.5	55.8	38.4	43.9	118	17.2
Prod.	Prev. 30B-A3B MoE (Qwen3-Coder-30B-A3B-Instruct)	22.0	11.5	14.5	0.679	25.2	68.4	50.2	47.7	45.1	13.6
Ours (4B)	Qwen3-4B-Instruct-2507	1.5	0.2	0.3	0.608	0.5	**86.9**	50.4	86.0	271	46.8
	+ context-grounded SFT	33.9	16.6	21.4	0.843	35.8	80.1	61.3	51.2	27.1	12.0
	+ gated-reward RL (GRPO)	38.6	19.0	21.2	0.884	41.0	79.5	63.2	50.8	21.2	11.2
	+ gated-reward RL (SAO)	**40.6**	**20.3**	**22.8**	**0.915**	**43.4**	80.2	**64.7**	49.9	19.8	10.8

Table 1: **The trained 4B models achieve higher shown-precision than the public baselines and previous 30B-A3B system.** All values are percentages except **Mean** r (the mean $0/1/2$ rating over all valid draws) and **Fired/R2** (mean characters among fired/rating-2 completions). **p@1/p@3** = a rating-2 completion in one draw / any of three draws; **S@1/D@1/Bl@1** = suggest/decline/blended pass@1; **Decl.** = decline rate. Bold marks the maximum in each quality or pass column; behavior/length columns have no single preferred direction.

et al., 2016); actor updates pause when the critic explains too little return variance (Section L). Relative to the eight-rollout GRPO setting, SAO reduces judge calls per training input by roughly $8\times$. Section 3.2 compares both optimizers under the same reward.

3 Experiments and Results

3.1 Setup and Evaluation

Our models start from Qwen3-4B-Instruct-2507 (Yang et al., 2025), are fine-tuned with SFT on anonymized context-grounded sessions, and then trained with the gated RL reward described in Section 2.3.1. For evaluation, we sample 1,000 triggers from each of two distinct frozen traffic periods. We exclude `no_match` rows, which lack reliable references, and draw $k{=}3$ completions at each remaining trigger.

Each completion is scored by Qwen3.6-27B-FP8 against the frozen reference package from Section 2.1. The evaluator assigns a rating $r \in \{0, 1, 2\}$, denoting incorrect, safe or partially useful, and fully useful outputs, and identifies the primary failure category. This evaluator is distinct from the Claude Sonnet 4.6 model used to construct the evaluation references; Section H validates their inter-annotator agreement. Training details are in Sections K and L.

Table 1 reports the three metric groups defined in Section 2.1.2. All metrics pool individual draws except pass@3, which groups three draws per trigger. Mean r averages the $0/1/2$ ratings over valid draws; pass@1 and pass@3 credit only rating-2 content, while decline and blended pass@1 also credit correct silence. Full formulas and confidence-interval details are in Sections C and G.

3.2 Main Comparison

Baselines. To disentangle model scale from task-specific training, Table 1 compares public code and general-purpose models at 8B–80B scale (ByteDance Seed, 2025; Cao et al., 2026; Qwen Team, 2026), the previous 30B-A3B production MoE, and a controlled 4B ladder from the base model through context-grounded SFT to gated-reward RL with GRPO or SAO. Every model receives the same generation interface and abstention policy, and every output is scored against the same frozen references by Qwen3.6-27B-FP8 (Section C).

Results. In shown-precision, every trained 4B point estimate exceeds all baselines: context-grounded SFT reaches 33.9%, GRPO 38.6%, and SAO 40.6%, compared with 22.0% for the previous 30B-A3B system (Fig. 2). Paired bootstrap intervals exclude zero for the GRPO differences from SFT (+4.7 points; 95% CI $[+2.6, +6.7]$) and from the previous system (+16.5 points; 95% CI $[+13.5, +19.6]$; Section G). The prompt-only 4B model is mostly silent: without task-specific fine-tuning, it returns empty on 86.0% of outputs and reaches 1.5% precision. Seed-Coder-8B takes the opposite extreme, firing on 98.1% of outputs; the 80B coder fires on 77.2%. Mean fired lengths are 479 and 488 characters. Among public baselines, precision peaks at 11.8% for the 35B model; the 80B coder reaches 5.8%. RL improves completion quality without increasing abstention. From SFT to SAO, suggest pass@1 rises from 35.8% to 43.4%, while decline pass@1 remains near 80% and the decline rate moves from 51.2% to 49.9%. Blended pass@1 rises from 61.3% to 64.7%.

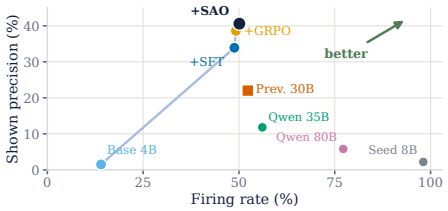


Figure 2: **Precision vs. firing rate across all models.** The 4B ladder (Base→SFT→GRPO→SAO) moves toward the upper-right corner. Public baselines and the previous 30B system occupy lower-precision or lower-coverage regions.

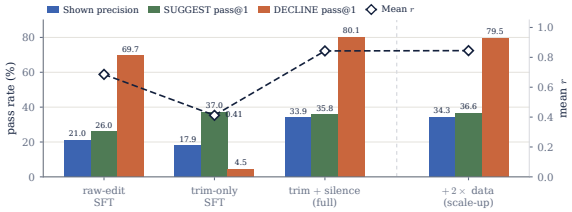


Figure 3: **Grounded content and silence labels provide complementary gains.** Trimming targets improves suggest pass@1; retaining rows trimmed to empty restores silence supervision and yields the full precision gain. Diamonds show mean r on the right axis.

Graded score	Prec.	Mean r	S@1
SQL-BLEU	30.9	0.813	34.7
LLM judge	33.6	0.845	37.2

(a)

Reward (LLM judge)	Prec.	F-fire	Lazy	C-sil.
Full gate pipeline	33.6	25.0	9.7	86.5
– grounding gate	33.0	25.6	10.2	86.4
– false-fire gate	32.2	31.0	7.7	81.1
– laziness gate	50.2	1.1	89.2	99.2

(b)

Table 2: **Reward ablations.** All rows use the same SFT initialization and two-epoch, 100k-subset SAO; compare only RL settings. (a) Graded-score comparison with gates fixed. (b) Leave-one-gate-out results. Values are percentages except mean r; **S@1/F-fire/Lazy/C-sil.** denote suggest pass@1 / false-fire / laziness / correct silence.

Mean fired length falls from 27.1 to 19.8 characters, while rating-2 length changes from 12.0 to 10.8. GRPO uses eight rollouts per prompt and SAO uses one; both improve suggest pass@1 under the same data and reward (Section L). On *partially predictable* requests, for which only a prefix of the user's eventual edit is inferable from the context available at the cursor, pass@1 rises from 26.0% for SFT to 35.4% for GRPO (+9.4 points; 95% CI $[+5.3, +13.5]$) and 37.1% for SAO (+11.1; $[+7.1, +15.3]$; Table 10). This result shows that RL improves the model's ability to produce useful, context-supported partial completions when the full edit cannot be inferred. We next isolate the contributions of grounded SFT targets and the gated RL reward.

3.3 Ablation I: Context-Grounded Data (SFT stage)

To isolate the contribution of SFT target construction, we hold the backbone and training recipe fixed. Replacing raw-edit targets with the full trim+silence grounded references produces a 12.9-point gain: precision rises from 21.0% to 33.9%, and mean r rises from 0.686 to 0.843 (Fig. 3).

Which part of grounding produces the gain? The trim-only variant keeps nonempty trimmed targets but drops rows trimmed to empty. On suggest requests, pass@1 rises from 26.0% to 37.0%, but decline pass@1 falls to 4.5% and precision drops to 17.9%, below the 21.0% raw-edit baseline.

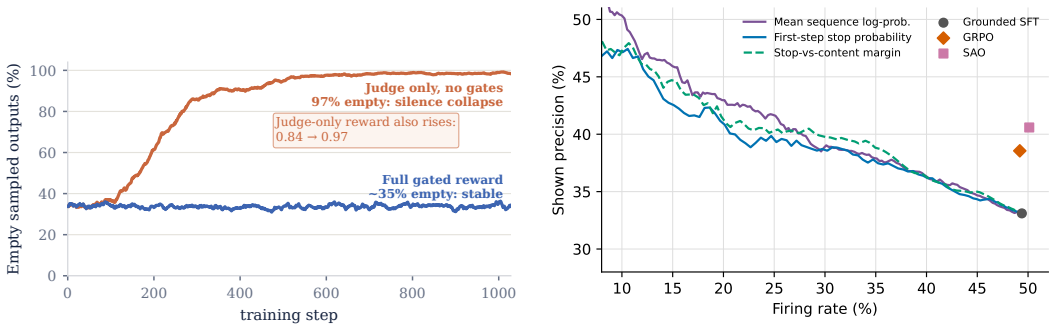
The full variant retains these rows as empty labels; decline pass@1 recovers to 80.1% and precision reaches 33.9% (Fig. 3). Thus, nonempty trimmed targets improve suggestion-side performance, while empty targets restore silence supervision; combining both yields the best precision.

Could dataset size explain the gain? To distinguish target construction from simply seeing more examples, we double the grounded dataset. Precision changes from 33.9% to 34.3% (Fig. 3), suggesting that performance saturates with respect to data size once targets are properly grounded.

3.4 Ablation II: Calibrated Reward (RL stage)

We next use the same context-grounded SFT checkpoint to ask which reward components improve quality while preserving calibrated firing. For the reward-component ablations in Table 2a, all runs start from this checkpoint and use two-epoch SAO on a 100k subset. This reduced setting compares RL configurations against each other rather than matching the full-data results in Table 1.

Does the graded score require an LLM judge? With gates fixed, the LLM judge beats SQL-BLEU in precision (33.6% vs. 30.9%), mean r (0.845 vs. 0.813), and suggest pass@1 (37.2% vs. 34.7%; Table 2a and Section F).



(a) (b)

Figure 4: **Training behavior and confidence baselines.** (a) With the judge alone, the sampled-output empty rate rises from 31% to 97% as reward rises from 0.84 to 0.97; with the gated reward it remains near 35%. (b) Threshold sweeps for mean sequence log-probability and two first-step stop scores. Near 49% firing, the SFT confidence variants remain around 33% shown-precision, below GRPO (38.6%) and SAO (40.6%).

Are both abstention gates necessary? Without the laziness gate, laziness rises from 9.7% to 89.2%, and precision appears to improve to 50.2% because the model rarely fires. Without the false-fire gate, false fires rise from 25.0% to 31.0% and correct silence falls from 86.5% to 81.1% (Table 2b).

Why does the judge-only reward fail? During training, the judge-only policy's sampled-output empty rate rises from 31% to 97% while reward rises from 0.84 to 0.97. At evaluation, the policy returns a nonempty completion for only 6 of 5,904 generations. With the full gate pipeline, the sampled-output empty rate stays near 35% (Fig. 4a).

Does the grounding gate bind? Removing it changes precision from 33.6% to 33.0% and false-fire from 25.0% to 25.6% (Table 2b), so we examine whether it is active during training. In the final full-data SAO run, the fraction of sampled completions rejected for context-unsupported identifiers falls from 2.42% in the first 200 steps to 0.62% near the end, totaling roughly 10k rollouts. The 100k full-gate SAO ablation shows the same decline (2.59% → 1.14%). In the no-gate judge-only ablation, the same diagnostic ungrounded-output rate starts at 2.6% and later falls because the policy collapses to empty outputs. To test whether these rejections identify worse completions, we apply the check post hoc to two SAO checkpoints. For the 100k-subset ablation trained without the grounding gate (the "− grounding gate" row in Table 2b), 24.5% of outputs that fail the check receive rating 2, compared with 34.1% of those that pass. The final full-data SAO checkpoint shows the same pattern (28.9% vs. 40.9%). Thus, within both checkpoints, outputs that fail the check have a lower rating-2 rate (Section D).

Can confidence filtering replace trained abstention? We compare mean sequence log-probability with the first-step stop probability and stop-versus-best-content margin (Ren et al., 2023; Kadavath et al., 2022). We use half of the requests to choose each threshold so that SFT's firing rate is as close as possible to GRPO's, without using correctness labels. Near 49% firing, all three SFT baselines reach 33.1–33.2% precision, versus 38.6% for GRPO and 40.6% for SAO (Fig. 4b and Section J). Thus, filtering does not recover the RL gain.

3.5 Real-World Deployment

Offline gains do not guarantee better product behavior, so we also compare the deployed systems (Table 3). The deployed 4B GRPO model achieves 38.6% shown-precision versus 22.0% for the previous 30B-A3B MoE. Across matched seven-day windows, live acceptance (the fraction of shown suggestions inserted) is 26.35% versus 17.8%, and median latency is 29% of the previous system's. Offline ratings and live acceptance are related but not identical: only 51.5% of rating-2 completions in the diagnostic sample are accepted (Table 9). We therefore report deployment behavior alongside offline correctness.

Metric	Prev.	New 4B	Δ
Offline shown-precision	22.0	**38.6**	+16.5
Live acceptance†	17.8	**26.35**	+8.6
Median latency	100	**29**	−71%

Table 3: **Deployment comparison of the previous and 4B models.** Deltas use unrounded values; latency is relative. † Matched seven-day windows.

3.6 Error Analysis

To identify the errors behind the grounding gain in Section 3.3, we audit new samples from the two SFT variants compared there. On partially predictable requests, our evaluator finds that grounding reduces unsupported identifiers from 14.5% to 6.1%, hallucinated literals from 52.0% to 12.6%, and boundary overruns from 19.6% to 12.1%. Overconfident guesses on decline requests also fall from 20.9% to 11.5%. Full confidence intervals are reported in Table 7.

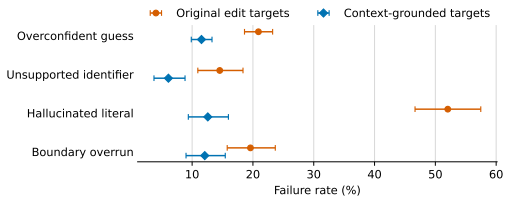


Figure 5: **Context-grounded targets reduce errors caused by supervision mismatch.** Request-clustered rates and 95% CIs; denominators differ by row as detailed in Table 7.

4 Related Work

Code and SQL completion. Fill-in-the-middle (FIM) models condition on both sides of a cursor (Bavarian et al., 2022; Chen et al., 2021; Fried et al., 2023; Li et al., 2023c). They have scaled through open code models (Lozhkov et al., 2024; Hui et al., 2024; Guo et al., 2024; Rozière et al., 2023), completion-specific releases (ByteDance Seed, 2025; Cao et al., 2026), and cross-file retrieval (Zhang et al., 2023; Ding et al., 2023; Liu et al., 2023; Wu et al., 2024); syntax-aware benchmarks evaluate FIM directly (Gong et al., 2024). For SQL, SnipSuggest mines log snippets (Khoussainova et al., 2010), whereas text-to-SQL maps a complete question to an executable query (Yu et al., 2018; Li et al., 2023b;a; Pourreza & Rafiei, 2023; Gao et al., 2023a; Lei et al., 2025). These settings typically evaluate a known target in a well-formed context; mid-edit SQL may not parse, has no explicit request, and sometimes requires an empty output.

Production code and SQL assistants. Deployed assistants include IntelliCode Compose (Svyatkovskiy et al., 2020), models trained from logged completion events (Aye et al., 2021), CodeCompose and its multi-line extension (Murali et al., 2024; Dunay et al., 2024), Copilot acceptance studies (Ziegler et al., 2022), and the closely related SqlCompose (Maddila et al., 2025). These works demonstrate the value of real usage data and of timing and length (Mozannar et al., 2024). We address an earlier question: which logged edits are legitimate learning targets, and how abstention can be trained rather than imposed by a serving heuristic.

Abstention, calibration, and reward hacking. Calibrated abstention extends selective prediction to trustworthy generation (Geifman & El-Yaniv, 2017; Kalai et al., 2025). Common methods threshold sequence probability, self-reported confidence, or semantic uncertainty (Ren et al., 2023; Kadavath et al., 2022; Kuhn et al., 2023); Section 3.4 compares against this approach. Grounding benchmarks instead test contextual support (Jacovi et al., 2025; Tang et al., 2024). We move both ideas into the learning signal so that grounded abstention is trained directly. Code RL typically uses execution rewards (Le et al., 2022; Gehring et al., 2025), unavailable for incomplete SQL, or overlap metrics such as BLEU (Papineni et al., 2002). LLM judges can capture intent beyond string overlap (Ouyang et al., 2022; Zheng et al., 2023; Li et al., 2024), but policies can exploit misspecified rewards (Skalse et al., 2022; Pan et al., 2022; Gao et al., 2023b); even user acceptance is gameable (Mozannar et al., 2024). We apply the same gate-first reward with GRPO (Shao et al., 2024; Guo et al., 2025) and single-rollout PPO-style optimization (Schulman et al., 2017; Hou et al., 2026).

5 Conclusion

Context grounding resolves a supervision mismatch in autocomplete: user editing logs contain completions that depend on information unavailable at the cursor. Enforcing one constraint across SFT and RL (never learn from or reward content not inferable at the cursor), SFT trims targets and adds silence labels while RL applies deterministic gates before the LLM judge. Trained without customer data, the resulting 4B dense model replaces a 30B-A3B MoE with higher precision and acceptance and 71% lower median latency.

Ethics and Broader Impact Statement

All evaluation and training data are drawn from internal employee editing sessions; *no customer query data* is used for training or evaluation. Identifiers and sensitive values are anonymized before training (Section 2.2). We mitigate privacy risks by excluding customer data, anonymizing internal data, training calibrated abstention, and requiring explicit user acceptance before any suggestion is inserted. The research conforms to the ICLR Code of Ethics.

Reproducibility and Data Availability Statement

Section 2 describes the reconstruction and training pipeline. The Appendix provides deterministic gate definitions, data processing procedures, evaluator rules, and training hyperparameters (Sections B to E and K). All public and larger instruct baselines use the same generation interface, abstention policy, reconstructed references, and offline evaluator as our models (Table 1). Code and a de-identified data subset will be publicly released upon publication.

AI Use Statement

Large language models were used to polish grammar and phrasing in this manuscript. LLMs are also core experimental components for reference reconstruction, offline evaluation, and reward judging; their roles and model identities are documented in Sections 2.1, 2.3 and C.

References

Gareth Ari Aye, Seohyun Kim, and Hongyu Li. Learning autocompletion from real-world datasets. In *IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2021.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.

ByteDance Seed. Seed-coder: Let the code model curate data for itself. *arXiv preprint arXiv:2506.03524*, 2025.

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, Zeyao Ma, Kashun Shum, Xuwu Wang, Jinxi Wei, Jiaxi Yang, Jiajun Zhang, Lei Zhang, Zongmeng Zhang, Wenting Zhao, and Fan Zhou. Qwen3-coder-next technical report. *arXiv preprint arXiv:2603.00729*, 2026. doi: 10.48550/arXiv.2603.00729. URL https://arxiv.org/abs/2603.00729.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Omer Dunay, Daniel Cheng, Adam Tait, Parth Thakkar, Peter C. Rigby, Andy Chiu, Imad Ahmad, Arun Ganesan, Chandra Maddila, Vijayaraghavan Murali, Ali Tayyebi, and Nachiappan Nagappan. Multi-line AI-assisted code authoring. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion)*, 2024.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. Incoder: A generative model for code infilling and synthesis. In *International Conference on Learning Representations (ICLR)*, 2023.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-SQL empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*, 2023a.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023b.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding code LLMs in execution feedback with reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Linyuan Gong, Sida Wang, Mostafa Elhoushi, and Alvin Cheung. Evaluation of LLMs on syntax-aware code fill-in-the-middle tasks. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

Daya Guo, Qihao Zhu, Dejian Yang, et al. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, et al. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Zhenyu Hou, Yujiang Li, Jie Tang, and Yuxiao Dong. Single-rollout asynchronous optimization for agentic reinforcement learning. *arXiv preprint arXiv:2607.07508*, 2026.

Binyuan Hui, Jian Yang, Zeyu Cui, et al. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Alon Jacovi, Andrew Wang, Chris Alberti, Connie Tao, Jon Lipovetz, Kate Olszewska, Lukas Haas, Michelle Liu, Nate Keating, Adam Bloniarz, et al. The facts grounding leaderboard: Benchmarking llms' ability to ground responses to long-form input. *arXiv preprint arXiv:2501.03200*, 2025.

Saurav Kadavath, Tom Conerly, Amanda Askell, et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

Adam Tauman Kalai, Ofir Nachum, Santosh S Vempala, and Edwin Zhang. Why language models hallucinate. *arXiv preprint arXiv:2509.04664*, 2025.

Nodira Khoussainova, YongChul Kwon, Magdalena Balazinska, and Dan Suciu. Snipsuggest: Context-aware autocompletion for SQL. *Proceedings of the VLDB Endowment*, 4(1), 2010.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *International Conference on Learning Representations (ICLR)*, 2023.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, et al. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *International Conference on Learning Representations (ICLR)*, 2025.

Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. Llms-as-judges: a comprehensive survey on llm-based evaluation methods. *arXiv preprint arXiv:2412.05579*, 2024.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023a.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, et al. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023b.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, et al. Starcoder: May the source be with you! *Transactions on Machine Learning Research (TMLR)*, 2023c.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

Chandra Maddila, Vijayaraghavan Murali, Nikolay Pavlovich Laptev, et al. AI-assisted SQL authoring at industry scale. In *IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2025.

Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. When to show a suggestion? integrating human feedback in AI-assisted programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2024.

Vijayaraghavan Murali, Chandra Maddila, Imad Ahmad, Michael Bolin, Daniel Cheng, Negar Ghorbani, Renuka Fernandez, Nachiappan Nagappan, and Peter C. Rigby. AI-assisted code authoring at scale: Fine-tuning, deploying, and mixed methods evaluation. *Proceedings of the ACM on Software Engineering (FSE)*, 2024.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. In *International Conference on Learning Representations (ICLR)*, 2022.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.

Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Jie Ren, Jiaming Luo, Yao Zhao, Kundan Krishna, Mohammad Saleh, Balaji Lakshminarayanan, and Peter J. Liu. Out-of-distribution detection and selective generation for conditional language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2020.

Liyan Tang, Philippe Laban, and Greg Durrett. Minicheck: Efficient fact-checking of llms on grounding documents. *EMNLP*, 2024.

Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. Repoformer: Selective retrieval for repository-level code completion. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. Repocoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (MAPS)*, 2022.

A LIMITATIONS

We study one SQL dialect and one product surface, so absolute results may not transfer to other editors, languages, or user populations. The reconstruction, grounding, and reward framework could be adapted to other settings, but its SQL-aware checks and model-facing interfaces require task-specific implementation and validation. Both offline evaluation and RL depend on LLM judges and therefore inherit judge bias. Deterministic gates reduce the policy's opportunity to exploit that bias but cannot eliminate residual judge errors.

The deployment comparison uses matched seven-day windows, so temporal traffic and interface effects may contribute to the observed differences. Live acceptance depends on UI, latency, and timing in addition to correctness, so offline ratings can explain it only partially. The current system also has limited support for corrective editing: when the prefix or suffix around the cursor already contains an error, the model may continue from that error instead of correcting it. Improving this correction ability is an important direction for future work.

B GOLD-CATEGORY TAXONOMY AND RECONSTRUCTION PIPELINE

Figure 6 expands the two-stage reconstruction described in Section 2.1. Stage 1 stores the extracted raw edit as RAW_GOLD and assigns one of {`match`, `already_complete`, `blank_slate`, `no_match`}. For matched rows, Stage 2 produces the final reference and refined category (Table 4). The final reference is stored as REASONABLE_GOLD for evaluation and used as the SFT target; the RL abstention gates use whether it is empty.

Candidate retrieval and deterministic filters. Candidate retrieval considers queries that the same user successfully executed within 20 minutes of the trigger. It retains queries containing the last prefix line whose whitespace-normalized prefix edit distance is below 10 against an equal-length query prefix; when a suffix exists, it applies the analogous suffix check. The earliest best-aligned query becomes the candidate. After model-assisted reconstruction, we retain only accepted matches marked as training-worthy, restore escaped target whitespace, and remove nonempty targets containing leaked reasoning tags or placeholder values. Each remaining nonempty target must occur in the matched executed query after whitespace normalization, must not duplicate the suffix, and must be at most 270 characters. These checks leave approximately 351k training examples.

Manual checks of reconstructed references. Because reconstruction errors would propagate to every model comparison, an author spot-checked reconstructed references against human judgment during pipeline development. The checks covered the matched executed query, cursor-level completion span, context-grounding trim point, and gold category, including the fire-versus-decline decision. These diagnostic checks exposed systematic extraction errors that were corrected before the evaluation sets were frozen.

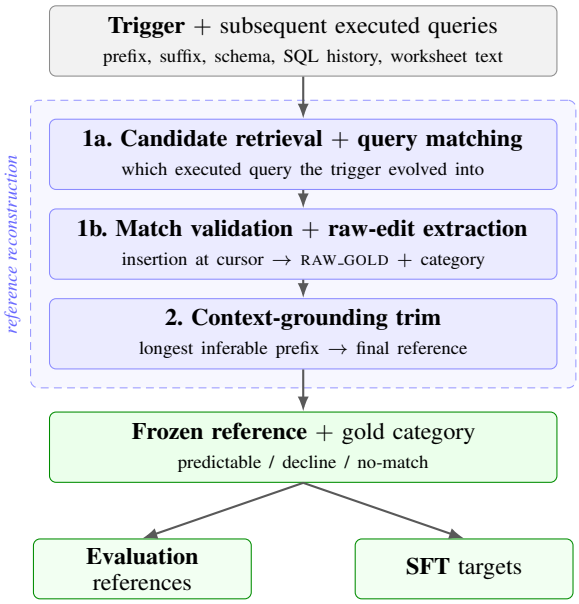


Figure 6: **One reconstruction pipeline produces both evaluation references and SFT targets.** Stage 1 combines deterministic candidate retrieval with LLM-assisted matching, validation, and raw-edit extraction. Stage 2 trims that edit to its longest context-grounded prefix.

Gold category	Meaning
`fully_predictable`	Entire intended edit is inferable from context.
`partially_predictable`	Only a prefix of the edit is inferable; trim to it.
`unpredictable`	Edit depends on information absent from context; target is empty.
`structurally_unreasonable`	Raw edit is not a plausible inline completion at this cursor (e.g., an unjustified clause jump); target is empty.
`already_complete`	Prefix + suffix already form a valid query; target empty.
`blank_slate`	Empty/new-statement position with no basis to predict; target empty.
`no_match`	No executed query aligns with the trigger; excluded.

Table 4: Gold-category taxonomy produced during reference reconstruction. In the frozen evaluation sets, `structurally_unreasonable` is grouped with `unpredictable`.

Post-freeze reference audit. To assess the fire-versus-decline decisions induced by the references reconstructed by Claude Sonnet 4.6 (Fig. 6), one author manually reviewed a uniform random sample of 150 unique evaluation requests. For each request, the author examined the cursor-time context available to the autocomplete model and labeled it *suggest* if at least one nonempty, context-supported completion could be safely inserted, and *decline* otherwise. The frozen decision disagreed with the manual label on $10/150$ requests. A recurring mismatch occurred when reaching the later executed query required replacing text already present at the cursor rather than simply inserting new text. The reconstruction pipeline nevertheless treated the newly added text as an insertable completion. For example, when the prefix ended with `usage` and the later query contained `user_name`, the reconstructed reference began with `user_name`; inserting it at the cursor would produce the invalid token `usageuser_name`, whereas obtaining the later query would require first deleting or replacing `usage`. Overall, the frozen fire-versus-decline labels agree with manual judgment on 93.3% of audited requests, with the remaining noise concentrated in such reconstruction edge cases.

C OFFLINE EVALUATOR

The frozen reference defines the intended edit, but exact match alone would miss useful alternatives and partial progress. We therefore have Qwen3.6-27B-FP8 rate each completion against the trusted reference package and return one failure or progress category. The package also contains TYPO_STATE, which records a prefix or suffix boundary typo. Table 5 defines the categories used for error analysis.

Category	Meaning
`false_prediction`	Fired when the query was already complete. This evaluator label is narrower than the false-fire metric, which covers all decline requests.
`laziness`	Empty output where a predictable, non-typo edit existed.
`overconfident_guess`	Non-empty, misaligned output at an unpredictable/blank position.
`wrong_direction`	Fluent completion heading toward a different edit.
`invalid_table_column`	References a schema object not in context.
`hallucinated_literal`	Invents an ungrounded string/number/date literal.
`semantic_error`	Logically wrong (e.g. `WHERE 1=1`, $\star$ mixed with columns).
`syntax_error`	Combined SQL does not parse / merged tokens at the junction.
`duplicated_query`	Re-types content already present in the context.
`conflicting_ground_truth`	Fired at a `no_match` position, where the executed query conflicts with the cursor context; the reference is unreliable and such rows are excluded from all metrics.

Table 5: Offline-evaluator failure categories (rating 0). Categories are gated by gold category to prevent inconsistent labels.

Evaluator design principles. Beyond freezing the reference once, three rubric choices define the evaluator's behavior:

1. **Credit verified alignment.** A completion that continues the selected executed query can receive credit even when predictability trimming made the final reference empty; `no_match` and `already_complete` remain exceptions.
2. **Anchor on the first committed identifier and inspect cursor junctions.** If a completion chooses `stats_minute` where the executed query uses `stats_hour`, it receives rating 0 even when both tables appear in context. A correct opening followed by unsupported or contradictory content also receives 0. Character-level checks catch fused tokens such as prefix `...= 1` plus completion `group by`, which produces `1group` but can be hidden by whitespace normalization. These rules block a strategy of aligning the first token and then inventing the remainder.
3. **Constrain labels by gold category.** A legality table rules out inconsistent category–failure pairs, such as `laziness` at an `already_complete` position.

Deterministic boundary post-check. Normalized comparisons can hide cursor-boundary errors, so a rule-based post-check inspects outputs that Qwen initially rates 2. It downgrades an output to rating 0 when direct concatenation either fuses distinct tokens across the prefix–completion boundary or causes the completion to repeat the normalized beginning of the suffix. These cases are labeled `syntax_error` and `duplicated_query`, respectively. The check never promotes a rating or changes a rating-1 output.

D CONTEXT-GROUNDING DETAILS

The offline trim and online gate implement the same grounding principle at different scopes. SFT trimming checks both identifiers and literals and cuts a raw edit at the first unsupported item. The deployed RL gate is deliberately deterministic: it checks introduced table/column entities and duplication, while the RL judge handles unsupported literals. The RL gate examines only the completion span. It:

1. tokenizes `prefix + completion + suffix` in the target SQL dialect;
2. extracts table and column identifiers whose tokens overlap $[\,|\text{prefix}|,\ |\text{prefix} + \text{completion}|\,)$;

3. removes SQL keywords, aliases, CTE names, and function calls;
4. adds words that cross the prefix–completion or completion–suffix boundary to both the entity list and context, so valid token continuations remain grounded;
5. combines prefix, suffix, SQL history, DDL, worksheet text, and recent executed queries into the remaining context string; and
6. requires every extracted entity to match that context at a word boundary.

Let u be the fraction of extracted entities that are ungrounded, with $u = 0$ when no entity is extracted. We mark a completion as duplicate when the whitespace-normalized full SQL, `prefix + completion + suffix`, has more than ten characters and appears in the neighboring worksheet text. The gate sets reward to 0 when $u > 0$ or when a non-exact completion produces such a duplicate.

Completion (given context)	**Grounded?**	**Kept**
`AND status = 'active'` (`'active'` in history)	yes	full
`AND region = 'APAC-7'` (`'APAC-7'` unseen)	no	trim before literal
`JOIN dim_users u ON` (`dim_users` in DDL)	yes	full
`GROUP BY acct_id` (`acct_id` nowhere)	no	empty

Table 6: Illustrative grounding decisions. Only the longest prefix supported by context is kept as a training target.

Error audit for original and context-grounded targets. To test whether target grounding changes the specific errors predicted by our supervision-mismatch hypothesis, we compare the two Qwen3-4B SFT models from Section 3.3. One uses the original cursor edits reconstructed from later queries as training targets, while the other restricts those targets to information available at the cursor. We regenerate outputs on two frozen traffic periods with temperature 0.7, top-$p = 0.8$, and $k = 3$.

Our Qwen3.6-27B-FP8 evaluator assigns all labels in this audit with model identity hidden. For decline requests, we use its primary label to measure decision-side overconfidence. For every non-empty output on a partially predictable request, the evaluator may assign one or more content-grounding labels; all $1{,}635$ outputs parse successfully. *Unsupported identifier* and *hallucinated literal* mark table or column names and string, numeric, or date values that are not supported by cursor-time context. *Boundary overrun* means that a completion first makes context-supported progress and then continues past the grounded reference boundary. The final row is the union of these three labels.

For every row we aggregate the indicated numerator and denominator within each request and use a paired percentile cluster bootstrap with $10{,}000$ replicates (seed 0); all three generations for a request move together. We use decline draws for overconfident guessing and non-empty outputs from partially predictable requests for the remaining error rates.

Failure type	**Denominator**	**Raw**	**Grounded**	**Δ (G−R), 95% CI**
Overconfident guess	decline draws	20.90	11.54	-9.37 $[-11.06, -7.74]$
Unsupported identifier†	partially predictable fires	14.54	6.10	-8.44 $[-11.90, -5.11]$
Hallucinated literal†	partially predictable fires	52.04	12.58	-39.47 $[-45.38, -33.69]$
Boundary overrun†	partially predictable fires	19.59	12.08	-7.51 $[-11.41, -3.76]$
Any grounding violation†	partially predictable fires	71.63	27.40	-44.24 $[-50.16, -38.34]$

Table 7: **Failure rates for SFT with original and context-grounded targets** (%). Negative differences favor context-grounded SFT. † Our evaluator assesses non-empty outputs from partially predictable requests with model identity hidden. Labels may overlap, and the union covers unsupported identifier, hallucinated literal, and boundary overrun.

E Reward Gate Pipeline

To make the ordering and shaping of the gate-first reward explicit, Algorithm 1 gives the deployed algorithm. With judge weights $w = (1, 1, 2, 1, 1)$ for (syntactic, intent, semantic, completeness, boundary) and intent in $\{0, 1, 5\}$, the maximum weighted total is 10. The reward normalizes this total

to $[0, 1]$ and squares it, applies a false-fire penalty of -0.5, and therefore has range $[-0.5, 1]$ with exact match returning 1. A final deterministic boundary gate returns zero when the completion ends in triple single, double, or backtick quotes, or when its presence or absence of leading whitespace differs from the final reference. The SQL-BLEU ablation reuses these gates and replaces only the final graded score (Section F).

Algorithm 1 Gated LLM-judge reward

Require: completion c, final reference g, context $\mathcal{C}$
$\text{empty}(x) \leftarrow [x = \varnothing$ **or** $\text{strip}(x) =$ `""`$]$
if $c = g$ **then**
  return 1.0 {exact match, incl. $c = g = \varnothing$}
end if
if $\text{empty}(g)$ and not $\text{empty}(c)$ **then**
  return -0.5 {false fire}
end if
if not $\text{empty}(g)$ and $\text{empty}(c)$ **then**
  return 0 {laziness}
end if
if $u(c, \mathcal{C}) > 0$ **or** $\text{duplicate}(c, \mathcal{C})$ **then**
  return 0 {grounding}
end if
if trailing triple quotes **or** leading-whitespace mismatch with g **then**
  return 0
end if
return $\big(J_{\text{LLM}}(c, g, \mathcal{C})/10\big)^2$ {weighted judge total $J_{\text{LLM}} \in [0, 10]$}

Why -0.5 and squaring. Both choices reduce the relative reward for firing. Consider a simplified fire-or-decline decision in a context where the probability of an empty reference is q. To isolate reward shaping, suppose a non-exact fired completion would pass the remaining gates when the reference is nonempty, and let $\bar{s}$ denote its expected graded score. Declining has expected reward q, whereas firing has expected reward $(1 - q)\,\bar{s} + q\,r_{\text{ff}}$, where r_{ff} is the false-fire reward. The two actions are tied at

$$q = \frac{\bar{s}}{1 + \bar{s} + |r_{\text{ff}}|}.$$

With a linear judge and $r_{\text{ff}} = 0$, non-exact gate-passing completions had mean scores of approximately 0.68–0.82, giving a tie point of $q \approx 0.40$–0.45. This makes firing the higher-reward action below that range, consistent with the over-firing observed in those runs.

The deployed reward instead uses $r_{\text{ff}} = -0.5$ and squares the normalized judge score. Squaring preserves scores 0 and 1 while reducing partial credit, for example from 0.6 to 0.36. Using the observed post-squaring mean $\bar{s} \approx 0.65$, the same approximation gives a tie point near $q = 0.30$. This calculation explains the direction of the change but is not an exact model of training, because actual completions may fail other gates and decisions depend on context. The 2×2 ablation provides the empirical check: each choice reduces false-fire at a fixed setting of the other, and their combination reduces it from 26.7% to 25.0% (Table 8).

False-fire reward	Judge shape	Prec.	False-fire	R2 len
0	linear	33.1	26.7	13.0
-0.5	linear	32.6	25.6	12.9
0	squared	33.0	26.0	12.2
-0.5	squared *(deployed)*	33.6	**25.0**	11.7

Table 8: **Reward-shaping ablation (2×2).** The false-fire penalty (0 vs. -0.5) and squaring of the judge score are swept under the Table 2b protocol (%; **R2 len** in characters). Precision stays within one percentage point across the grid, and each shaping choice reduces false-fire at a fixed setting of the other; their combination is lowest. Together the shaping choices reduce false-fire by 1.7 points, compared with the 6.0-point increase from removing the false-fire gate (Table 2b).

F SQL-BLEU ALGORITHM

This ablation tests whether a deterministic overlap score can replace the LLM judge after the same grounding and abstention checks. It therefore uses the gate pipeline in Algorithm 1: an exact string match returns 1; a nonempty completion against an effectively empty reference returns -0.5; and an empty completion against a nonempty reference returns 0. The entity-grounding and duplication gate, trailing-triple-quote check, and leading-whitespace check are also unchanged. Only a non-exact completion that passes every gate reaches the graded score, where this ablation returns SQL-BLEU rather than the squared LLM-judge score.

Boundary completion and tokenization. The n-gram calculation tokenizes the generated and reference *fragments*, not the full queries. When the reference connects directly to a partial word in the prefix or suffix, the same boundary fragment is attached to both fragments before tokenization. The SQLGlot-based tokenizer preserves original case and whitespace spans as tokens, together with SQL operators, punctuation, quoted literals, and compound keywords. The AST branch separately parses `prefix + raw fragment + suffix` in the target dialect.

Structural first-order term. After the exact-match check, the first precision term p_1 is structural rather than clipped unigram precision:

- If both full strings parse and the reference AST contains at least two nodes, $p_1 = \max(0, 1 - E/N)$, where E is the number of AST edit operations and N the number of reference-AST nodes.
- If a full parse fails, the implementation finds the longest parseable prefix of each full string. When both exist, it combines their AST similarity with matching-block precision on their remaining fragment tokens, weighted by the mean parsed and unparsed token counts across generation and reference.
- In the remaining cases, including when either side has no parseable prefix or the full reference AST has fewer than two nodes, it uses matching-block precision on the complete fragment-token sequences. Concretely, Python `SequenceMatcher` produces matching blocks with total size M, and $p_1 = M/m$, where m is the number of generated tokens.

The implementation records both full-AST and hybrid cases as tier 1, and the matching-block fallback as tier 2.

BLEU aggregation. Let m and n be the generated and reference token counts after boundary completion, and set $K = \min(4, m)$. For $k = 2, \ldots, K$, p_k is standard clipped k-gram precision. If $p_1 = 0$, the score is 0, except that token sequences differing only in case return the case multiplier defined below. Zero values among $p_2, \ldots, p_K$ receive NIST Method-3 smoothing, successively replacing them by $1/2, 1/4, \ldots$; p_1 is never smoothed. The final score is

$$\exp\Big(\min\Big(0, 1 - \frac{n}{m}\Big)\Big) \exp\left(\frac{1}{K}\sum_{k=1}^{K} \log p_k\right) 0.9^d,$$

where d counts aligned token positions that differ only in case. Thus the score is sensitive to both whitespace and case. For the special $p_1 = 0$ case in which the lowercased token sequences are identical, the implementation returns 0.9^d directly.

G EVALUATION METRICS AND PROTOCOL

The body reports complementary measures of shown quality and firing decisions; here we specify their exact denominators. A generation is valid when its trigger has a reliable reference, so `no_match` triggers are excluded. All metrics pool individual generations except pass@k, which groups the k

draws from each trigger:

$$\text{precision} = \frac{\#\{\text{fired and } r{=}2\}}{\#\{\text{fired}\}}, \qquad \text{pass@1} = \frac{\#\{r{=}2\}}{\#\{\text{valid}\}},$$
$$\text{pass@}k = \frac{\#\{\text{req. with any } r{=}2 \text{ in } k\}}{\#\{\text{req.}\}}, \qquad \text{decline} = \frac{\#\{\text{empty}\}}{\#\{\text{valid}\}},$$
$$\text{mean } r = \frac{1}{\#\{\text{valid}\}} \sum_{i \in \text{valid}} r_i.$$

For the gold-aware metrics, define a per-draw pass bit as

$$b_i = \begin{cases} \mathbf{1}[\text{fired and } r_i = 2], & i \in \text{suggest}, \\ \mathbf{1}[\text{empty or } r_i = 2], & i \in \text{decline}. \end{cases}$$

During offline evaluation, an empty reference does not imply that every nonempty completion is incorrect. Even when predictability trimming removes the matched edit, a model may happen to generate a useful completion that aligns with the query the user later executed. If the offline evaluator verifies that alignment, it may assign $r_i = 2$, and we count the output as a pass. This is retrospective credit for a correct guess, not evidence that the edit was predictable at the cursor; the exception does not apply to `already_complete` requests.

Suggest and decline pass@1 average b_i within their respective subsets, and blended pass@1 averages it over all valid draws. False-fire is the fraction of decline draws that are nonempty.

Laziness is the empty-output rate at predictable, non-typo positions; correct silence is the empty-output rate on `already_complete` and `blank_slate` requests. Length is averaged separately over fired and rating-2 suggestions.

We sample $1{,}000$ triggers from each of two distinct frozen traffic periods. For each trigger, we draw $k = 3$ generations and exclude reconstruction-labeled `no_match` rows because their references are unreliable. For the evaluation sets, this label is produced by Claude Sonnet 4.6 before any model outputs are scored.

Precision confidence intervals. We compute 95% percentile cluster-bootstrap intervals by resampling trigger-level *request_ids* with at least one fired output $B = 10{,}000$ times with random seed 0. All $k = 3$ generations from a trigger move together, and the point estimates reproduce Table 1.

For paired comparisons among SFT, GRPO, and SAO, and between GRPO and the previous 30B model, we bootstrap the *paired precision difference*. Each replicate samples from the sorted union of request IDs on which either model fires, recomputes both models' precision, and takes their difference. A model that does not fire on a sampled trigger contributes neither a fired-output count nor a rating-2 count. Point differences use the unrounded full-set precisions. Because both models are evaluated on the same triggers, the paired bootstrap accounts for shared request-level variation. Marginal intervals can therefore overlap even when the paired-difference interval excludes zero.

H Offline-Evaluator Validation

Because all offline model comparisons depend on the evaluator, we test both its consistency with a second judge and its relation to recorded user behavior. First, we compare Qwen3.6-27B-FP8 with Claude Sonnet 4.6 on identical generations using the same fixed reconstructed references. Their ratings reach roughly 94% exact agreement. Quadratic-weighted κ, which adjusts agreement for chance and gives partial credit to adjacent ratings, is approximately 0.88. The total-variation distance, half the summed absolute difference between their three rating proportions, is at most 0.015. We therefore use this two-step protocol for all model comparisons.

Validation against real user behavior. Agreement between Qwen and Claude tests rating consistency; recorded product behavior provides a separate relevance check. In a diagnostic sample separate from the matched deployment windows, we pair each production completion's evaluator rating with its recorded accept-or-dismiss action. The sample contains $4{,}021$ decisions from one traffic week.

The two signals need not agree on every completion. The evaluator rates whether a completion is useful given the frozen context and reconstructed reference, whereas acceptance records a product

action that also depends on timing and user behavior. A user may dismiss a useful completion by continuing to type or may accept a completion and then edit it. We therefore test whether higher ratings correspond to higher acceptance rates, rather than treating acceptance as a correctness label.

Acceptance rises monotonically across evaluator ratings $0/1/2$: $9.3\% \to 15.2\% \to 51.5\%$, compared with a 21.1% overall rate (Table 9). Treating the rating as an ordinal score and accept/dismiss as the binary outcome gives area under the receiver-operating-characteristic curve (AUC) $= 0.755$. Here AUC is $P(r_{\text{accept}} > r_{\text{dismiss}}) + \frac{1}{2}P(r_{\text{accept}} = r_{\text{dismiss}})$, so 0.755 means that the rating orders an accepted completion above a dismissed one in 75.5% of pairwise comparisons when ties receive half credit. Among rating-2 completions, 51.5% were accepted, $2.44\times$ the overall acceptance rate.

Complementary association measures give the same direction. Spearman $\rho = 0.442$ is the rank correlation between rating and acceptance, while Kendall $\tau_b = 0.433$ measures concordant versus discordant pairs and adjusts for ties. Point-biserial $r = 0.451$ is the ordinary Pearson correlation between numeric rating $(0/1/2)$ and binary acceptance $(0/1)$. Somers' $D = 0.511$ is the directional rank association, computed here as $2\,\text{AUC} - 1$. These positive values are consistent with the monotonic acceptance rates in Table 9.

These two checks support using the evaluator for relative offline model comparisons: its ratings closely match Claude's on the same examples and rank recorded user acceptance well above the 0.5 chance level. They do not make acceptance a correctness label or remove the need to report live behavior separately.

Table 9: **The offline evaluator's rating tracks real user behavior.** Ratings on the $4{,}021$ production completions shown to employees over one traffic week, compared with the accept/dismiss actions those employees took. Real accept rate rises monotonically with the rating.

Evaluator rating	#	share (%)	real accept rate (%)
0 (bad)	2723	67.7	9.3
1 (safe/partial)	198	4.9	15.2
2 (strong)	1100	27.4	51.5
all	4021	100.0	21.1

Spearman ρ=0.442, Kendall τ_b=0.433, point-biserial (Pearson) r=0.451, AUC=0.755, and Somers' D=0.511.

I Per-Category Results

Aggregate suggest and decline scores can hide which cursor types improve. Table 10 therefore breaks the gold-aware pass@1 results in Table 1 down by gold category. GRPO and SAO are trained separately from the same SFT checkpoint.

	gold category	$n_{\text{S/G/A}}$	**SFT**	**GRPO**	**SAO**	Δ
suggest	fully predictable	1614/1613/1614	41.1	44.0	**46.9**	+5.8
	partially predictable	881/882/882	26.0	35.4	**37.1**	+11.1
	aggregate	2495/2495/2496	35.8	41.0	**43.4**	+7.7
decline	unpredictable	882/882/882	59.0	55.2	**60.2**	+1.2
	already complete	2226/2229/2226	88.2	**89.2**	87.6	−0.5
	blank slate	300/300/300	82.0	78.7	**84.0**	+2.0
	aggregate	3408/3411/3408	80.1	79.5	**80.2**	+0.1
	overall	5903/5906/5904	61.3	63.2	**64.7**	+3.3

Table 10: **Gold-aware pass@1 by gold category and model** (%). $n_{\text{S/G/A}}$ gives the actual valid-draw counts for SFT/GRPO/SAO; rare evaluator parse failures cause the small differences.
Each valid trigger has three attempted draws, and `no_match` triggers are excluded. The frozen evaluation sets group `structurally_unreasonable` with `unpredictable`. Aggregates reproduce S@1, D@1, and blended pass@1 in Table 1. Here Δ is SAO minus SFT and is computed before rounding. Bold marks the largest point estimate in each row.

The largest RL gains occur on partially predictable requests. GRPO improves over SFT by $+9.38$ points (95% CI $[+5.33, +13.46]$) and SAO by $+11.08$ ($[+7.11, +15.28]$). On fully predictable requests, the corresponding gains are $+2.94$ ($[+0.25, +5.67]$) and $+5.82$ ($[+3.16, +8.61]$). The difference between the partially and fully predictable gains is $+6.44$ points for GRPO ($[+1.64, +11.24]$) and $+5.26$ for SAO ($[+0.38, +10.23]$). Aggregate decline pass@1 changes by only $+0.1$ points from SFT to SAO, and its category-level changes range from -0.5 to $+2.0$ points.

J Confidence-Threshold Abstention Baseline

To test whether the RL policies' higher precision comes only from better abstention, we ask whether post hoc confidence filtering of grounded SFT outputs can recover the same precision–coverage tradeoff. This analysis uses a separately generated SFT sample for which token scores were retained; its unfiltered shown-precision is 33.1%, close to the 33.9% main SFT result in Table 1. We evaluate three scores. The first is mean sequence log-probability,

$$T^{-1} \sum_t \log p(y_t \mid y_{<t}, x).$$

The other two require only one direct forward pass at the first generation position. Let $\mathcal{E} = \{151643, 151645\}$ be the checkpoint's immediate-stop IDs (<|endoftext|> and <|im_end|>). We use $\log p_{\text{stop}} = \log \sum_{e \in \mathcal{E}} p(e \mid x)$ and the margin $\log p_{\text{stop}} - \max_{v \notin \mathcal{E}} \log p(v \mid x)$. Higher values of either score indicate a stronger preference for stopping. Of $1{,}968$ valid prompts, $1{,}966$ fit the original $32{,}256$-token serving limit and are scored; the other two also have no original generation.

For each score, a threshold may suppress an originally fired SFT completion but never revive an empty output. We choose the threshold on a fixed 50% request-level calibration split (seed 0) to make the firing rate as close as possible to GRPO's 49.19% full-set rate. Evaluator ratings are not used for this choice. The unfiltered SFT firing rate on the calibration split is already 48.93%, so the closest operating point for both first-step scores applies no additional suppression. On the full set this yields 49.39% firing and 33.10% precision. Mean sequence log-probability gives 48.93% firing and 33.24% precision. Table 11 reports the paired comparisons with both RL policies.

Policy / filtering score	Firing	Prec.	Δ vs. GRPO, 95% CI
SFT + mean sequence log-prob.	48.93	33.24	-5.31 $[-7.40, -3.23]$
SFT + first-step stop probability	49.39	33.10	-5.45 $[-7.52, -3.35]$
SFT + stop/content margin	49.39	33.10	-5.45 $[-7.52, -3.35]$
GRPO	49.19	38.55	reference
SAO	50.12	40.59	$+2.04$ $[+0.1, +4.0]$

Table 11: **Calibration-selected SFT confidence baselines versus RL** (%). Differences and intervals use 10,000 paired request-clustered bootstrap replicates. Relative to SAO, the three SFT baselines are lower by 7.35, 7.48, and 7.48 points, with intervals $[-9.51, -5.30]$, $[-9.62, -5.41]$, and $[-9.62, -5.41]$, respectively.

For comparison, selecting the closest first-step threshold on the full set using only firing rate gives 49.19% firing and 33.21% precision for both stop scores. Their gaps from GRPO remain -5.35 points, with paired intervals $[-7.40, -3.29]$ for stop probability and $[-7.48, -3.26]$ for the margin. Thus, neither the first-step stop scores nor sequence-level confidence accounts for the RL gain. Across the full sequence-confidence sweep, the highest firing rate attaining GRPO's 38.6% precision is only 32.1%.

K Hyperparameters and Infrastructure

For reproducibility, Table 12 lists the main settings and separates the models used for reconstruction, evaluation, and reward. Reference reconstruction for the evaluation sets runs on Claude Sonnet 4.6; the training set runs the same pipeline plus the training-worthiness filter at scale on Qwen3.6-27B-FP8. The offline evaluator is Qwen3.6-27B-FP8, and the RL reward judge is Qwen3-Coder-Next-FP8. The reward judge uses tensor parallelism (TP) $= 2$, JSON-object constrained decoding, temperature

0, a 128-token output limit, and disabled thinking; the offline evaluator instead uses structured-text outputs parsed with deterministic regular expressions. SFT uses example packing with a FIM loss mask on approximately 351k anonymized, context-grounded examples; RL uses no Kullback–Leibler (KL) term in the reward or loss and left-truncates over-length prompts.

Infrastructure. All training experiments use NVIDIA H200 GPUs with 141 GB of HBM per GPU. During RL, eight GPUs are used: a dedicated tensor-parallel pair serves the reward judge while the remaining six train the policy. A co-located inference engine generates rollouts, making reward-judge scoring the throughput bottleneck. A single SFT run takes approximately 8 hours; full-data GRPO training takes approximately 5 days; full-data SAO training takes approximately 3 days; and each SAO ablation takes approximately 18 hours. Evaluation generation draws $k = 3$ samples per trigger with temperature 0.7, top-$p = 0.8$, and at most 512 new tokens. The offline evaluator runs as eight single-GPU workers with prefix caching, deterministic decoding (temperature 0), and thinking disabled. Generating and evaluating the full evaluation set takes a few hours end to end.

Each public baseline is evaluated without task-specific fine-tuning and uses its native token family. The generation interface, reconstructed references, gold categories, evaluation sets, decoding configuration, and offline evaluator remain fixed.

L SAO Implementation Notes

To clarify how the single-rollout alternative in Section 2.3.2 differs from GRPO, we give its implementation details here. Single-Rollout Asynchronous Optimization (SAO) is a PPO-style optimizer with a learned critic. We initialize both actor and critic from the same SFT checkpoint and add a zero-initialized value head to the critic. The critic's attention parameters are frozen, while its multilayer-perceptron (MLP), embedding, and value-head parameters train. Token importance ratios are computed against rollout-engine log-probabilities. SAO uses Direct Double-Sided Importance Sampling (DIS), which masks tokens with ratios outside $[0.7, 6.0]$ rather than clipping them. Actor updates proceed only when the critic's five-step moving-average explained variance (EV) reaches 0.2. Advantages use $\gamma = 1$, a length-adaptive policy-side generalized advantage estimation (GAE) parameter $\lambda = \mathrm{clip}(1 - 1/(1.5L), 0, 1)$ for response length L, Monte Carlo critic returns, and no advantage whitening.

The final SAO run uses the full approximately 351k-example RL set for three epochs. We report the final checkpoint after the third epoch; the evaluation set is not used for checkpoint selection. Table 13 gives the detailed SAO settings used for this run.

Comparison with GRPO. GRPO uses eight rollouts per prompt, whereas SAO uses one; because judge scoring is the throughput bottleneck, SAO requires roughly $8\times$ fewer judge calls per prompt. Its 40.6% precision is 2.0 points above GRPO's 38.6% (paired 95% CI $[+0.1, +4.0]$). The deployed model uses GRPO; the SAO experiment evaluates the same reward with a lower-rollout optimizer.

Setting	Value
Base model	Qwen3-4B-Instruct-2507 (dense)
SFT data	≈351k anonymized, grounded
Supervised fine-tuning	
Learning rate	$1e{-5}$
Epochs	3
Sequence parallel / ZeRO	2 / stage 2
Packed max length	133,072
FIM loss mask	from <|fim_middle|>
Group Relative Policy Optimization (GRPO)	
Actor learning rate	$1e{-7}$
Train batch size	192
Proximal policy optimization (PPO) mini / micro batch	24 / 4
Group size (rollout n)	8
PPO clip ϵ / epochs	0.2 / 1
Max prompt / response len	14,000 / 1,024
KL in reward / loss	none
Rollout engine	co-located, TP $= 1$
Reward (judge)	
Criterion weights	$1, 1, 2, 1, 1$
Intent range	$\{0, 1, 5\}$; max total 10
Reward shaping	false-fire -0.5; judge^2

Table 12: Main training and reward hyperparameters.

Setting	SAO value
Initialization	Same context-grounded SFT checkpoint
Training data / epochs	≈351k / 3
Train batch size	192
Actor learning rate	$1e{-7}$
Actor PPO mini batch / micro batch per GPU	24 / 4
Critic learning rate	$5e{-6}$
Critic PPO mini batch / micro batch per GPU	96 / 4
Critic updates per policy step	2 minibatches (1 critic epoch)
Rollouts per prompt	1
Maximum prompt / response length	14,000 / 1,024
Prompt truncation	left; over-length prompts filtered
Policy / critic GAE λ	$\mathrm{clip}(1 - 1/(1.5L), 0, 1)$ / 1.0
Discount γ / advantage whitening	1.0 / none
DIS importance-ratio band	$[0.7, 6.0]$; mask outside band
Actor loss aggregation	sequence mean, then token mean
Critic value clipping	0.2
EV threshold / window	0.2 / 5 steps
Fixed critic warmup	0 steps (EV gate only)
Critic initialization	zero-initialized value head
Critic trainable modules	MLP, embeddings, and value head
Critic execution	packed forward; parameter and optimizer offload
KL in reward / actor loss	none
Rollout engine	co-located vLLM, TP $= 1$
vLLM memory utilization / sleep mode	0.4 / disabled
Training / reward-judge GPUs	6 / 2 (judge TP $= 2$)
Validation / checkpoint frequency	10 / 50 steps

Table 13: SAO hyperparameters for the final full-data run.