

PROCESS AUTOMATION TRAINING

A staged SFT + GRPO training methodology for an agentic case-to-invoice foundation model

A supervised fine-tuning stage that teaches the task's action grammar, followed by a GRPO stage that **directly optimizes goal-prediction and process-question-answering behavior**

AUTHOR

Neurologic AI

PUBLISHED

September 2026

BASE MODEL

Qwen/Qwen3.6-27B

ABSTRACT

Business-process automation agents built on large language models are typically never trained on the target process at all: they are deployed as out-of-the-box models steered purely through in-context prompting and workflow scaffolding,

an approach that can imitate the surface shape of a task but does not internalize its procedural detail, and leaves the agent unable to recover once it encounters a failure or exception the prompt did not anticipate. This paper describes a two-phase training methodology for a foundation model that automates the case-to-invoice billing workflow: a supervised fine-tuning (SFT) stage that teaches a base vision-language model the task's action grammar, entity vocabulary, and stepwise reasoning from recorded case traces, followed by a Group Relative Policy Optimization (GRPO) reinforcement-learning stage that directly optimizes goal-prediction and process-question-answering behavior against a large-language-model-as-judge reward. We detail the composition of the training corpora (49,853 supervised samples over 61 cases; 10,553 reinforcement-learning prompts over 68 cases), the parameter-efficient adaptation strategy used at each stage (LoRA under DeepSpeed ZeRO-2 for SFT; QLoRA under 4-bit NF4 quantization for GRPO), and the design of a two-rubric, per-family judge reward with an explicit format-penalty mechanism. We report held-out generation accuracy from the completed SFT run (90.4% correct goal prediction, 71.7% semantically correct step-level question answering) and full end-to-end validation evidence for the GRPO stage, including a live 40-generation trace showing the reward signal discriminates whenever candidate answers differ.

49,853

supervised samples
over 61 cases

10,553

reinforcement-
learning prompts
over 68 cases

90.4%

correct goal
prediction

71.7%

semantically correct
step-level question
answering

01

Introduction

Case-to-invoice processing (reviewing a case record, applying pricing and billability rules, and generating the resulting invoice documentation) is a high-volume back-office workflow that today depends on a person working through a sequence of screens, applying domain rules that are only partially codified, and producing outputs that must be both procedurally correct and auditable. As with other screen-mediated business processes, most automation

attempts targeting this workflow are never trained on the process at all: an out-of-the-box model is wrapped in an in-context workflow prompt and deployed directly, with no mechanism to internalize the process's procedural detail or to directly optimize the behavior the business actually cares about: correctly identifying the next goal in a case and answering process questions the way an expert biller would. Because such an agent has learned nothing about the task beyond what fits in its prompt, it has no way to recognize or recover from a failure or exception outside the cases the prompt happened to anticipate.

The work described in this paper takes a two-stage approach. First, a general-purpose vision-language model is supervised fine-tuned on tens of thousands of recorded case steps so that it learns the task's action vocabulary, entity grounding conventions, and the chain-of-thought structure the downstream system expects. This stage is analogous to imitation learning: the model learns to reproduce the distribution of actions and reasoning traces present in the training corpus. Second, the resulting checkpoint is used as the starting policy for a reinforcement-learning stage built on Group Relative Policy Optimization (GRPO), in which the model generates multiple candidate completions per prompt and an independent large-language-model judge scores them against a shared process playbook, allowing the policy to be pushed toward completions the judge (and by extension the encoded business rules) prefers, rather than merely toward completions that resemble the training distribution.

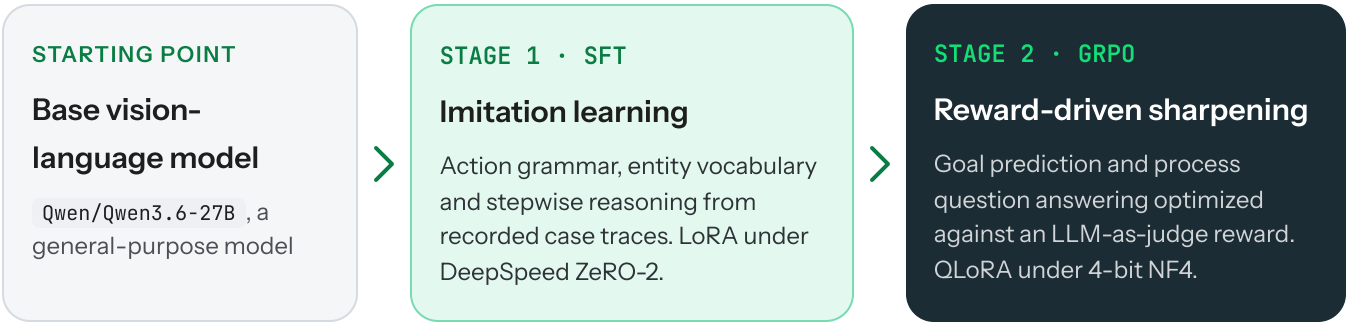


Figure 1. The staged design: an imitation-style phase that establishes broad competence, followed by a reward-driven phase that sharpens behavior against an explicit evaluation signal.

This staged design mirrors a broader pattern in modern foundation-model training: an imitation-style pretraining or fine-tuning phase that establishes broad competence, followed by a preference- or reward-driven phase that sharpens the model's behavior against an explicit evaluation signal. The remainder of this paper describes the data underlying each stage, the training methodology in detail, and the evidence gathered to date on whether (and how well) each stage achieves its goal.

Data

2.1 Overview

The two training stages draw on two distinct, non-overlapping corpora, split at the case level so that no case's steps appear in more than one partition. The supervised fine-tuning corpus is multimodal, pairing recorded case-processing screens with the case's chat-formatted action history; the reinforcement-learning corpus is text-only and is purpose-built for two specific task families rather than for general imitation.

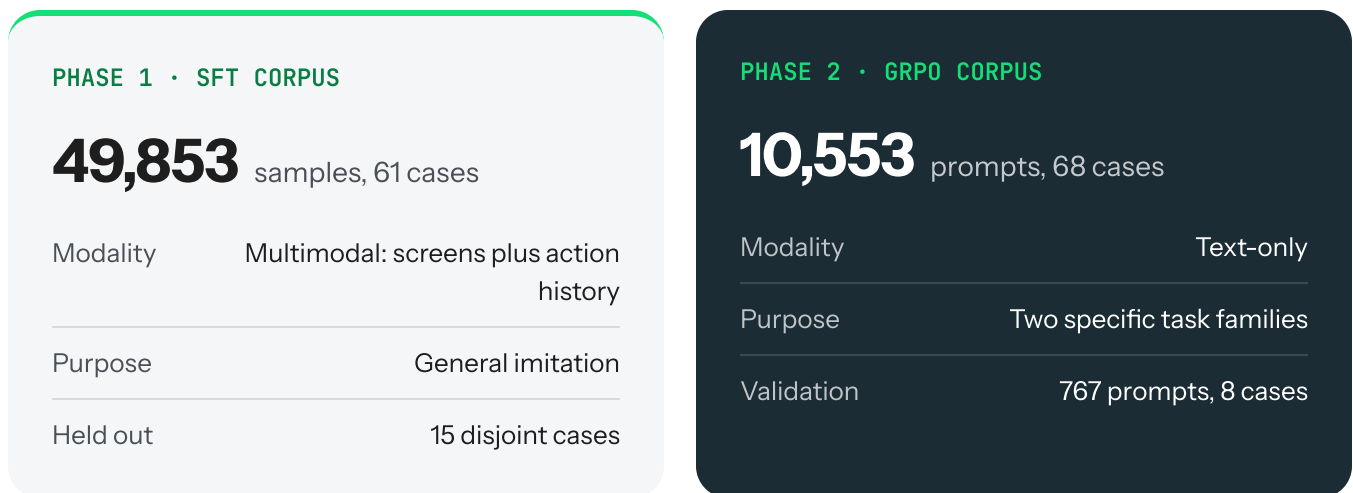


Figure 2. The two distinct, non-overlapping training corpora.

2.2 Phase 1: supervised fine-tuning corpus

The SFT corpus consists of **49,853 samples drawn from 61 cases**, held in a `training_v2/train` directory tree organized one subfolder per case with one JSON file per recorded step (or, alternatively, a manifest file listing absolute sample paths for reproducibility). Each sample is a chat-formatted record (a sequence of system, user, and assistant messages in an OpenAI-style schema, optionally wrapped in a `{"messages": [...], "metadata": {...}}` envelope) in which the user turn may include one or more embedded screenshots (base64-encoded image content) alongside text describing the case state, and the assistant turn contains the recorded action and reasoning. Approximately **79% of samples carry an accompanying screenshot**, with the remainder being text-only case-state turns. Training loss is masked to the assistant turn only (computed on tokens following `<im_start>assistant`), so the model is not penalized for reproducing the system or user context verbatim. The corpus was measured directly for

sequence length (99th-percentile token count $\approx 3,700$, maximum $\approx 5,000$ tokens against a configured maximum sequence length of 8,549), confirming that no samples are silently truncated during training.

One SFT sample: a chat-formatted record		One JSON file per recorded step
SYSTEM	System context	Loss masked
USER	Text describing the case state, plus one or more embedded screenshots (about 79% of samples)	Loss masked
ASSISTANT	The recorded action and reasoning	Loss computed here

Figure 3. Anatomy of an SFT sample. Training loss is masked to the assistant turn only.

Evaluation during training draws from a separate `training_v1/test` directory representing **15 fully disjoint held-out cases** (case-level split, so no case contributes to both training and evaluation), sub-sampled to a fixed, seeded 500-example subset per evaluation pass to bound evaluation wall-clock time without changing which cases are held out.

2.3 Phase 2: GRPO corpus

The reinforcement-learning corpus, internally named `training_v1_grpo_1.1`, is deliberately *not* the SFT corpus. It is text-only (no images) and comprises **10,553 training prompts over 68 cases and 767 validation prompts over 8 cases**, split by case with zero case overlap between partitions, independently re-verified by direct file count. The corpus is organized into two task families, routed deterministically by filename rather than inferred by the model or the judge:

Task family	Train count	Output shape
<code>goal_rm</code> (next-goal prediction)	10,292	JSON <code>{goal, steps}</code>
<code>workflow_qa</code> (process question answering)	261	Prose, 2–5 sentences

Table 1. GRPO training corpus composition by task family.

OUTPUT CONTRACT

Every sample in both families follows a fixed output contract: exactly one `<think>...` `</think>` reasoning block followed by the answer, with the requirement that the answer must stand on its own once the reasoning block is removed (i.e., the answer may not refer back to the discarded chain of thought). Mean prompt length across the corpus is approximately 2,555 to 2,561 tokens.

2.4 Dataset splitting

Both corpora are split strictly at the case level, never at the sample level, so that no individual case's steps, screens, or downstream invoice outcome can leak across the training/evaluation boundary. For the SFT stage this yields 61 training cases and 15 held-out cases; for the GRPO stage, 68 training cases and 8 validation cases. This mirrors standard practice in workflow and clinical modeling settings where a single entity (a patient, a case) may generate many correlated samples, and prevents an evaluation result from reflecting memorization of a specific case rather than generalization of the underlying process.

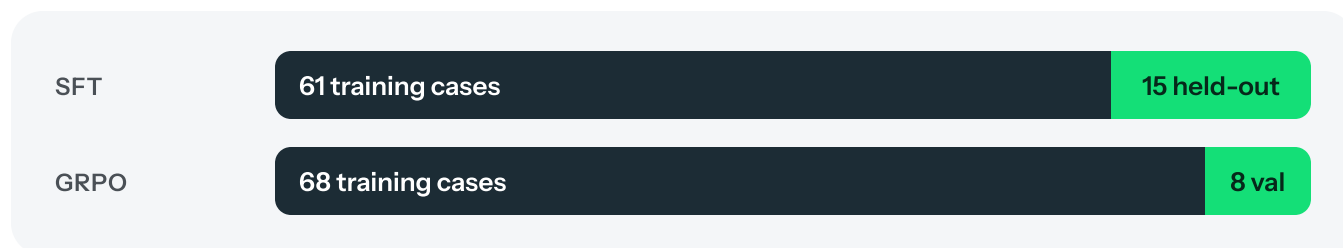


Figure 4. Case-level splits for each stage, drawn to scale by number of cases.

03

Model training

Both training stages adapt the same base model, `Qwen/Qwen3.6-27B` (a ≈ 274 -billion parameter multimodal model, HF architecture identifier `qwen3_5`), using parameter-efficient fine-tuning rather than full-parameter updates at every stage.

3.1 Stage 1: supervised fine-tuning (SFT)

Base model	<code>Qwen/Qwen3.6-27B</code> , bf16, vision tower frozen (<code>tune_merger: false</code>).
------------	--

Adapter	LoRA, rank 16, $\alpha = 32$, dropout 0.05, applied to the <code>q, v, o, gate, up, down</code> projection matrices: 156M trainable parameters, 0.6% of the full model.
Optimizer	AdamW ($\beta=0.9/0.999$), learning rate 0.0001, weight decay 0.01, cosine schedule with 3% warmup, gradient clipping at norm 1.0.
Batching	micro-batch size 1 per device $\times$ gradient accumulation 16×2 GPUs = global batch size 32.
Precision and parallelism	bf16 compute with tf32 enabled, DeepSpeed ZeRO stage 2. ZeRO-3 was evaluated and rejected: it re-gathers the frozen vision-tower parameters individually for every image and stalls under the deployment's MIG topology, which lacks direct peer-to-peer GPU access.
Hardware	2 $\times$ NVIDIA RTX Pro 6000 Blackwell GPUs, partitioned as MIG <code>4g.96gb</code> slices.
Schedule	2 completed epochs (1,558 steps/epoch, ≈ 47 seconds/step), checkpointed every 500 steps with a 5-checkpoint retention window.
Software	transformers 5.14.1, torch 2.13.0+cu130, DeepSpeed 0.19.3, PEFT 0.20.0; transformers ≥ 5.2 was required, as the <code>qwen3_5</code> architecture identifier is absent from earlier 4.57/5.0 releases.

	Epoch 1	Epoch 2
Train loss	0.481	0.389
Validation loss	0.4431	0.4247

Table 2. Phase 1 SFT loss by epoch (final run, `attempt_11`).

3.2 Stage 2: GRPO reinforcement learning

The GRPO stage continues training from the phase-1 LoRA adapter (verified byte-identical to the phase-1 checkpoint at global step 3000) rather than from the frozen base model, so the policy entering reinforcement learning already carries the task competence learned during SFT.

Quantization	QLoRA: base weights stored in 4-bit NF4 with double quantization, compute in bf16, only the LoRA adapter trains (in fp32). Audited on the loaded model rather than assumed from configuration: 606/606 quantized linear modules confirmed <code>nf4</code> with <code>bf16</code> compute dtype; 78,118,912 trainable parameters (0.28% of the base model), all LoRA. This reduces resident model memory from ≈ 53 GB (bf16) to ≈ 21.7 GB, which is what makes a completion batch size of 6 feasible on a single MIG slice. <code>prepare_model_for_kbit_training</code> is run <i>before</i> the adapter is attached, since running it after would freeze the newly-added LoRA weights along with everything else.
Adapter	LoRA rank 16, $\alpha = 32$, dropout 0.05 on the same <code>q, v, o, gate, up, down</code> projections used in phase 1; vision tower frozen.
GRPO configuration	learning rate 0.000001 (two orders of magnitude below the SFT learning rate), AdamW-8bit, cosine schedule, $\epsilon = 0.3$ clip range, $\beta = 0$ (no KL penalty to a reference policy), $G = 4$ generations per prompt, maximum completion length 3,072 tokens, generation temperature 1.0 (top- p 0.95, top- k 80).
Batching	6 completions per device $\times$ gradient accumulation 4 (single-GPU configuration) = 24 completions per optimizer step $\div G = 4$ generations = 6 prompts per step; one full epoch over the 10,553-sample corpus is 1,758 steps.

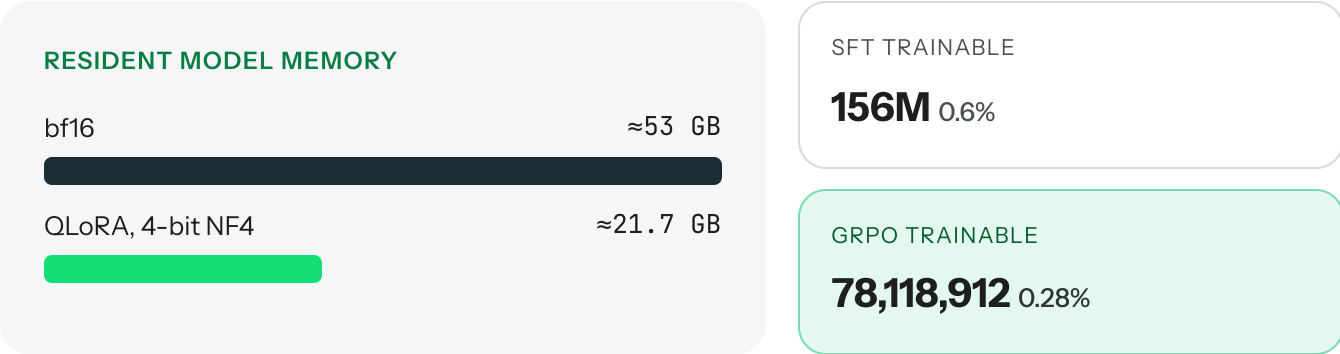


Figure 5. QLoRA reduces resident model memory enough to make a completion batch size of 6 feasible on a single MIG slice. Bars drawn to scale.

3.2.1 Reward design: LLM-as-judge

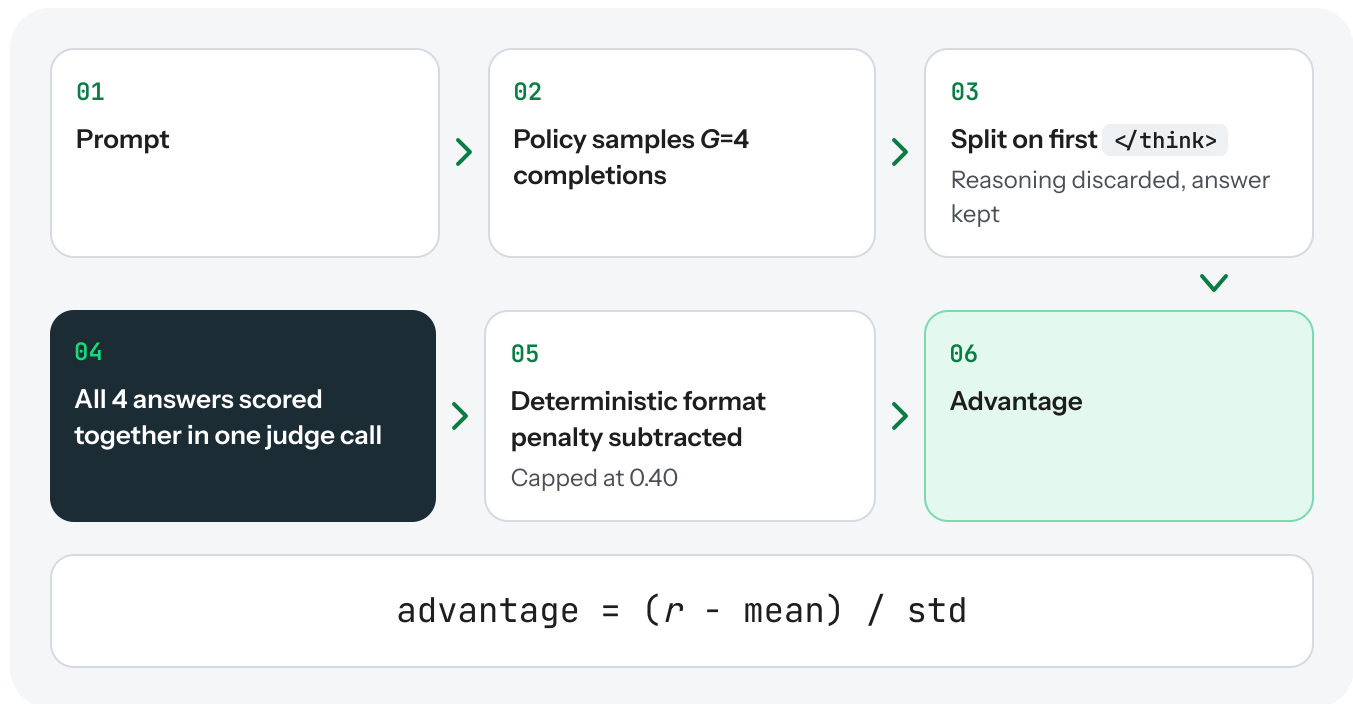


Figure 6. The reward pipeline, from prompt to group-normalized advantage.

The reward is produced by an independent large language model acting as judge, rather than by a learned reward model or a rule-based simulator. All four completions for a given prompt are scored in a single judge call so that the judge can rank them relative to one another rather than in isolation. Routing between the two task-specific rubrics is deterministic, based on the sample's family label, rather than inferred by the judge itself; an earlier design that asked the judge to classify the task before scoring it misclassified samples and applied the wrong rubric.

Judge model. Development and end-to-end validation were measured against Qwen3.6-122B-FP8 served via vLLM; production judging was migrated to DeepSeek V4 Pro, which reduced per-call judge latency from ≈ 92 s to ≈ 10 s largely because the DeepSeek deployment does not emit the $\approx 3,800$ tokens of discarded reasoning per call that the Qwen judge did ($\approx 89\%$ of its output). The judge runs at temperature 0.0 to reduce scoring variance.

PER-CALL JUDGE LATENCY

Qwen3.6-122B-FP8		≈92 s
DeepSeek V4 Pro		≈10 s

Figure 7. Migrating production judging to DeepSeek V4 Pro. Bars drawn to scale.

Rubrics. Two independent rubrics are used, each embedding the same underlying process playbook so that scoring is consistent with the document the training labels were generated against:

- **Next-goal prediction (`goal_rm`):** scored as a weighted sum over goal correctness (weight 4), step recall/precision (weight 4, $0.6 \times \text{recall} + 0.4 \times \text{precision}$), entity grounding against source identifiers (weight 2), and boundary discipline (staying within a single goal rather than drifting into the next; weight 1), normalized over a denominator of 11.
- **Workflow question answering (`workflow_qa`):** scored over factual correctness (weight 4), completeness across both branches of any conditional process step (weight 3), concrete grounding in real system/screen/control names rather than generic competence (weight 3), and output form (prose only, no headings or bullet lists; weight 1), again normalized over 11.

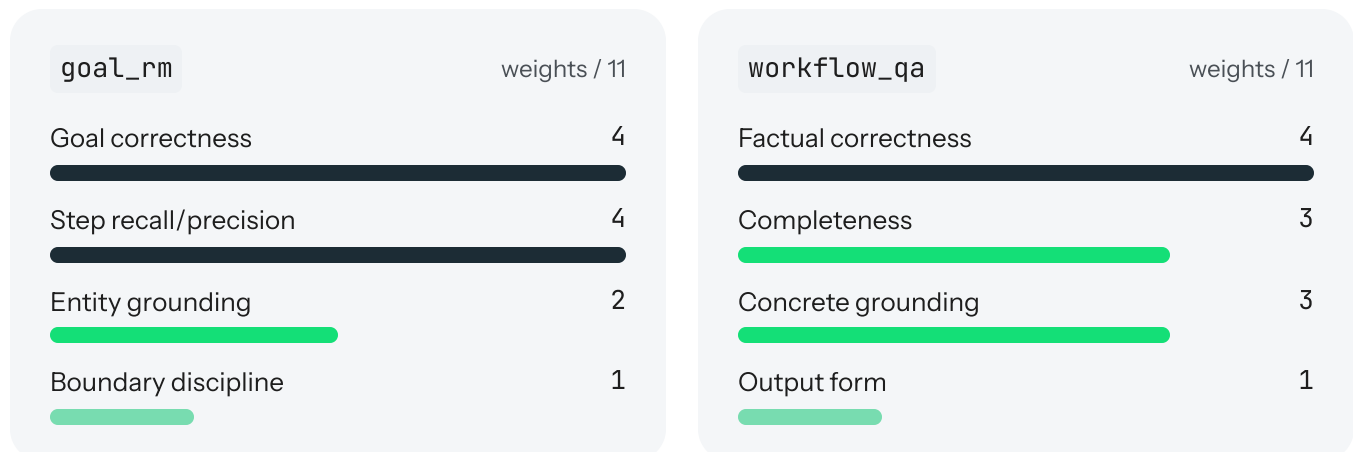


Figure 8. The two per-family rubrics and their criterion weights.

Format penalty

A deterministic, code-based penalty (not judged by the LLM) is subtracted from the judge's score for structural violations (a missing closing reasoning tag, malformed or prose-wrapped JSON, wrong output keys, disallowed headings or bullets in a prose answer, or an answer that refers back to the discarded reasoning), capped at a maximum deduction of 0.40 and floored at zero, so that a single structural defect cannot alone zero out a response's reward.

Failure handling

Any judge failure (an unparseable reply, a missing score field, or an API failure after 5 retries) returns a reward of 0.5 for every completion in the group. Because GRPO's advantage is computed by normalizing within the group, a uniform 0.5 produces zero standard deviation and the group is excluded from the policy update rather than injecting noise from a failed measurement.

04

Metrics and results

4.1 Phase 1: held-out generation accuracy

Beyond the loss curve reported in Table 2, the completed SFT checkpoint was evaluated by generation (not teacher-forced loss) on 4 fully held-out cases comprising 1,139 samples:

90.4%

Goal prediction accuracy

96.9–100%

Output-format validity

71.7%

Step-level QA (semantic correctness)

Metric	Result
Goal prediction accuracy	90.4% correct
Output-format validity	96.9–100%
Step-level QA (semantic correctness)	71.7% correct

Table 3. Phase 1 held-out generation evaluation, 4 cases / 1,139 samples.

4.2 Phase 2: end-to-end validation and reward behavior

At the time of writing, the GRPO stage has been exercised end to end against the real corpus and the live judge (model loading under 4-bit quantization, generation, reasoning/answer parsing, per-family judge routing, advantage computation, backward pass, checkpoint save, and checkpoint resume) with a production run in progress, but the repository does not record a completed full-epoch result. The strongest evidence available is a 10-prompt × 4-generation production-settings trace, reproduced in Table 4.

Step	Family	Judge scores	Std.	Gradient?	Distinct answers
1	workflow_qa	0.97, 0.83, 0.93, 0.88	0.053	yes	4
2	goal_rm	0.80, 0.80, 0.80, 0.80	0	no	1
3	goal_rm	1.00, 1.00, 1.00, 1.00	0	no	1
4	workflow_qa	0.00, 0.00, 0.00, 0.00	0	no	4 (corrupt input)
5	goal_rm	0.99, 1.00, 1.00, 1.00	0.004	yes	2
6	workflow_qa	0.70, 0.81, 0.89, 0.97	0.100	yes	4
7	goal_rm	1.00, 0.92, 0.93, 1.00	0.038	yes	3
8	goal_rm	1.00, 1.00, 1.00, 1.00	0	no	1
9	goal_rm	0.95, 0.95, 0.91, 0.95	0.017	yes	2
10	goal_rm	1.00, 1.00, 0.96, 1.00	0.017	yes	2

Table 4. Production-settings trace, 10 prompts × 4 generations, live judge, from the phase-1 adapter.

Six of the ten steps produced a non-zero gradient. Critically, on every step where the four sampled completions actually differed from one another, the judge assigned them different scores (6 of 6); the zero-gradient steps occurred only where all four completions were byte-identical, meaning there was genuinely nothing for the judge to discriminate. This addresses an earlier concern in the project that the judge itself was insufficiently discriminating: the per-family rubric redesign shows the reward signal is sensitive to real differences in output

quality, and the limiting factor on training signal is completion diversity, not rubric weakness. Step 4's zero-signal, uniformly-zero-score outcome was traced to a corrupted input file, which has since been fixed in the data loader.

4.3 Cost of a full training run

Measured at the production batch size of 6, with judge and GPU time costed separately because only GPU time scales with batch size:



Figure 9. Per-prompt wall-clock cost, split into judge and GPU time. Drawn to scale.

Component	Cost	Notes
Judge latency	94.5 s/prompt	16 real calls, range 32-145 s; batch-size independent
GPU (generation + backward/optimizer)	33.9 s/prompt	scales with batch size
Total	128.5 s/prompt	

Table 5. Per-prompt wall-clock cost breakdown, single-GPU baseline.

THE REWARD SIGNAL DISCRIMINATES

The limiting factor on training signal is completion diversity, not rubric weakness.

