SketchVL: Policy Optimization via Fine-Grained Credit Assignment for Chart Understanding and More

Supplementary Material

1. Visualization Cases of FinePRM

In this section, we provide qualitative visualizations to assess the scoring capability of our FinePRM. As shown in Figure 1, 2 and 3, we intuitively evaluate FinePRM through visualization on complex chart and image scenarios.

Specifically, to produce the scoring heatmaps shown in the following figures, we define a specific intent and apply the action across a 32×32 image grid. FinePRM then scores each action independently. The resulting heatmaps demonstrate that our FinePRM can clearly distinguish between correct and incorrect regions, assigning the highest scores to the areas that most accurately fulfill the intent.

2. Prompts for Data Generation

In this section, we provide the detailed prompts used in our pipeline. To ensure the reproducibility of our method, we present the translated English versions of the prompts used for Data Enrichment, Intent-Action Synthesis (FinePRM), and Cold-Start Data Generation (SketchVL Cold Start).

Prompts for Data Enrichment and Cleaning

1. Chart Attribute Enrichment Prompt
You are a top-tier chart data annotation expert. Your task is to supplement detailed structured information for each annotation based on the given chart image and OCR-recognized text labels.
Input Format: Normalized bounding boxes (0-1000).
Task: For each JSON object in the "Pending Annotations":
- `legend_label`: If the text relates to a legend, fill in the full legend text; otherwise, return "".
- `color`: Describe the color of the text or associated element (e.g., "Red", "Dark Blue").
- `describe`: Detailed description of the role and context (e.g., "X-axis label", "Value at the top of a bar", "Pie sector label").

Output: A strictly valid JSON array.

---

2. General Image Object Verification (Validity)
Please refer to the two images provided (original and cropped crop with red box). Determine whether the content within the red box constitutes a meaningful, relatively complete object or a recognizable part of an object (e.g., a wheel is meaningful; a patch of solid color or sky is not). Answer only "Yes" or "No".

---

3. General Image Fine-grained Labeling
Your task is to generate a structured description for the object inside the red box to uniquely identify it in the original image. Return JSON format:
- `label`: Precise name of the object (e.g., "White porcelain plate").
- `relation`: Detailed description of the spatial relationship with surrounding objects to eliminate ambiguity (e.g., "In the middle of the table, below the laptop").

---

4. Duplicate Resolution Prompt
Task: Analyze two highly overlapping annotations to determine if they point to the same object and decide which description is better.
Steps: Check the cropped images and the original image. If they are the same entity, compare "Label" and "Relation" to select the one that is more accurate and informative. **Output:** Only the `Label` name of the retained annotation.

Prompts for Intent-Action Pair Synthesis

System Prompt:
Your task is to generate training data for a Reward Model. Based on the given image/chart configuration and annotation information, generate a JSON list containing N independent "Action Pairs".
Each item must contain:
- `explanation`: A natural language description of the drawing intent. It should explain *what* to mark and *where* it is, without revealing specific coordinates or the fact that you are looking at metadata. (e.g., "Circle the window with grilles located in the center right of the image").
- `action`: A single drawing instruction JSON object ('point', 'line', 'circle', 'rectangle', or 'text') derived accurately from the provided annotations.

Constraints:
- **Diversity:** The actions must strictly follow a specified sequence (e.g., point - line - circle) and cover different objects.
- **Accuracy:** Do not guess coordinates. Use the provided annotation data strictly.
- **Chart Specifics:** For charts, X-axis labels should be marked at the top edge center; Y-axis labels at the right edge center. Do not mark legends if they are not in the annotation.
- **Image Specifics:** For general images, ensure the description uniquely locates the object using spatial relationships provided in the context.

Output Format: A strict JSON list of objects containing 'explanation' and 'action' keys.

Prompts for Cold-Start Data Generation

1. Visual Question Generation
Please generate a visual question regarding the chart/image content based on the provided annotations. The question should:
- Focus on the visual content (e.g., values, trends, object relationships, counting).
- Be concise and answerable based strictly on the image.
- (For Charts) Focus on topics like: Extremes & Ranking, Comparisons, Trends, Statistics.
- (For Images) Focus on topics like: Spatial relationships, Attributes, Actions, Context.

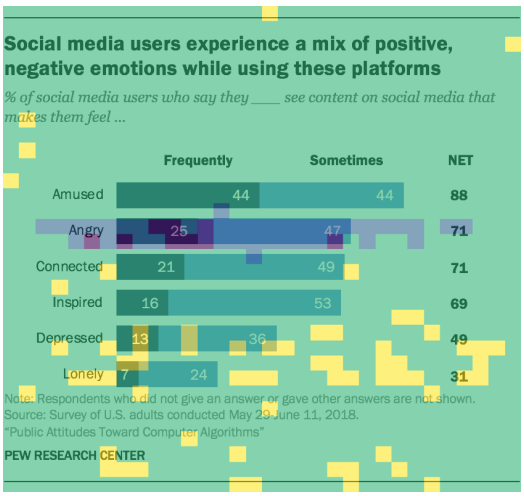
Output: Only the question string.

---

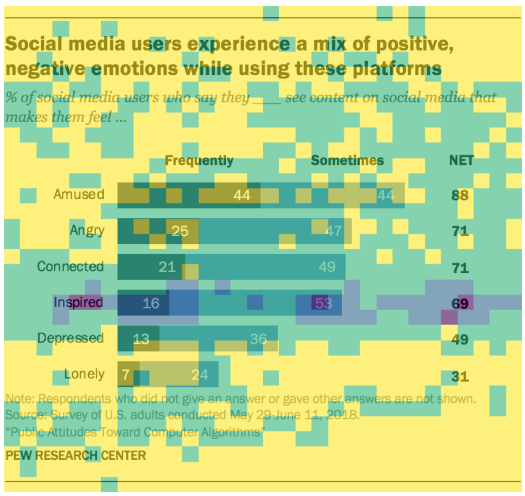
2. Sketch-CoT Reasoning Generation
Your task is to answer the user's question by generating a multi-step reasoning process that includes visual sketching actions.
Output Format: A JSON list of lists, where each sub-list represents a reasoning turn.
- `explanation`: Summarize the previous step or plan the current drawing. Do not state the final answer until the last step.
- `action`: A dictionary defining a drawing operation ('point', 'line', 'circle', 'rectangle', 'text') with normalized coordinates (0-1000).

Key Instructions:
- **Draw then Conclude:** You should first perform a drawing action (visual grounding) and then discuss the finding in the next turn.
- **No Hallucination:** Use strictly existing coordinates from the provided metadata.
- **Chain of Thought:** The reasoning should be linear. The first turn plans the path; intermediate turns verify data points via drawing; the final turn (with empty action) provides the direct answer to the question.
- **Conciseness:** Solve the problem in typically 3-4 turns.

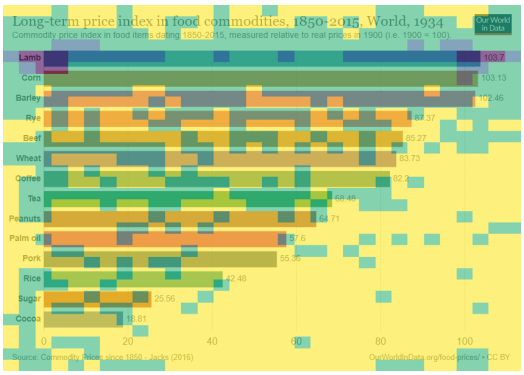
Input Provided: Chart HTML/Image Annotations and the User Query.



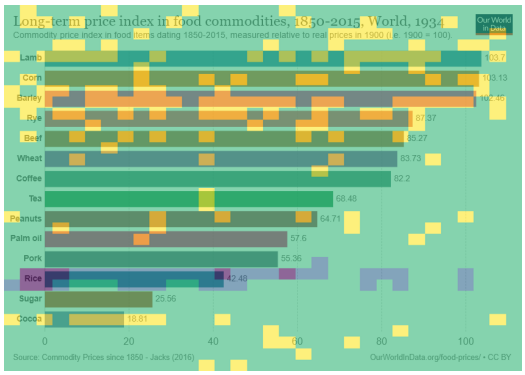
Mark the Frequently Angry.



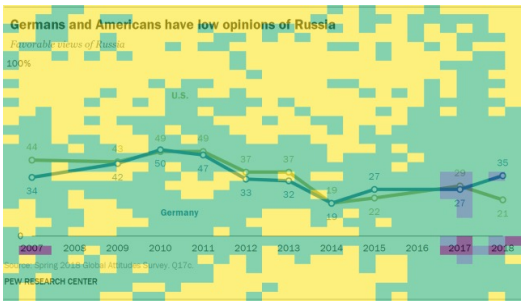
Mark the Inspired in the NET category.



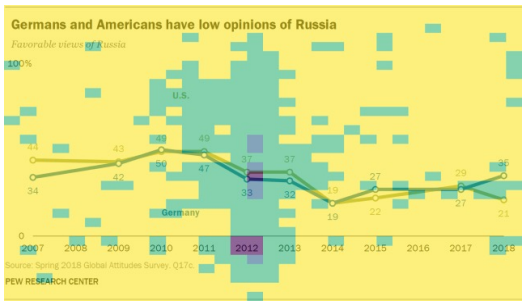
Highlight the Lamb label in the vertical axis.



Highlight the Rice label in the vertical axis.

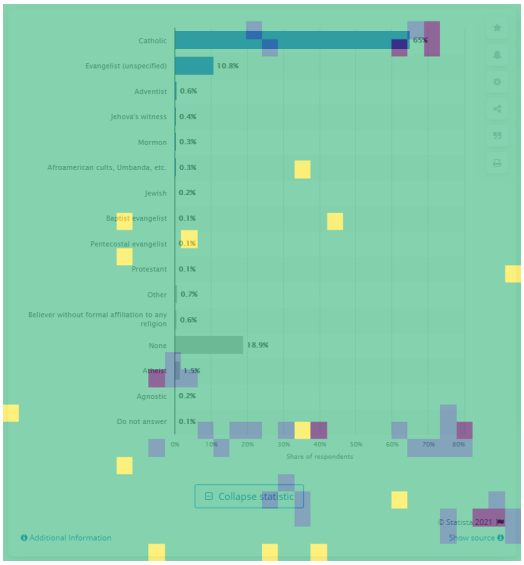


Highlight the Germany value label in the year 2017.

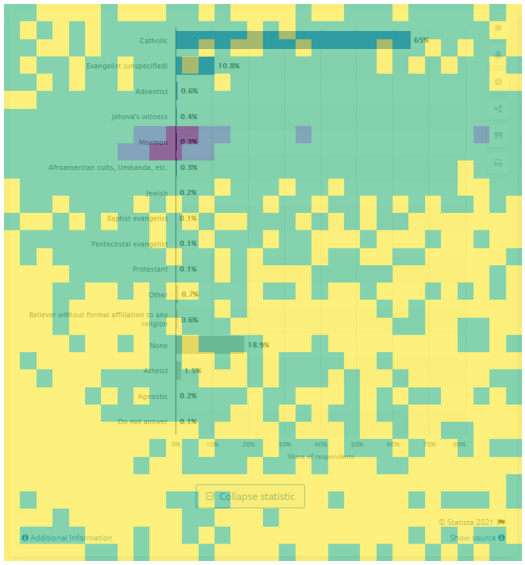


Highlight the US value label in the year 2012.

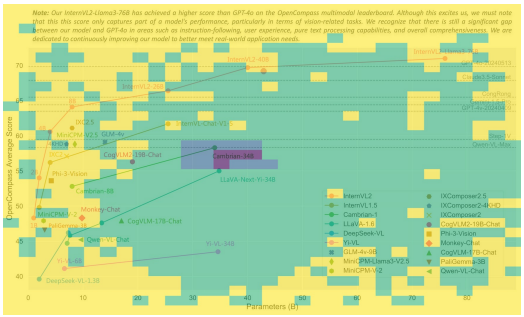
Figure 1. **Visualization of FinePRM scoring heatmaps (Case 1).** We visualize the spatial credit assignment by FinePRM. The heatmap represents the score distribution across the image for a given intent. High-scoring regions indicate that FinePRM correctly identifies the areas most relevant to the instruction.



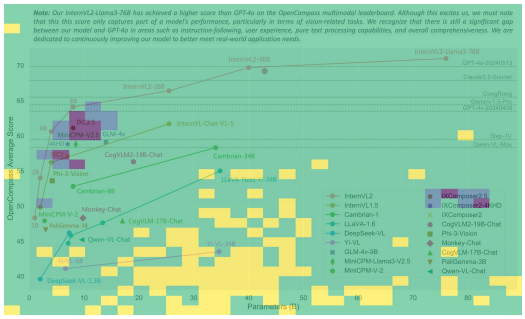
Highlight the 75 number on the x axis in the image.



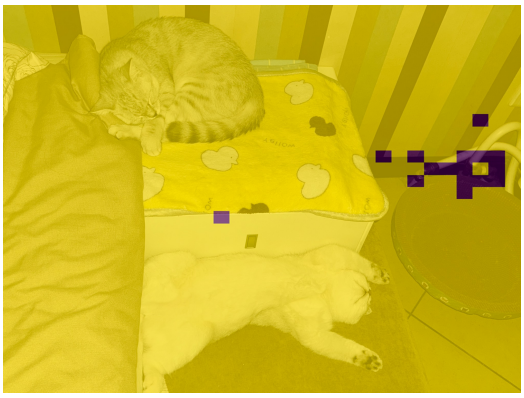
Mark the mormon on the y axis in the image.



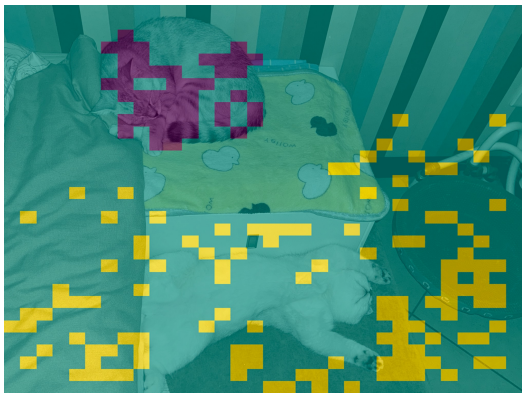
Mark the position of cambrain 34B.



Mark the legend label of IXC25 in Legend area.



Circle the heating pipes in the image.



Highlight the gray cat in the image.

Figure 2. **Visualization of FinePRM scoring heatmaps (Case 2).**

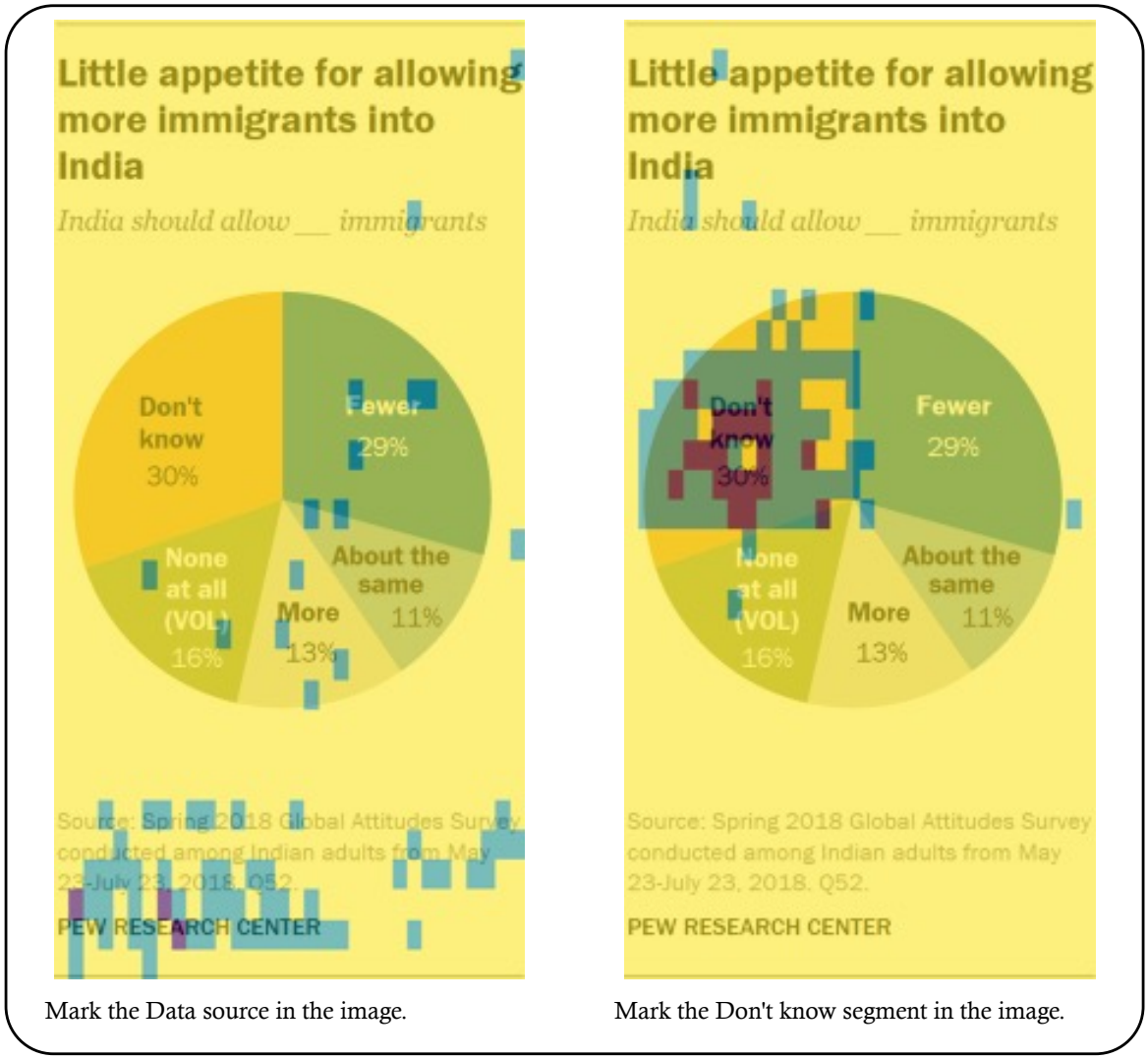


Figure 3. **Visualization of FinePRM scoring heatmaps (Case 3).** Further visualization of FinePRM's scoring process on complex charts and general images. The legend indicates the score quality, where darker colors (e.g., Purple for 'Excellent') represent higher scores, and lighter colors (e.g., Yellow for 'Unacceptable') represent lower scores. As shown, the high-scoring regions (darker patches) align precisely with the semantic intent described below each image, confirming that FinePRM provides accurate, fine-grained feedback.

3. Implementation of Fine-Grained Credit Assignment

In this section, we provide the implementation logic for our proposed Fine-Grained Policy Optimization (FinePO). Due to the complexity of the logic, we present the code in a single-column format.

3.1. Variable Definitions

To facilitate the understanding of the provided Python implementation, we map the key variables in the code to the mathematical notations defined in the FinePO methodology.

- `inputs`: The batch data structure containing the generated trajectories $\{y_1, \ldots, y_k\}$, their corresponding FinePRM process scores p_j, and associated metadata.
- `prior_action_distribution`: Represents the fixed prior distribution $Q(a)$ derived from the training set, serving as the anchor for the KL divergence constraint.
- `action_history`: A sliding window buffer used to estimate the current policy's action distribution $P_k(a)$, ensuring a stable calculation for the dynamic KL penalty.
- `kl_lambda`: Corresponds to the coefficient λ_{KL}. It controls the strength of the penalty applied when the current action distribution deviates from the prior.
- `reward_offsets`: Stores the computed clipped KL offset values $\mathcal{O}_{\text{clipped}}(a_j)$. These offsets are added to the raw FinePRM scores to regularize the action types.
- `scalar_adv`: Represents the coarse, cross-trajectory advantage $A(y_i)$. This scalar serves as the base value and the total "budget" to be redistributed among the individual steps.
- `s_avg`: The length-weighted average score $\bar{p}$ of all steps within a single response. It acts as the intra-response baseline to determine the relative quality Δ_j of each step.
- `k`: The dynamic scaling factor k. It scales the fine-grained adjustment based on the magnitude of the coarse advantage $|A(y_i)|$ and the maximum score deviation.
- `alpha`: Corresponds to the hyperparameter α. It governs the intensity of the redistribution, determining how much the fine-grained scores influence the final token-level advantage.
- `token_spans`: A utility mapping used to determine the token length L_j and specific indices for each reasoning step s_j, enabling the precise application of the final step-level advantage $A(s_j)$ to the token sequence.

Algorithm 1: Fine-Grained Advantage Redistribution

```
def _prepare_batch_inputs(self, inputs: DataType) -> List[DataType]:
    """
    Prepare batch inputs with KL penalty and Token-Level Credit Assignment.
    This function runs on each worker process.
    """
    # --- Part 1: Global Distribution Alignment (KL Penalty) ---
    credit_key = f'credit_details_{self.reward_func_names[0]}'

    # 1. Gather local actions from the current batch
    local_actions = []
    for data in inputs:
        if credit_key in data:
            # Extract action types (e.g., 'point', 'line') from step strings
            for step_str in data[credit_key].keys():
                action = self._parse_action_type(step_str)
                if action in self.prior_action_distribution:
                    local_actions.append(action)

    # 2. Synchronize action counts across all distributed workers
    # gathered_actions_flat_list contains actions from ALL GPUs
    global_actions = gather_object(local_actions)
    reward_offsets = {}

    # 3. Calculate KL-based Reward Offsets (Executed on Main Process)
    if self.accelerator.is_main_process:
        # Update sliding window history with current global counts
        global_counts = collections.Counter(global_actions)
        self.action_history.append(global_counts)

        # Aggregate counts from the sliding window
        total_counts = collections.defaultdict(int)
        for counts in self.action_history:
            for act, c in counts.items():
```

```
                total_counts[act] += c
        total_n = sum(total_counts.values())

        if total_n > 0:
            # Calculate current distribution P_curr
            p_current = {a: c / total_n for a, c in total_counts.items()}

            # Calculate Offset: -lambda * log(P_curr / P_prior)
            for action, p_prior in self.prior_action_distribution.items():
                p_curr = p_current.get(action, 0.0) + self.epsilon
                # KL Divergence direction: Penalty if P_curr > P_prior
                offset = -self.kl_lambda * math.log(p_curr / (p_prior + self.epsilon))
                # Clip offset to prevent training instability
                reward_offsets[action] = clip(offset, -self.clip_val, self.clip_val)

    # 4. Broadcast calculated offsets to all workers
    broadcast_object_list([reward_offsets], from_process=0)

    # 5. Apply offsets to local data (Update Reward Scores)
    for data in inputs:
        if credit_key in data:
            for step_str, score in data[credit_key].items():
                action = self._parse_action_type(step_str)
                # Add the global penalty/bonus to the step score
                data[credit_key][step_str] = score + reward_offsets.get(action, 0.0)

    # --- Part 2: Token-Level Credit Assignment ---
    # Split data into mini-batches for memory efficiency
    batches = self.split_by_mini_batches(inputs)
    processed_batches = []

    for batch in batches:
        # (Standard encoding and masking logic omitted for brevity...)
        # ...

        # Initialize a tensor to hold per-token advantages
        # Shape: [Batch_Size, Sequence_Length]
        advantages_tensor = torch.zeros_like(labels, dtype=torch.float32)

        for j, sample in enumerate(batch):
            scalar_adv = sample['advantages'].item() # Original GAE advantage
            credit_details = sample.get(credit_key, {})

            # Identify valid completion tokens
            mask = (labels[j] != -100)

            # Default: Assign the scalar advantage to ALL tokens in the response
            per_token_adv = torch.full((mask.sum(),), scalar_adv)

            # If FinePRM scores exist, perform redistribution
            if credit_details:
                # 1. Map logical steps to physical token indices
                # Returns: {'step_1': (start_idx, end_idx), ...}
                token_spans = self._map_sub_steps_to_tokens(sample, credit_details)

                # 2. Calculate Length-Weighted Average Score (S_avg)
                w_score_sum = 0
                total_len = 0
                for step_str, score in credit_details.items():
                    if step_str in token_spans:
                        span_len = token_spans[step_str].len
                        w_score_sum += score * span_len
                        total_len += span_len

                s_avg = w_score_sum / total_len if total_len > 0 else 0

                # 3. Determine Dynamic Scaling Factor (k)
                # We calculate deviations from the average
                offsets = [score - s_avg for score in credit_details.values()]
                max_deviation = max(offsets) if offsets else 0

                if max_deviation > 0:
                    # k scales the deviation so that the max deviation roughly
                    # corresponds to the magnitude of the scalar advantage.
                    k = abs(scalar_adv) / (max_deviation + 1e-6)
                else:
                    k = 0
```

```
                # 4. Redistribute Advantage
                for step_str, score in credit_details.items():
                    if step_str not in token_spans: continue

                    start, end = token_spans[step_str]
                    offset = score - s_avg

                    # Core Redistribution Formula:
                    # A_new = A_original + alpha * k * (Score_step - Score_avg)
                    adjustment = self.alpha * k * offset
                    mod_adv = scalar_adv + adjustment

                    # 5. Conservative Clipping (Advantage Conservation)
                    # Ensure the sign doesn't flip excessively and stays within bounds
                    if scalar_adv < 0: # Negative feedback case
                        # Allow worse steps to be more negative, but limit improvement
                        lower_bound = 2.0 * scalar_adv
                        mod_adv = max(lower_bound, min(mod_adv, 0.0))
                    else: # Positive feedback case
                        # Allow better steps to be more positive, but limit punishment
                        upper_bound = 2.0 * scalar_adv
                        mod_adv = max(0.0, min(mod_adv, upper_bound))

                    # Assign modified advantage to the specific token span
                    per_token_adv[start:end] = mod_adv

            # Fill the tensor
            advantages_tensor[j, mask] = per_token_adv

        # Update the batch data structure
        encoded_batch['advantages'] = advantages_tensor
        processed_batches.append(encoded_batch)

    return processed_batches
```

4. Evaluation Prompts

To ensure a fair and rigorous comparison, we employ a model-based judge (DeepSeek-R1-Distill-Qwen-14B) to evaluate the correctness of SketchVL's responses. Depending on the dataset, we apply specific evaluation criteria to handle the nuances of chart reasoning. Below are the translated system prompts used for each benchmark.

Prompt for ChartQA-Pro Evaluation

System Instruction:
Please judge the correctness of the following answer. Briefly analyze or calculate first, and finally append a `[true]` tag if correct, or `[false]` if incorrect. Focus solely on whether the final conclusion matches the ground truth label. Do not judge the reasoning process.
Evaluation Principles (Relaxed Correctness Metric):
1. **MCQ & Fact Checking:** Use **Exact Match**. The predicted option (A/B/C/D) or Boolean (True/False) must strictly match the label.
2. **Year Values:** Use **Exact Match**. Years (e.g., 2010 vs 2009) must be precise; visual approximation errors are not allowed for years.
3. **Other Numeric Values:** Allow a **5% relative error**. Calculate error $E = |Pred - GT|/|GT|$. If $E \leq 0.05$, mark as correct. Handle unit conversions if necessary.
4. **Textual Answers:** Assess **Semantic Similarity**. Exact match is not required; ignore minor differences like pluralization ('Female' vs 'Females') or casing.
5. **List Answers:** Split the list and evaluate each element individually using rules 1-4. All elements must be correct.

Prompt for PlotQA Evaluation

System Instruction:
Analyze the correctness of the response. Output `[true]` or `[false]` at the end. Do not provide an ambiguous "neither" option.
Evaluation Principles:
- **Non-Numeric Answers:** Analyze semantic understanding. Colors are correct if they are semantically similar.
- **Numeric Answers:** You must perform a calculation to verify the error. Compute accuracy $acc = |Label - Answer|/|Label|$.
- **Threshold:** If $acc \leq 0.05$, evaluate as True; otherwise False.
- **Units:** If units do not align, convert them based on context before calculating the error.
- **Formulas:** If the assistant outputs an uncomputed formula, evaluate its final result against the label with the 5% tolerance.

Prompt for EvoChart Evaluation

System Instruction:
Judge the correctness based on the provided content. Output `[true]` or `[false]`.
Evaluation Principles:
- **Condition `is_clear`:** If `is_clear=true` is specified in metadata, require strictly precise numerical correspondence.
- **Condition `not is_clear`:** If `is_clear=false`, allow a 5% tolerance for all numeric labels.
- **Calculation:** For numeric labels (e.g., 27 vs 28), if the relative error is within $\pm 5\%$, count it as correct. Example: $(22 - 21)/21 < 0.05$ is acceptable.
- **API Errors:** If an API error occurs in the model output, look for a correct answer generated prior to the error.

Prompt for General, Math and Other Evaluation

System Instruction:
Judge the correctness of this subjective or general question. Output `[true]` or `[false]`.
Evaluation Principles:
- **Numeric Answers:** Allow a 5% tolerance excluding units.
- **Years:** Be lenient. Treat years as numeric values with 5% tolerance (e.g., 2018 vs 2022 is within 5% relative error).
- **Non-Numeric:** Colors and text need only be semantically close.
- **Objective:** Focus on the final answer, ignoring intermediate reasoning steps unless they contradict the correct final result.

5. Training Dynamics Analysis

We first summarize the key hyperparameter settings used in the FinePO reinforcement learning phase in Table 1.

Table 1. Key hyperparameters for the FinePO RL phase.

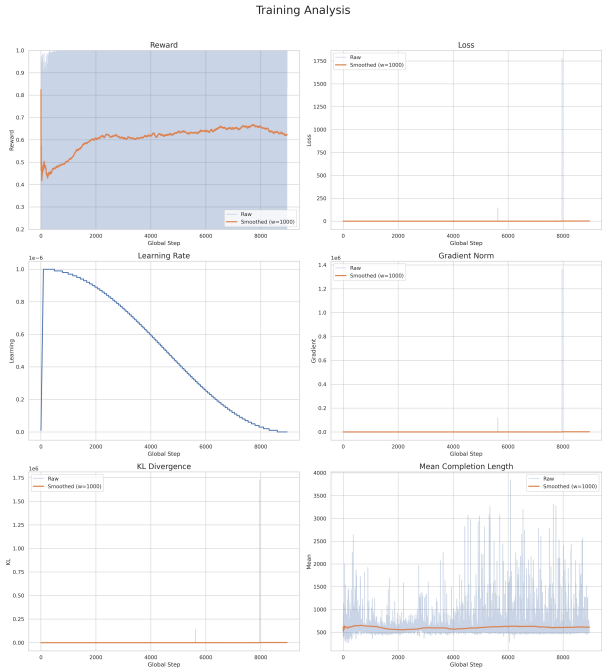
Symbol	Meaning	Value
k	Number of generations per prompt	24
λ_{KL}	KL penalty coefficient	0.1
γ	Clipping threshold for KL offset	0.5
α	Credit adjustment intensity	0.2
β	Clipping range factor for advantage	2.0
-	Learning Rate	1e-6
-	Temperature	1.0
-	GRPO KL beta	0.01

To further validate the effectiveness of our method, we visualize the training dynamics of the SketchVL-3B model and its ablation variants. Figure 4 presents the curves for Reward, Loss, Learning Rate, Gradient Norm, KL Divergence, and Mean Completion Length over the training steps.

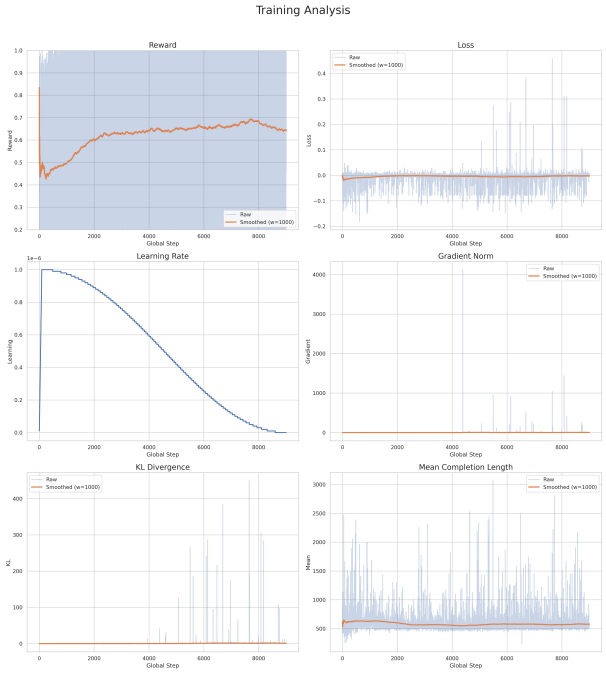
Stability Analysis. As observed in the **Gradient Norm** and **KL Divergence** curves, the full SketchVL-3B model (Figure 4a) exhibits significantly higher training stability compared to the baselines.

- The **w/o FinePO (naive GRPO)** model (Figure 4b), which relies on coarse trajectory-level rewards (Naive GRPO), shows drastic spikes in Gradient Norm and unstable KL divergence. This confirms that uniformly broadcasting rewards to all tokens introduces substantial noise, destabilizing the policy update.
- Similarly, the **w/o Sketch (zero GRPO)** model (Figure 4c) lacks the explicit grounding of reasoning steps, leading to a harder optimization landscape. Notably, although it achieves a relatively ideal training reward, it performs poorly on test benchmarks.

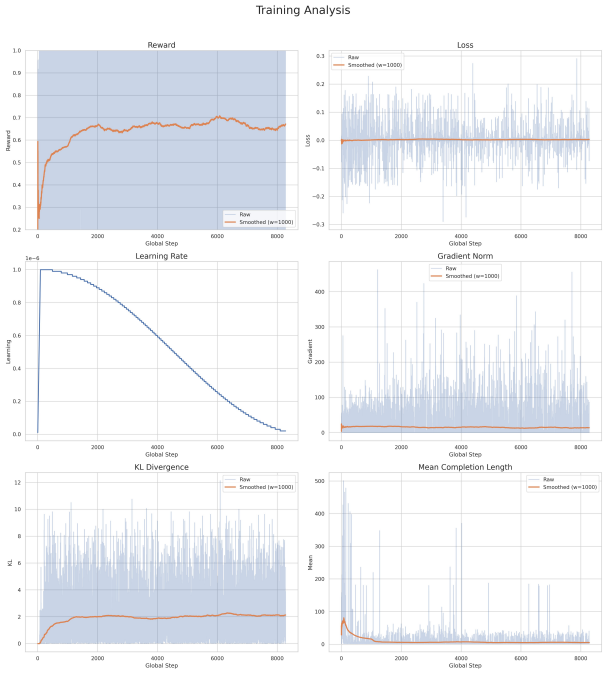
In contrast, our FinePO mechanism (Figure 4a) effectively smooths the learning process by assigning precise credit to individual steps, resulting in stable gradient updates and controlled policy deviation.



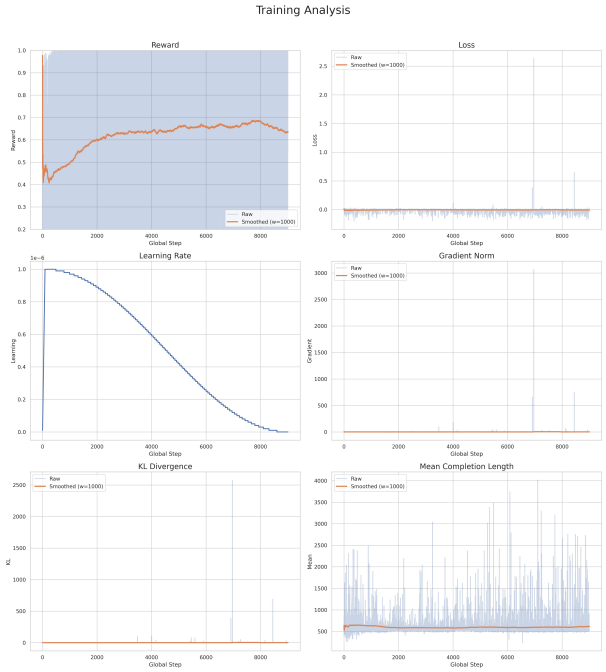
(a) **Full SketchVL-3B (Ours).** Trained with FinePO and FinePRM.



(b) **w/o FinePO (Naive GRPO).** Uses only coarse trajectory-level rewards.



(c) **w/o Sketch (Zero GRPO).** No interactive reasoning actions.



(d) **w/o FinePRM (Random Scores).** Credit is assigned randomly.

Figure 4. **Comparison of Training Dynamics.** We visualize the training metrics for the full SketchVL-3B model and three ablation baselines. The curves include Reward (mean training reward), Loss, Learning Rate, Gradient Norm, KL Divergence (between the current policy and the reference model), and Mean Completion Length.

Table 2. Additional ablation study and baseline comparison.

Method	ChartQA	ChartBench	PlotQA
ChartGemma	70.04	23.87	25.76
CoT + GRPO	72.56	57.10	43.20
PRM-Agg	75.92	59.40	46.50
SketchVL w/ 3B PRM	74.00	56.33	41.50
SketchVL	77.20	59.96	48.32

Table 3. Inference throughput analysis.

Model	Avg. Input Tok/s	Avg. Output Tok/s
Qwen2.5-VL-7B	987	1,455
SketchVL-7B	2,217	619

6. Additional Analyses and Implementation Details

This section supplements the training dynamics analysis with additional baselines, inference-time throughput results, and implementation details.

6.1. Additional Baselines and Ablation Study

To provide a stronger empirical comparison, we include three additional baselines: (1) a text-only CoT cold start followed by GRPO optimization, (2) an aggregate-reward PRM baseline (PRM-Agg), and (3) a chart-specialized expert model, ChartGemma. The results are summarized in Table 2. SketchVL consistently outperforms these baselines across ChartQA, ChartBench, and PlotQA, suggesting that step-level credit assignment with FinePRM is more effective than relying only on final-answer supervision or aggregate process rewards.

6.2. Inference Throughput

We further analyze the inference-time cost of SketchVL. Compared with standard CoT reasoning on Qwen2.5-VL-7B, SketchVL shifts more computation into the prefill stage while reducing decoding throughput. In practice, this trade-off is acceptable in our setting because the interactive sketch tokens compress part of the reasoning process into structured visual actions. All models in this comparison are deployed on 4$\times$A800 40GB GPUs.

6.3. Implementation and Evaluation Details

We provide several additional implementation details for completeness.

Role of FinePRM. Our goal is to use FinePRM to constrain and improve the intermediate reasoning process, rather than to directly distill the process reward model into the policy.

Credit assignment and clipping. For correct steps that appear in suboptimal trajectories, we assign lower penalties instead of treating them as entirely negative. Under policy gradient training, this functions as a relative positive signal. In practice, clipping is rarely triggered in our experiments.

Evaluation criteria. The answer-matching criteria follow the official specifications of each benchmark dataset, rather than using a fixed tolerance threshold such as 5% across all tasks.

Training setup. The 3B and 7B models currently share the same training data and overall optimization setup, while most design and tuning effort is focused on the 3B setting. This partially explains why the relative gains on the stronger 7B backbone are smaller under the same budget.

6.4. Discussion of Process Quality and Final Accuracy

In multi-step visual reasoning, the final outcome reward is positively correlated with process quality but is highly nonlinear with respect to it. Final correctness often depends disproportionately on the last few critical steps and is therefore less sensitive to how many intermediate steps are correct. Consistent with this observation, we find that the process-level PRM score typically improves before the accuracy gap widens. This suggests that improvements in intermediate reasoning emerge earlier, while gains in end-task accuracy appear more gradually under limited compute.

Reward Analysis. The final converged reward of the full SketchVL model is slightly lower than that of some baselines (e.g., w/o FinePO), which is expected. FinePO introduces additional constraints, particularly **KL Action Regularization** and **FinePRM alignment**, to prevent reward hacking, where the model exploits shortcuts to obtain high scores without proper reasoning. Although these constraints may slightly reduce the absolute reward value, they encourage a more rigorous and generalizable reasoning process, ultimately leading to better benchmark performance.