LLMs are Lazy and Unreliable: On the Impact of Instance Order in Multi-Instance Processing

Anonymous ACL submission

Abstract

Users increasingly rely on large language models (LLMs) to analyse collections of independent inputs, such as product reviews or social media posts, and return a single aggregated answer. This setting requires multi-instance processing (MIP), where a model must process all instances and combine their evidence into one prediction. Since MIP inputs are unordered sets, the correct answer should be invariant to any permutation of the instances. We use this property as a necessary reliability criterion and design a controlled probe to test whether current LLMs satisfy this permutation invariance prerequisite. Evaluating fourteen LLMs on a simple majority-label task across five datasets and seven order patterns, we find that models are far from permutation invariant. Accuracy differs by up to 90% between the best and worst order patterns, with majorities placed at the start being consistently easier to identify than those placed in the middle or at the end. By examining the reasoning traces of six models, we observe a recurring behaviour that we refer to as "laziness". Models often process only part of the input, frequently around a quarter or fewer instances, despite being instructed to consider all instances. These results show that current LLMs are not yet reliable aggregators for MIP, as trivial order changes can systematically steer their predictions in a specific direction.

1 Introduction

Large language models (LLMs) are increasingly used to assist users in analysing large collections of independent inputs, such as product reviews, social media posts, user records, or news snippets (Sun et al., 2025; Tang et al., 2025). This scenario requires multi-instance processing (MIP): the model must process each instance and aggregate the evidence into one prediction (Bertsch et al., 2025; Chen et al., 2026). MIP differs in an important way from long-document and retrieval-augmented settings. In those settings, the passages follow a fixed order and the task typically requires extracting only a small, relevant fraction of the input while ignoring the rest (Hsieh et al., 2024a; Lewis et al., 2020; Liu et al., 2025). In MIP, by contrast, the input constitutes an unordered set where presentation sequence carries no information. Because every instance contributes to the final aggregation, omitting even a small fraction can alter the predicted answer (Chen et al., 2026). Given that these aggregated predictions frequently drive downstream decisions, ensuring their reliability is of real practical importance (Zhu et al., 2026).

Fully quantifying LLM reliability in MIP is challenging, as it requires defining a general and objective threshold for what constitutes sufficient reliability (Liu et al., 2023). However, one critical facet of reliability circumvents this need. Because MIP operates over an unordered set of instances, the target output is strictly permutation invariant. Any deviation in a model's prediction when the input instances are reordered constitutes a provable error. We therefore treat permutation invariance as a rigorous reliability criterion; while it is a necessary rather than sufficient condition for overall reliability, its violation alone definitively exposes an LLM as unreliable in the MIP setting.

Despite its importance, limited prior work evaluates the reliability of LLMs in MIP, specifically regarding permutation invariance. Within MIP itself, existing work primarily explores batch prompting for efficiency at modest scales (Cheng et al., 2023; Lin et al., 2024), or treats multi-instance inputs as one subtask inside a broader long-context benchmarks (Qiu et al., 2024; Shaham et al., 2023). The study closest to ours measures accuracy degradation as instance count and context length increase, but crucially, it does not investigate instance order (Bertsch et al., 2025; Chen et al., 2026). A separate line of research examines positional biases (Gao et al., 2024; Tian et al., 2025; Veseli et al.,

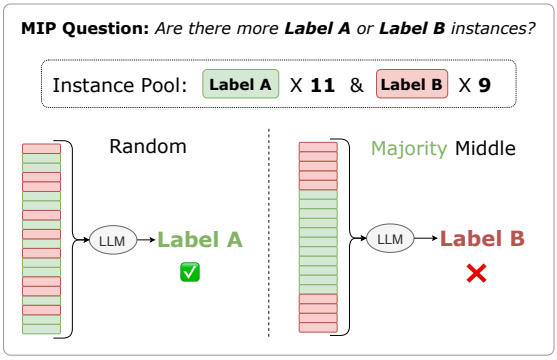

Figure 1: A toy example of 20 instances with two labels. Each instance corresponds to a text input (e.g. a movie review) and labels are binary (e.g. positive or negative). On the left of the figure, instances are randomly distributed, while on the right, the instances with the majority label (i.e. Label A) are placed in the middle.

2025), such as the lost-in-the-middle effect in retrieval (Liu et al., 2024) and sensitivity to the order of in-context examples or answer options (Cobbina and Zhou, 2025; Pezeshkpour and Hruschka, 2024; Zheng et al., 2024). These tasks, however, involve finding or selecting a single relevant item. They do not require aggregating over a set where the correct answer is provably independent of presentation sequence. Consequently, MIP reliability remains largely untested against a definitive, threshold-free criterion like permutation invariance.

To test whether LLMs are permutation invariant in MIP, we build a controlled probe to explore whether they are robust to instance order. We instantiate a majority-label task, where each instance carries a binary label that the model must infer, after which it needs to answer which label is in the majority, a target that is order-independent by construction. Figure 1 illustrates an example of our setting. We find that LLMs are broadly not robust to instance order. Across models, accuracy swings by up to around 90% between the best and worst order patterns, and the effect is systematic and directional. A majority placed at the start is far easier to identify than one placed later in the middle. This failure is accompanied by a consistent behaviour we call "laziness". The LLM reasons over only part of the input, often less than a quarter of all instances, and favours instances from the beginning.

2 Related Work

Multi-Instance Processing. Several lines of work study LLM prompts that contain multiple independent inputs, problems, or evidence units. One line of research frames such inputs primarily through the lens of batch processing and multi-task inference, which investigate whether multiple items can be processed jointly to reduce repeated instructions, API calls, or inference cost while largely preserving task performance (Cheng et al., 2023; Lin et al., 2024; Kolawole et al., 2026; Son et al., 2024). Another line treats aggregation as one task inside broader long-context benchmarks, including tasks requiring information fusion, aggregation, or deep reasoning over long inputs (Bai et al., 2025; Hsieh et al., 2024a; Qiu et al., 2024; Shaham et al., 2023). Compared to them, our setting instead isolates a clean MIP regime in which multiple semantically independent instances jointly determine one answer, enabling a controlled study of order sensitivity and reliability.

Position Bias and Lost-in-the-Middle Effects. A substantial body of work shows that LLMs are highly sensitive to where relevant information appears in the input. Liu et al. (2024) established the "lost-in-the-middle" effect, showing that long-context performance often follows a U-shaped curve over position. Subsequent work broadened this picture by studying multi-evidence and position-aware settings, including the effects of absolute position, relative spacing between relevant pieces, and multi-evidence retrieval (Hsieh et al., 2024b; Song et al., 2025; Tian et al., 2025; Veseli et al., 2025). Related order effects also arise outside retrieval: few-shot in-context learning is sensitive to demonstration order, and multiple-choice reasoning is brittle to option permutations and selection bias (Cobbina and Zhou, 2025; Pezeshkpour and Hruschka, 2024; Zheng et al., 2024). Our work is closest in spirit in that it also tests whether semantically irrelevant reordering changes model behaviour. However, unlike prior work that usually focuses on retrieving one answer-bearing span, a small set of relevant passages, or reordered answer options, we study long contexts in which all instances are relevant to the final answer and the task is globally aggregative rather than retrieval-based.

3 Are LLMs Robust to Instance Order?

To study if LLMs can perform MIP robustly under instance permutations, we use five heterogeneous classification tasks. The common goal in our setting is to find the majority label across a fixed instance pool. Within each pool, we present the in-

Name	Target	Label 1 : Label 0
Math	Answer	odd_number : even_number
NER	Entity	PER : LOC
Occurrence	"the"	exactly_once : more_than_once
Sentiment	Polarity	positive : negative
WSD	"java"	programming lang : place

Table 1: Overview of our binary-labelled tasks, where the MIP objective is to identify the majority label. For example, the *Math* task requires the LLM to classify the answer as either "odd_number" or "even_number".

stances in various order patterns, expecting similar accuracy across all of them if the LLMs are truly permutation invariant.

3.1 Experimental Setting

3.1.1 Individual Tasks

We design five diverse classification tasks from existing datasets: *Math* (Cobbe et al., 2021), *NER* (named-entity recognition) (Rahimi et al., 2019),[1] *Occurrence* (Barbieri et al., 2020), *Sentiment* (Socher et al., 2013) and *WSD* (word sense disambiguation) (Loureiro et al., 2021). We treat each instance within a given task as an independent binary classification problem. In the MIP setting, we ask the model to identify the majority label across all instances, allowing it to output "tie" if both labels appear with equal frequency. Table 1 shows the information about each task (more details about each specific task can be found in Appendix A).

3.1.2 Models and Prompting

We evaluate fourteen LLMs[2], including six pairs of models from the same company: (*DeepSeek-V4-Flash*, *DeepSeek-V4-Pro*), (*Gemini 3.1 Flash-Lite*, *Gemini 3.1 Pro*),[3] (*Gemma 4 26B A4B*, *Gemma 4 31B*), (*gpt-oss-120b*, *GPT-5.4 nano*), (*Llama 4 Scout*, *Llama 4 Maverick*), and (*Qwen3.6-35B-A3B*, *Qwen3.6-Plus*); and two other models: *Grok 4.1 Fast* and *MiniMax M2.7*.

For prompting, we use a temperature of 0 and a maximum output length of 20K tokens for consistency across all models. To allow limited tolerance for invalid model responses, we permit up to three attempts when a model fails to produce a valid output. The full set of prompting templates is provided in Appendix B. Unless otherwise specified, we use the general prompts for each task.

3.1.3 Instance Filtering

Given that our experimental design relies on pre-defined labels, we expect these labels to be consistent with LLMs' judgements. We use two LLMs to filter the dataset: *Qwen3.6-35B-A3B* and *Gemini 3.1 Flash-Lite*. We prompt both models to classify each instance into its respective label (i.e. usually Label 1 or Label 0). However, because the ground-truth targets for *Math* and *Occurrence* are not inherently binary, we instead extract their intermediate reasoning. Rather than directly querying for the binary outcome (e.g. odd/even or single/multiple occurrences), we ask the models to generate the final numerical answer or specific occurrence count.

We evaluate 2,500 instances per task, split evenly across the two labels. The two filtering LLMs achieve over 90% agreement with the pre-defined labels across all tasks, suggesting that the dataset is well-aligned with model judgements. To further isolate aggregation capabilities, we retain only those instances classified correctly by both LLMs. This filtering procedure isolates the multi-instance processing variable; consequently, incorrect MIP predictions are more likely to reflect aggregation failures rather than individual instance difficulty. Additional filtering details are available in Appendix C.1.

3.1.4 Multi-Instance Input Construction

Each input instance set $\mathcal{X}$ is constructed according to five factors, denoted by $(n, p, y_{\text{maj}}, s, \pi)$. Here, n is the instance count, p is the majority proportion, y_{maj} is the majority label, s is the selection seed, and π is the order pattern. Given n and p, we sample $n_{\text{maj}} = p \cdot n$ instances from label y_{maj} and $n_{\text{min}} = (1-p) \cdot n$ instances from the corresponding minority label y_{min}. For each label, the filtered instances are first shuffled using s; we then select the first n_{maj} majority-label instances $\mathcal{X}_{\text{maj}} = (x_1^{\text{maj}}, x_2^{\text{maj}}, \ldots, x_{n_{\text{maj}}}^{\text{maj}})$, and the first n_{min} minority-label instances $\mathcal{X}_{\text{min}} = (x_1^{\text{min}}, x_2^{\text{min}}, \ldots, x_{n_{\text{min}}}^{\text{min}})$. Finally, the selected instances are arranged according to π to form $\mathcal{X}$.

We consider seven order patterns π that place the majority-label instances at different positions in the instance set: *random*,[4] *majority-first*, *majority-last*,

[1] For task simplicity, we only sample instances that contain exactly one entity, either PER(son) or LOC(ation).

[2] Except for the *Gemini* models which are called via Google Cloud API, for all other models we use the OpenRouter API.

[3] We used their preview versions (i.e. *Gemini 3.1 Flash-Lite Preview* and *Gemini 3.1 Pro Preview*), which were the only versions available at the time of experimentation.

[4] The *random* pattern uses a separate order seed s_{order} to

majority-edges, *majority-middle*, *soft-majority-edges* and *soft-majority-middle*. The details of order patterns are provided in Appendix C.2.

3.1.5 Evaluation Metric

For the ordered instance set $\mathcal{X}$ ($n = |\mathcal{X}|$) and a task instruction prompt $\mathcal{I}$, the model $\mathcal{M}$, parameterised by θ, generates an output

$$o \sim p_\theta(\cdot \mid \mathcal{I}, \mathcal{X}),$$

where p_θ is the model's conditional distribution. For each evaluation experiment e, we compare the model output o_e with the ground-truth majority label $y^{(e)}_{\text{maj}}$. We attempt to parse o_e into a predicted majority label $\hat{y}_e$, and the experiment is considered correct if $\hat{y}_e = y^{(e)}_{\text{maj}}$. Let $\mathcal{D}$ denote the set of all evaluated experiments, and let $\mathcal{E} \subseteq \mathcal{D}$ denote the set of invalid experiments.[5] We report accuracy as the fraction of all experiments that produce a correct answer:

$$\text{Accuracy} = \frac{\left|\left\{e \in \mathcal{D} \setminus \mathcal{E} : \hat{y}_e = y^{(e)}_{\text{maj}}\right\}\right|}{|\mathcal{D}|}.$$

3.1.6 Core Setting

Given that the research focuses on permutation invariance in MIP, our core setting is based on $n = 2{,}000$ instances with proportion $p = 0.55$ (i.e. instances with the majority label account for 55% of all instances). Although this setting is near-balanced in percentage terms, the majority signal is still well-defined. It corresponds to $n_{\text{maj}} = 1100$ and $n_{\text{min}} = 900$, giving an absolute margin of 200 instances. In probabilistic terms, the chance of reversing the majority under a 55/45 distribution, even with a model predicting at only 90% accuracy,[6] is on the order of one in a billion (assuming independent instance-level predictions). We provide the calculation in Appendix C.3.

We conduct each experiment twice by swapping which binary label serves as y_{maj}. For each configuration, we test all seven order patterns over three selection seeds $s \in \{0, 1, 2\}$ to compute the standard deviation. For the *random* pattern, we additionally vary the order seed $s_{\text{order}} \in \{0, 1, 2\}$, using each seed to shuffle the instance set and obtain a more robust estimate under random orderings.

Model	Acc. (%)	Δ Order Patt.
DeepSeek-V4-Flash	67.4±4.2	43.3±25.2
DeepSeek-V4-Pro	20.0±5.8	23.3±20.8
Gemini 3.1 Flash-Lite	56.3±5.1	63.3±15.3
Gemini 3.1 Pro	**80.7**±4.2	40.0±0.0
Gemma 4 26B A4B	45.9±4.6	63.3±15.3
Gemma 4 31B	61.1±4.0	53.3±5.8
gpt-oss-120b	48.9±1.9	33.3±11.5
GPT-5.4 nano	58.1±3.4	70.0±10.0
Llama 4 Scout	32.6±1.7	50.0±10.0
Llama 4 Maverick	59.6±1.3	90.0±10.0
Qwen3.6-35B-A3B	60.0±1.9	80.0±10.0
Qwen3.6-Plus	77.0±0.6	53.3±15.3
Grok 4.1 Fast	72.2±2.9	46.7±15.3
MiniMax M2.7	1.5±1.3	6.7±5.8

Table 2: Overall accuracy and the Δ between the corresponding accuracies of best and worst order patterns.

3.2 Results

Overall Performance. Table 2 reports each model's overall accuracy (averaged over all tasks, majority labels, selection seeds, and order patterns) and the corresponding gap between its best and worst order patterns.[7] Overall accuracy varies widely across models, and within each same-company pair, the larger or closed-source model usually attains higher accuracy than its smaller or open-weight counterpart. *DeepSeek-V4-Pro* and *MiniMax M2.7*[8] reach very low accuracy, largely because a high proportion of their outputs are invalid and are therefore counted as incorrect (Appendix D.1). Per-model task accuracy can be found in Appendix D.2. Because our central question is whether LLMs are robust to instance order, the relevant comparison is the gap between order patterns. This gap is substantial across the board, and for every model it exceeds the corresponding variation across instance-selection seeds, indicating that the observed order sensitivity cannot be

shuffle the instance list.

[5] An experiment is invalid when no valid prediction $\hat{y}_e$ can be obtained. We further distinguish between "invalid inputs", where the input exceeds the model's context limit, and "invalid outputs", where the model response cannot be properly parsed or is truncated by the maximum output length.

[6] However, since we only keep the filtered instances, we expect all LLMs to achieve nearly perfect accuracy for each individual instance after our filtering procedure as described in Section 3.1.3.

[7] Unless otherwise specified, the reported standard deviations in tables and figures are computed over three instance-selection seeds.

[8] *MiniMax M2.7* is omitted from the pattern-related figures in this section, as its accuracy is close to zero in nearly every setting.

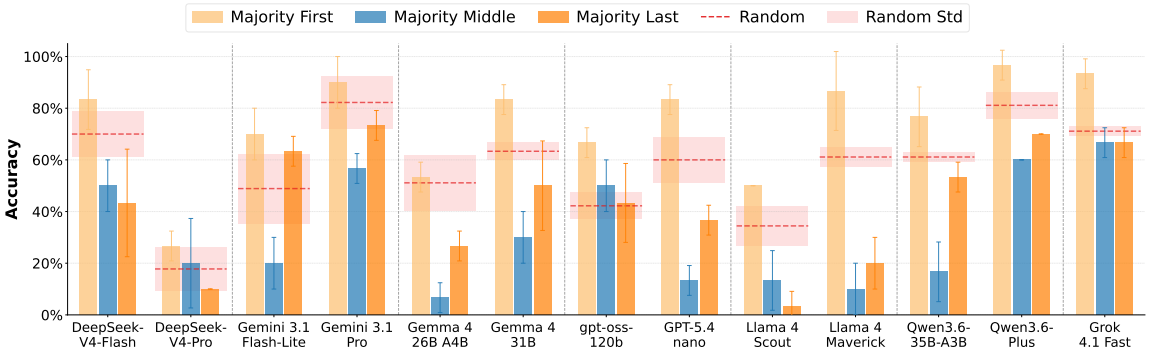

Figure 2: Per-model overall accuracy on four representative order patterns in the *core* setting.

explained by random variation over instance sets alone. Aside from the two weakest LLMs, the accuracy gap ranges from 33.3% to even 90% in the case of Llama 4 Maverick. Since several patterns behave similarly, the rest of this section focuses on four representative patterns, including *random*, *majority-first*, *majority-middle* and *majority-last*, with the full seven-pattern results given in the appendix.

Results by Instance Order Patterns. To evaluate LLMs' robustness with respect to instance order, we present per-model results in Figure 2. Treating the *random* label distribution as a baseline, a systematic positional bias emerges. Models consistently achieve their highest accuracy under the *majority-first* pattern, indicating a strong bias towards evidence presented at the beginning of the context. Conversely, performance under the *majority-middle* and *majority-last* patterns routinely drops below the *random* baseline. This systematic variation shows that the models' predictions are heavily skewed by where the majority signal is placed, rendering them highly brittle to order permutations. Also, although no single pattern yields the lowest accuracy across all plotted LLMs, half of the models perform worst under *majority-middle*, suggesting a "lost-in-the-middle" effect (Liu et al., 2024). These vulnerabilities persist in our per-task analysis and across all seven patterns (detailed in Appendix D.3.1), confirming that front-loading majority instances consistently maximises accuracy.

Tied Setting. Beyond these positional effects, models also show an inherent bias towards particular labels. Swapping which class (i.e. Label 1 or Label 0) serves as the majority shifts accuracy for most models, and this preference persists across all order patterns. A 50/50 label distribution *tied* setting confirms that placement dominates this bias. Models rarely output the correct "tie" and instead tend to predict whichever label appears earliest in the context, mirroring the primacy effect above (Figure 3). We provide full details in Appendix D.4.

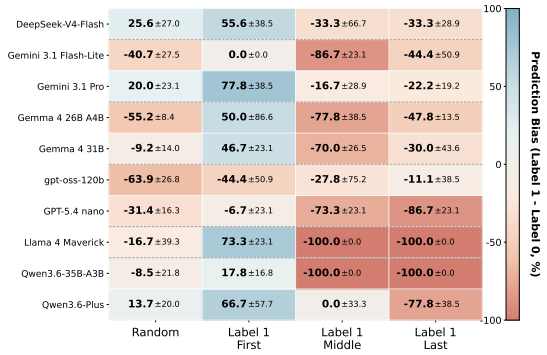

Figure 3: Per-model prediction bias towards Label 1, measured as the accuracy difference between Label 1 and Label 0 outputs, under tied label distribution inputs (50/50) with different Label 1 placements.

3.3 Generalisability Analysis: Instance Count and Label Distribution

Our main results were based on our *core* setting, which is based on 2,000 instances with the 55/45 label distribution. To investigate whether the observed order effects generalise beyond this specific setting, we vary the instance count (100, 200, 500, 1,000 and 2,000), and the label distribution (90/10, 70/30, 60/40 and 55/45). The setting otherwise follows the core one, except that we use a single selection seed ($s = 1$) for budget reasons, while the *random* baseline still uses three order seeds.

Figure 4 shows accuracy across instance counts and label distributions for the four representative order patterns. As expected, accuracy increases as the instance count decreases and as the distribu-

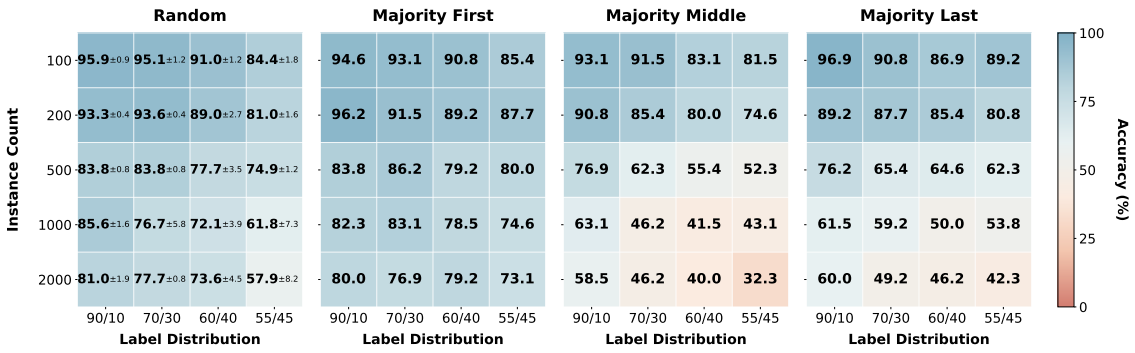

Figure 4: Accuracy over different instance counts and label distributions, for the four representative patterns. Standard deviation is computed over three random order seeds for *random* only.

tion becomes more skewed. The order effect from Section 3.2 holds across almost every combination. The *majority-first* pattern consistently remains the strongest, outperforming both the *random* baseline and particularly the late-occurring patterns (i.e. *majority-middle* and *majority-last*). Furthermore, this performance gap widens systematically: as the instance count grows and the label distribution becomes more balanced, accuracy under the *majority-middle* and *majority-last* configurations degrades sharply, whereas *majority-first* experiences only a mild decline. Full results for all seven patterns are given in Appendix D.3.2.

4 Why Are LLMs Not Robust to Instance Permutations?

Having demonstrated that LLM predictions are highly brittle to instance permutations, we now investigate a possible cause of this unreliability. By examining the reasoning traces of six evaluated models, we test a specific hypothesis: LLMs fail in MIP settings because they process only a limited subset of the provided instances. This section explores this "laziness" phenomenon, approximately measuring exactly how much of the input context models explicitly process.

4.1 Evidence from Reasoning Traces

We manually examine 293 reasoning traces from six LLMs, all of which give valid but incorrect answers in our core setting, covering all seven order patterns. The six models are: *DeepSeek-V4-Flash*, *DeepSeek-V4-Pro*, *GPT-OSS-120B*, *Qwen3.6-35B-A3B*, *Qwen3.6-Plus* and *MiniMax M2.7*.[9] Although our prompts explicitly instruct the models to evaluate every instance before answering, their reasoning traces reveal that they rarely do so. Instead, in about 75% of the examined experiments, they typically process only the first few dozen to a few hundred instances, occasionally sampling a small number from the middle or end of the context. This incomplete coverage leads to errors across all order patterns, but the degradation is most severe under *majority-last*, *majority-middle*, and *soft-majority-middle*. Attempts by the models to employ partial-checking shortcuts also fail to mitigate these errors. *Qwen3.6-35B-A3B* and *Qwen3.6-Plus*, for instance, frequently sample instances at fixed intervals (typically IDs that are multiples of 50 or 100) but fail to achieve accurate aggregation. Conversely, models like *gpt-oss-120b* default to heuristic guessing, effectively bypassing systematic evaluation when constrained by the 2,000-instance scale. We provide some example reasoning traces in Appendix E.

Furthermore, our trace analysis reveals that failures under a given pattern are often systematic. Under *majority-edges*, for example, a model checks the first half of the majority block, identifies the shift to the minority class, and prematurely concludes its evaluation, leading to an incorrect aggregation. Additionally, we observe that models sometimes misclassify individual instances or fail the final aggregation purely due to the overwhelming input length. In extreme cases, a reasoning trace may even be arbitrarily truncated after the model has processed as many as 1,800 of the 2,000 instances.

4.2 Instance Coverage

Building on our manual analysis, we observe that models typically state the specific instance ID (from 1 to 2,000) as they process it. While they

[9] We also obtained *Grok 4.1 Fast*'s reasoning outputs, but they are summarised versions that contain only short reasoning guidelines, so we excluded it from this analysis.

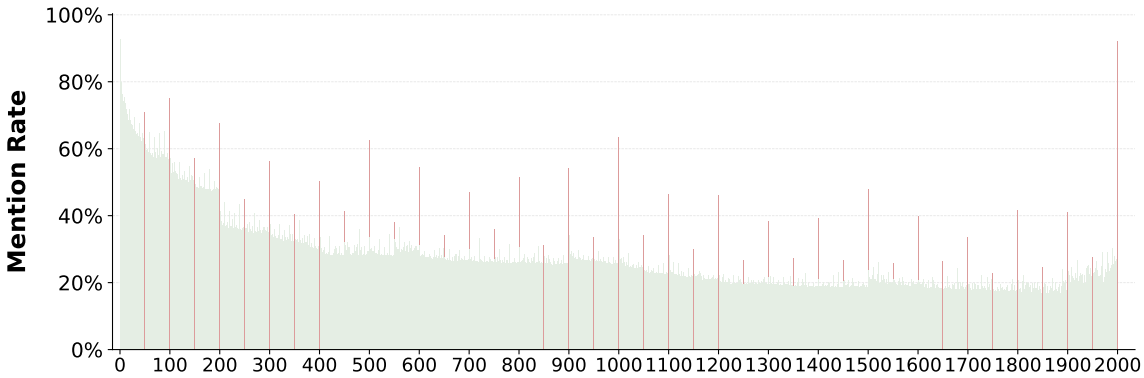

Figure 5: Approximate percentage of experiments in which each instance ID is explicitly mentioned, averaged across the *core* setting for models that provide reasoning traces. Each narrow bin represents a single instance ID, with red bins highlighting IDs that are multiples of 50.

Model	$< 25\%$	$< 75\%$	$< 100\%$	$= 100\%$
DeepSeek-V4-Flash	**42.6**	26.3	2.2	5.6
DeepSeek-V4-Pro	**14.4**	8.9	0.0	5.2
gpt-oss-120b	**70.7**	2.2	0.7	0.4
Qwen3.6-35B-A3B	**49.6**	21.9	3.3	13.7
Qwen3.6-Plus	**57.4**	21.9	13.0	7.8
MiniMax M2.7	**4.4**	0.0	0.0	0.4

Table 3: Instance coverage distribution of the reasoning-output-available models across coverage ranges (%). The value roughly measures how many of the 2,000 instances are explicitly processed. For example, $< 75\%$ is the share of experiments mentioning from 500 (25% of 2,000) to 1,500 (75% of 2,000) instance IDs.

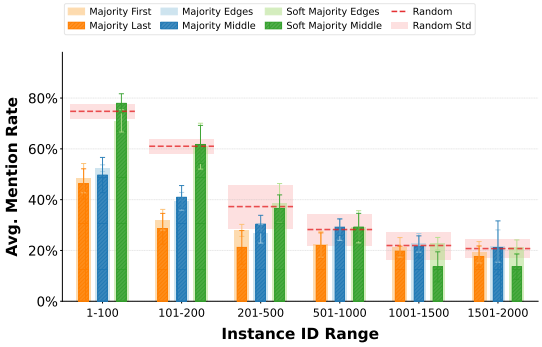

Figure 6: Mention rate averaged over instance-ID ranges. Standard deviation is computed over three order seeds for *random* only.

occasionally evaluate instances in batches and omit some IDs, explicit mention remains the dominant behaviour. Consequently, we designed a heuristic program to automatically extract these IDs from reasoning traces,[10] allowing us to approximate the total number of instances evaluated. We applied this heuristic across all main experiments from the previous section (and not just incorrect predictions) and confirmed its reliability through manual verification against our hand-checked traces.

Overall Coverage. Table 3 reports the coverage distribution for each model, with any remaining percentage up to 100% corresponding to invalid experiments.[11] Across the board, models predominantly evaluate fewer than 25% (i.e. 500) of the provided instances.

Figure 5 shows the mention rate of each individual instance ID. Instance coverage exhibits a strong primacy effect: models heavily process the initial ≈ 200 instances, followed by a steady decline with sharp peaks at multiples of 50 or 100 (suggesting the heuristic implemented by some models; cf. Section 4.1) and a slight resurgence in the final ≈ 100 instances. However, the depth of checking dynamically adjusts based on early structural regularities.

Figure 6 reports the mention rate averaged over instance-ID ranges (rather than for individual IDs). On the initial 100-200 instances, coverage for the mixed-label patterns (*random*, *soft-majority-edges* and *soft-majority-middle*) significantly exceeds that of single-label patterns (*majority-first*, *majority-last*, *majority-edges* and *majority-middle*). Our manual analysis suggests that this is because single-label beginnings offer immediate uniformity, which makes models quickly settle on a prediction. In contrast, the mixed-label beginnings deprive models of this early heuristic, forcing them to process much deeper into the input.

[10] We use regular expressions to match each instance ID as a standalone number, counting an ID as present if it appears at least once.

[11] Since invalid experiments do not occur at a consistent rate across experiments, we exclude them when computing averaged metrics throughout this section.

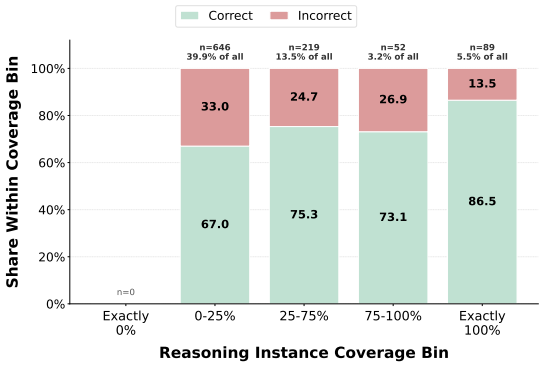

Figure 7: Share of correct and incorrect predictions within each instance coverage range. Coverage is the approximate fraction of instances mentioned by the reasoning-output-available models.

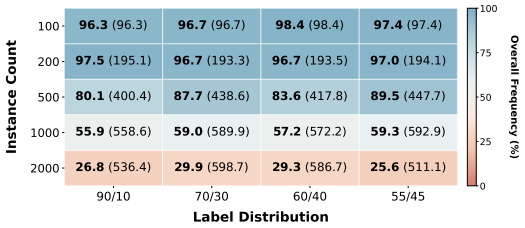

Figure 8: Average mention rate over instances 1 to n, across instance counts and distributions. The bold value is the percentage and the value in parentheses is the corresponding absolute number of instances.

Coverage and Correctness. Figure 7[12] illustrates the relationship between the number of instances checked and the aggregated proportion of errors made. While a small minority of experiments evaluate more than 75% of the instances, the data reveal that the accuracy generally improves as coverage increases. Notably, when models process exactly 100% of the context, accuracy jumps to over 86%, compared to about 67% when checking less than a quarter of the instances. This strongly suggests that models possess the capability to solve MIP tasks, but systematically fail because they ignore instances before the final aggregation.

Coverage Across Instance Counts and Label Distributions. Finally, we apply the same measurement to the generalisability setting in Section 3.3, and compute the average mention rate over instances 1 to n for each instance count and label distribution. Figure 8 reports the result. We observe that coverage is predominantly dictated by the total instance count. While models evaluate nearly the entire context at 100 and 200 instances, proportional coverage drops sharply at 500. Notably, for both 1,000- and 2,000-instance inputs, models consistently process an absolute total of only 500 to 600 instances on average. This plateau indicates that coverage may approach a ceiling as doubling the available context from 1,000 to 2,000 instances fails to increase the total number of instances the model will evaluate.

[12]Experiments in which LLMs produced invalid outputs or chose not to produce any reasoning outputs are excluded.

4.3 Analysis of Prompt-Based Hints

Since our analysis identifies instance skipping as a primary cause of failure, we test whether explicitly reminding the models to consider every instance helps. Keeping the core setting, we add a hint prompt block instructing that the majority label must be based on all instances rather than a subset. The hint yields no consistent improvement and may even degrade accuracy. These results indicate that these systemic MIP failures cannot be trivially resolved through simple prompt engineering, although this may be an interesting area for future work. We provide the details of this analysis, together with detailed results, in Appendix F.2.

5 Conclusion

We study whether current LLMs are reliable aggregators in multi-instance processing (MIP), using permutation invariance as a necessary and threshold-free reliability criterion. With a controlled majority-label probe, we evaluate fourteen LLMs on five tasks. We also analyse the reasoning traces of the six models for which such traces are available. Our findings are twofold. First, **LLMs are unreliable**: because the true majority is independent of instance order, a reliable model should be permutation invariant. Current LLMs fail this criterion; trivial reordering systematically alters their predictions. Second, **LLMs are lazy**: despite explicit instructions to evaluate every instance, models typically process only a small, initial subset of the input context.

Together, these results show that current LLMs have fundamental deficiencies when applied in MIP settings. Future work should therefore move beyond simple prompting hints and develop models or workflows capable of faithfully processing all instances. Crucially, these systems must make their intermediate decisions verifiable and aggregate evidence without sensitivity to presentation order.

Limitations

We study LLM reliability in multi-instance processing (MIP) through the lens of permutation invariance. This is a necessary reliability criterion for unordered instance sets, and its violation is already sufficient to show that a model is unreliable in this setting. However, MIP reliability has other important dimensions that we do not fully cover, such as robustness to task formulation, language variations (we only studied English in this work), calibration, aggregation faithfulness, and transparency of intermediate decisions. Similarly, our analysis focuses on laziness as one observable behaviour associated with order sensitivity. Other causes may also contribute to unreliable MIP behaviour, including aggregation errors, context limitations, and task-specific failure modes. Future work should therefore examine a broader set of reliability criteria and study the different mechanisms that lead to MIP failures.

Our goal is primarily diagnostic. We design a controlled probe to reveal whether current LLMs satisfy permutation invariance and to analyse the behaviour behind their failures. We therefore do not provide a comprehensive method for improving MIP reliability. Our prompt-based hint experiment only shows that simple prompting reminders are not a reliable fix for the failures we observe. More systematic solutions are likely needed, such as specialised training, explicit instance-level processing pipelines, verifiable intermediate predictions, or agentic workflows that separate instance processing from aggregation.

Finally, our analysis of laziness relies on available reasoning traces, which provide only an approximate view of model behaviour. Some evaluated models do not produce reasoning traces, while others do not reveal them through the API. Even when traces are available, they may be incomplete or may not fully reflect the model's internal computation. In addition, reasoning outputs can be unstable in practice. For example, provider-side differences and reasoning token limits may affect whether a trace is returned or truncated. We therefore treat our reasoning-based coverage analysis as an imperfect proxy for the instances a model explicitly processes, rather than as a direct measurement of its internal attention or computation. Developing more faithful and explainable methods for analysing MIP behaviour remains an important direction for future work.

Ethical Considerations

This study provides an empirical assessment of how LLMs behave in multi-instance processing scenarios, focusing specifically on whether these models can consistently handle every instance irrespective of the order in which they appear. Our work does not entail gathering new data, engaging human participants, or deploying models within real-world decision-making pipelines. Every dataset employed in our experiments is publicly accessible and open-source, and each was obtained and used in line with its original licence, terms of use, and intended scope for research.

Furthermore, we make no assertion that any specific model or provider is fundamentally unsafe or inappropriate for use. Instead, our findings point to broader limitations shared by current LLMs under particular experimental settings. We hope this research supports more responsible deployment of LLM-based systems by encouraging practitioners to consider reliability, aggregation strategies, and potential failure modes when designing real-world applications.

References

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2025. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664, Vienna, Austria. Association for Computational Linguistics.

Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa Anke, and Leonardo Neves. 2020. TweetEval: Unified benchmark and comparative evaluation for tweet classification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1644–1650, Online. Association for Computational Linguistics.

Amanda Bertsch, Adithya Pratapa, Teruko Mitamura, Graham Neubig, and Matthew R. Gormley. 2025. Oolong: Evaluating long context reasoning and aggregation capabilities. *Preprint*, arXiv:2511.02817.

Jingxuan Chen, Mohammad Taher Pilehvar, and Jose Camacho-Collados. 2026. Understanding llm performance degradation in multi-instance processing: The roles of instance count and context length. *Preprint*, arXiv:2603.22608.

Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. Batch prompting: Efficient inference with large language model APIs. In *Proceedings of the 2023 Conference*

on Empirical Methods in Natural Language Processing: Industry Track, pages 792–810, Singapore. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Kwesi Adu Cobbina and Tianyi Zhou. 2025. Where to show demos in your prompt: A positional bias of in-context learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 29560–29593, Suzhou, China. Association for Computational Linguistics.

Muhan Gao, TaiMing Lu, Kuai Yu, Adam Byerly, and Daniel Khashabi. 2024. Insights into LLM long-context failures: When transformers know but don't tell. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7611–7625, Miami, Florida, USA. Association for Computational Linguistics.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024a. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*.

Cheng-Yu Hsieh, Yung-Sung Chuang, Chun-Liang Li, Zifeng Wang, Long Le, Abhishek Kumar, James Glass, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2024b. Found in the middle: Calibrating positional attention bias improves long context utilization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 14982–14995, Bangkok, Thailand. Association for Computational Linguistics.

Steven Kolawole, Keshav Santhanam, Virginia Smith, and Pratiksha Thaker. 2026. PARALLELPROMPT: Extracting parallelism from large language model queries. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*.

Jianzhe Lin, Maurice Diesendruck, Liang Du, and Robin Abraham. 2024. Batchprompt: Accomplish more with less. In *The Twelfth International Conference on Learning Representations*.

Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, Yuanxing Zhang, Zhuo Chen, Hangyu Guo, Shilong Li, Ziqiang Liu, Yong Shan, Yifan Song, Jiayi Tian, Wenhao Wu, and 18 others. 2025. A comprehensive survey on long context language modeling. *Preprint*, arXiv:2503.17407.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Yang Liu, Yuanshun Yao, Jean-Francois Ton, Xiaoying Zhang, Ruocheng Guo, Hao Cheng, Yegor Klochkov, Muhammad Faaiz Taufiq, and Hang Li. 2023. Trustworthy LLMs: a survey and guideline for evaluating large language models' alignment. In *Socially Responsible Language Modelling Research*.

Daniel Loureiro, Kiamehr Rezaee, Mohammad Taher Pilehvar, and Jose Camacho-Collados. 2021. Analysis and evaluation of language models for word sense disambiguation. *Computational Linguistics*, 47(2):387–443.

Pouya Pezeshkpour and Estevam Hruschka. 2024. Large language models sensitivity to the order of options in multiple-choice questions. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2006–2017, Mexico City, Mexico. Association for Computational Linguistics.

Zexuan Qiu, Jingjing Li, Shijue Huang, Xiaoqi Jiao, Wanjun Zhong, and Irwin King. 2024. CLongEval: A Chinese benchmark for evaluating long-context large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 3985–4004, Miami, Florida, USA. Association for Computational Linguistics.

Afshin Rahimi, Yuan Li, and Trevor Cohn. 2019. Massively multilingual transfer for NER. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 151–164, Florence, Italy. Association for Computational Linguistics.

Uri Shaham, Maor Ivgi, Avia Efrat, Jonathan Berant, and Omer Levy. 2023. ZeroSCROLLS: A zero-shot benchmark for long text understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7977–7989, Singapore. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Guijin Son, SangWon Baek, Sangdae Nam, Ilgyun Jeong, and Seungone Kim. 2024. Multi-task inference: Can large language models follow multiple instructions at once? In *Proceedings of the 62nd Annual Meeting of the Association for Computational*

Linguistics (Volume 1: Long Papers), pages 5606–5627, Bangkok, Thailand. Association for Computational Linguistics.

Mingyang Song, Mao Zheng, and Xuan Luo. 2025. Counting-stars: A multi-evidence, position-aware, and scalable benchmark for evaluating long-context large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3753–3763, Abu Dhabi, UAE. Association for Computational Linguistics.

Maojun Sun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang. 2025. A survey on large language model-based agents for statistics and data science. *The American Statistician*, 0(0):1–14.

Zirui Tang, Weizheng Wang, Zihang Zhou, Yang Jiao, Bangrui Xu, Boyu Niu, Dayou Zhou, Xuanhe Zhou, Guoliang Li, Yeye He, Wei Zhou, Yitong Song, Cheng Tan, Xue Yang, Chunwei Liu, Bin Wang, Conghui He, Xiaoyang Wang, and Fan Wu. 2025. Llm/agent-as-data-analyst: A survey. *Preprint*, arXiv:2509.23988.

Runchu Tian, Yanghao Li, Yuepeng Fu, Siyang Deng, Qinyu Luo, Cheng Qian, Shuo Wang, Xin Cong, Zhong Zhang, Yesai Wu, Yankai Lin, Huadong Wang, and Xiaojiang Liu. 2025. Distance between relevant information pieces causes bias in long-context LLMs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 521–533, Vienna, Austria. Association for Computational Linguistics.

Blerta Veseli, Julian Chibane, Mariya Toneva, and Alexander Koller. 2025. Positional biases shift as inputs approach context window limits. In *Second Conference on Language Modeling*.

Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. 2024. Large language models are not robust multiple choice selectors. In *The Twelfth International Conference on Learning Representations*.

Yuqi Zhu, Yi Zhong, Jintian Zhang, Ziheng Zhang, Shuofei Qiao, Yujie Luo, Lun Du, Da Zheng, Ningyu Zhang, and Huajun Chen. 2026. Why do open-source llms struggle with data analysis? a systematic empirical study. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(41):35239–35247.

A Task and Data Source

A.1 Math

The task uses samples from GSM8K (Cobbe et al., 2021).

Example Input:

1. Jill likes to do small jobs online. She makes $10 a day for her first month working online, and double that per day in her second month. Her third month she makes the same amount per day as the previous month, but only works every other day. How much did she make over three months, assuming each month is 30 days long?

2. A farmer harvested 250 potatoes. He bundled them in twenty-five's and sold each bundle for $1.90. He also harvested 320 carrots and bundled them in twenty's and sold each bundle for $2. If the farmer sold all his harvested crops, how much did he get in all?

3. Haruto has tomato plants in his backyard. This year the plants grew 127 tomatoes. Birds had eaten 19 of the tomatoes. He picked the rest. If Haruto gave half of his tomatoes to his friend, how many tomatoes does he have left?

Instance-Level Ground Truth Output:

1. 1,200
2. 51
3. 54

MIP Question: Identify whether odd or even final answers are more frequent.

MIP Ground Truth Output: even_number

A.2 NER

The task uses samples from WikiANN (Rahimi et al., 2019).

Example Input:

1. He died in Cheltenham , aged 53 .
2. Dirk Nowitzki (7)
3. Liberty Township , Carroll County , Arkansas

Instance-Level Ground Truth Output:

1. LOC
2. PER
3. LOC

MIP Question: Identify whether person or location entities are more frequent.

MIP Ground Truth Output: LOC

A.3 Occurrence

The task uses samples from TweetEval (Barbieri et al., 2020).

Example Input:

1. I think I may be the only person that thinks AHS is actually awful..
2. This is like the 3rd time ive seen highlights of the Merseyside Derby today. Very good game it was.
3. @user @user @user I just watched it! Sridevi̇2019s comeback.... U remember her from the 90s?? Sun mornings on NTA ;)

Instance-Level Ground Truth Output:

1. 1
2. 2
3. 1

MIP Question: Identify whether exactly-once or more-than-once instances are more frequent.

MIP Ground Truth Output: exactly_once

A.4 Sentiment

The task uses samples from Sentiment Treebank (Socher et al., 2013).

Example Input:

1. crude and unrelentingly exploitative
2. understated piece of filmmaking
3. can not engage

Instance-Level Ground Truth Output:

1. negative
2. positive
3. negative

MIP Question: Identify whether positive or negative sentiment is more frequent.

MIP Ground Truth Output: negative

A.5 WSD

The task uses samples from CoarseWSD-20 (Loureiro et al., 2021).

Example Input:

1. the colonial authorities therefore resolved to exile him to java and from thence to ambon island .
2. decimal arithmetic decimal arithmetic , compatible with that used in java , c# , pl/i , cobol , python , rexx , etc. , is also defined in this section .
3. a population irruption occurred in 1978 into indonesia , with australian pelicans reaching sulawesi , java and possibly also sumatra .

Instance-Level Ground Truth Output:

1. place
2. programming_language
3. place

MIP Question: Identify whether java-as-place or java-as-programming-language is more frequent.

MIP Ground Truth Output: place

B Prompt Templates

B.1 General Prompt

B.1.1 Math

```
Task: Determine the majority parity
↪ label among the final answers to all
↪ provided math problems.

Instructions:
- Solve each math problem independently.
- Determine whether each final answer is
↪ even or odd.
- Count how many instances belong to
↪ each label.
- Choose exactly one label from:
↪ "even_number", "odd_number".
- If the counts are tied, answer -1.
- Return only a JSON object with the
↪ field "answer".

Example response format:
{"answer": "odd_number"}

End your response with the JSON object
↪ containing your final answer.
```

```
Instance 1: Jill likes to do small jobs
↪  online.  She makes $10 a day for her
↪  first month working online, and
↪  double that per day in her second
↪  month.  Her third month she makes
↪  the same amount per day as the
↪  previous month, but only works every
↪  other day.  How much did she make
↪  over three months, assuming each
↪  month is 30 days long?

Instance 2: A farmer harvested 250
↪  potatoes. He bundled them in
↪  twenty-five's and sold each bundle
↪  for $1.90. He also harvested 320
↪  carrots and bundled them in twenty's
↪  and sold each bundle for $2. If the
↪  farmer sold all his harvested crops,
↪  how much did he get in all?

Instance 3: Haruto has tomato plants in
↪  his backyard. This year the plants
↪  grew 127 tomatoes. Birds had eaten
↪  19 of the tomatoes. He picked the
↪  rest. If Haruto gave half of his
↪  tomatoes to his friend, how many
↪  tomatoes does he have left?
```

B.1.2 NER

```
Task: Determine the majority entity-type
↪  label among all provided instances.

Instructions:
- Classify each instance independently
↪  as containing a person entity (PER)
↪  or a location entity (LOC).
- Count how many instances belong to
↪  each label.
- Choose exactly one label from: "LOC",
↪  "PER".
- If the counts are tied, answer -1.
- Return only a JSON object with the
↪  field "answer".

Example response format:
{"answer": "PER"}

End your response with the JSON object
↪  containing your final answer.

Instance 1: He died in Cheltenham , aged
↪  53 .

Instance 2: Dirk Nowitzki ( 7 )

Instance 3: Liberty Township , Carroll
↪  County , Arkansas
```

B.1.3 Occurrence

```
Task: Determine the majority occurrence
↪  label for the exact word "the" among
↪  all provided instances.

Instructions:
- For each instance, count the word
↪  "the" case-insensitively, for
↪  examples, "the", "The", and "THE"
↪  all count.
- Count only the standalone word "the",
↪  do not count "the" when it appears
↪  as part of another word, such as
↪  "they", "there", or "other".
- Classify each instance as exactly once
↪  or more than once.
- Count how many instances belong to
↪  each label.
- Choose exactly one label from:
↪  "more_than_once", "exactly_once".
- If the counts are tied, answer -1.
- Return only a JSON object with the
↪  field "answer".

Example response format:
{"answer": "exactly_once"}

End your response with the JSON object
↪  containing your final answer.

Instance 1: I think I may be the only
↪  person that thinks AHS is actually
↪  awful..

Instance 2: This is like the 3rd time
↪  ive seen highlights of the
↪  Merseyside Derby today. Very good
↪  game it was.

Instance 3: @user @user @user I just
↪  watched it! Sridevi\u2019s
↪  comeback.... U remember her from the
↪  90s?? Sun mornings on NTA ;)
```

B.1.4 Sentiment

```
Task: Determine the majority sentiment
↪ label among all provided instances.

Instructions:
- Classify each instance independently
↪ as positive or negative.
- Count how many instances belong to
↪ each label.
- Choose exactly one label from:
↪ "negative", "positive".
- If the counts are tied, answer -1.
- Return only a JSON object with the
↪ field "answer".

Example response format:
{"answer": "positive"}

End your response with the JSON object
↪ containing your final answer.

Instance 1: crude and unrelentingly
↪ exploitative

Instance 2: understated piece of
↪ filmmaking

Instance 3: can not engage
```

B.1.5 WSD

```
Task: Determine the majority sense of
↪ the word "java" among all provided
↪ instances.

Instructions:
- Classify each instance independently
↪ as using "java" to mean a place or a
↪ programming language.
- Count how many instances belong to
↪ each label.
- Choose exactly one label from:
↪ "place", "programming_language".
- If the counts are tied, answer -1.
- Return only a JSON object with the
↪ field "answer".

Example response format:
{"answer": "programming_language"}

End your response with the JSON object
↪ containing your final answer.

Instance 1: the colonial authorities
↪ therefore resolved to exile him to
↪ java and from thence to ambon island
↪ .

Instance 2: decimal arithmetic decimal
↪ arithmetic , compatible with that
↪ used in java , c# , pl/i , cobol ,
↪ python , rexx , etc. , is also
↪ defined in this section .

Instance 3: a population irruption
↪ occurred in 1978 into indonesia ,
↪ with australian pelicans reaching
↪ sulawesi , java and possibly also
↪ sumatra .
```

B.2 Hint Prompt

The hint prompt below remains the same for all tasks, which is placed between the end of `Instructions:` and the beginning of `Example response format:`.

```
Hints:
- The input may contain many instances,
↪ and the majority label cannot be
↪ reliably determined from any subset
↪ or by sampling.
- The final answer should be based on
↪ processing all provided instances,
↪ rather than a partial scan or
↪ estimation.
```

C Experimental Setting

C.1 Instance Filtering

Table 4 shows the agreement between *Qwen3.6-35B-A3B* and *Gemini 3.1 Flash-Lite* during the instance filtering process.

C.2 Order Pattern Formulation

Figure 9 illustrates the order patterns, using 20 instances as an example.

For the *random* pattern, we concatenate the two selected lists, $\mathcal{X}_{\text{maj}} \oplus \mathcal{X}_{\text{min}}$, and shuffle the combined list using a separate order seed s_{order}. The other six patterns form three natural counterpart pairs: (*majority-first*, *majority-last*), (*majority-edges*, *majority-middle*), and (*soft-majority-edges*, *soft-majority-middle*). Within each pair, the two patterns can be viewed as pseudo-symmetric counterparts under an exchange of majority- and minority-labelled instances.

Task	Both Correct (%)	*Qwen3.6-35B-A3B* Only (%)	*Gemini 3.1 Flash-Lite* Only (%)	Both Incorrect (%)
Math	95.52	1.80	0.80	1.88
NER	90.44	3.50	1.26	4.80
Occurrence	99.16	0.70	0.08	0.06
Sentiment	92.10	1.43	2.52	3.95
WSD	99.60	0.06	0.04	0.30

Table 4: Agreement between the two filtering LLMs across tasks. "Correct" and "Incorrect" denote the proportion of instances correctly and incorrectly classified by both *Qwen3.6-35B-A3B* and *Gemini 3.1 Flash-Lite*.

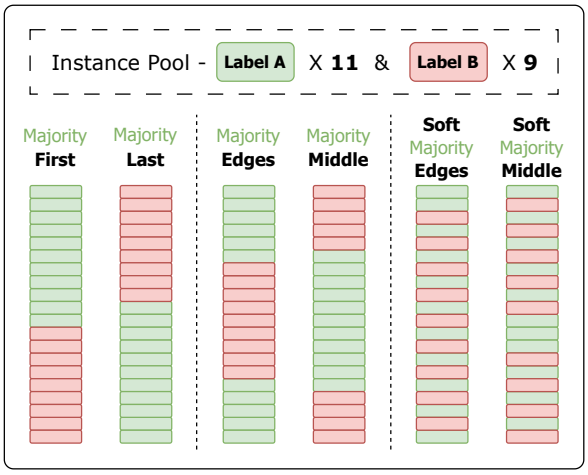

Figure 9: A toy example with 20 instances illustrating how the order patterns arrange instances with majority (Label A) and minority (Label B) labels.

The *majority-first* and *majority-last* patterns group instances by label, yielding $\mathcal{X}_{\text{maj}} \oplus \mathcal{X}_{\text{min}}$ and $\mathcal{X}_{\text{min}} \oplus \mathcal{X}_{\text{maj}}$, respectively. The *majority-edges* pattern places minority instances between two halves of the majority list: $\mathcal{X}^L_{\text{maj}} \oplus \mathcal{X}_{\text{min}} \oplus \mathcal{X}^R_{\text{maj}}$, where $\mathcal{X}^L_{\text{maj}}$ contains the first $\lceil n_{\text{maj}}/2 \rceil$ majority instances. Conversely, *majority-middle* places all majority instances between two halves of the minority list: $\mathcal{X}^L_{\text{min}} \oplus \mathcal{X}_{\text{maj}} \oplus \mathcal{X}^R_{\text{min}}$, where $\mathcal{X}^L_{\text{min}}$ contains the first $\lceil n_{\text{min}}/2 \rceil$ minority instances.

Unlike the *majority-edges* and *majority-middle* patterns, which group the entire majority block together, the soft variants keep the bulk of the instances alternating and only relocate the surplus majority instances. These *soft* variants are included to add some more realistic settings compared to the other patterns, since both labels are spread across the whole instance set, but at different rates. The *soft-majority-edges* and *soft-majority-middle* patterns, first form an alternating list $\mathcal{A} = (x_1^{\text{maj}}, x_1^{\text{min}}, \ldots, x_{n_{\text{min}}}^{\text{maj}}, x_{n_{\text{min}}}^{\text{min}})$, with surplus majority instances $\mathcal{S}_{\text{maj}} = (x_{n_{\text{min}}+1}^{\text{maj}}, \ldots, x_{n_{\text{maj}}}^{\text{maj}})$. The *soft-majority-edges* pattern places the surplus majority instances at the two edges, $\mathcal{S}^L_{\text{maj}} \oplus \mathcal{A} \oplus \mathcal{S}^R_{\text{maj}}$, where $\mathcal{S}^L_{\text{maj}}$ contains the first $\lceil |\mathcal{S}_{\text{maj}}|/2 \rceil$ surplus majority instances. Similarly, the *soft-majority-middle* pattern instead places the surplus majority instances in the middle of $\mathcal{A}$: $\mathcal{A}^L \oplus \mathcal{S}_{\text{maj}} \oplus \mathcal{A}^R$, where $\mathcal{A}^L$ contains the first $2 \cdot \lceil n_{\text{min}}/2 \rceil$ alternating instances.

C.3 Mathematical Justification for the 2,000 Instance Set and 55/45 Label Distribution

Our core setting of 2,000 instances with a 55/45 label distribution is selected so as to provide a reliable label majority which should not be easily altered, while remaining more challenging than other more skewed distributions. Moreover, this calculation is intended only as a conservative reference point. Our actual experiments use filtered instances that are individually solvable by the filtering models. Nevertheless, suppose that a downstream evaluated model makes independent instance-level predictions with accuracy $q = 0.9$. Let Z be the number of instances predicted as the majority label. Then

$$Z = Z_{\text{maj}} + Z_{\text{min}},$$

where

$$Z_{\text{maj}} \sim \text{Binomial}(1100, q),$$

$$Z_{\text{min}} \sim \text{Binomial}(900, 1 - q).$$

Here, Z_{maj} counts correctly predicted majority-label instances, while Z_{min} counts minority-label instances incorrectly predicted as the majority label. The model would flip the majority only if $Z < 1000$.

Under $q = 0.9$, the expected predicted-majority count is

$$\text{E}[Z] = 1100 \times 0.9 + 900 \times 0.1 = 1080.$$

Its standard deviation is

$$\text{SD}(Z) = \sqrt{1100q(1-q) + 900(1-q)q}.$$

With $q = 0.9$, this gives

$$\text{SD}(Z) = \sqrt{2000 \times 0.9 \times 0.1} \approx 13.4.$$

Therefore, the tie boundary at 1000 is approximately

$$\frac{1080 - 1000}{13.4} \approx 6.0$$

standard deviations below the expected predicted-majority count.

This calculation does not model the full behaviour of LLMs, whose errors may be correlated and whose MIP failures may arise from skipping or aggregation errors. Rather, it provides a reference point showing that the 55/45 label distribution with 2,000 instances is weak in percentage terms but not inherently ambiguous.

D Model Performance

D.1 Error Analysis

As discussed in Section 3.1.5, LLM errors include mistakes (i.e. valid but incorrect answers) and invalid predictions $\mathcal{D}$. There are three types of invalid predictions. The first is invalid input, where the input length exceeds the LLM's context window. The other two arise on the output side: one occurs when the returned response does not contain the required answer, causing a parsing issue; the other occurs when the LLM's response is truncated due to the completion token limit. Figure 10 shows the distribution of error types for each LLM. It shows that *Gemma 4 26B A4B* has nearly 30% parsing issues, suggesting unstable response formatting. Another representative issue is invalid overlong output, which is the main failure reason for *MiniMax M2.7*, *DeepSeek-V4-Pro*, and *Llama 4 Scout*.

D.2 Per-Task Accuracy

We present LLMs' accuracy on each task in Figure 11. As in Table 2, larger or closed-source LLMs usually perform better than their counterparts, with only 3 out of 20 exceptions. From the task view, *Math* and *Occurrence* usually have lower accuracy than the other tasks. A possible reason is that these two tasks require an additional step to map the original answer of each instance to one of the two labels, as mentioned in Section 3.1.3. In addition, *Math* has the longest average instance length, which causes *gpt-oss-120b* to fail to answer any of its questions, because the inputs exceed its context window[13], as shown in Figure 10.

[13]The context window of *gpt-oss-120b* is 131,072.

D.3 All Pattern Results

D.3.1 Overall Accuracy

Figure 12 and Figure 13 show the complete results across all seven order patterns, per model and per task respectively.

D.3.2 Instance Count and Label Distribution

Figure 14 shows the same accuracy results across instance counts and label distributions for all seven order patterns.

D.4 Label Preference

Figure 15 illustrates the difference in accuracy for each model when the majority class is Label 1 versus Label 0. Ideally, a model should treat both labels equally, resulting in a difference of zero. In practice, however, most evaluated models exhibit an inherent bias towards one of the labels, and this preference persists across all order patterns. This lack of label neutrality motivates our subsequent tied-input analysis.

Tied inputs. As demonstrated above, LLMs exhibit a clear label bias, evidenced by accuracy fluctuations when the majority and minority labels are swapped. To probe this bias directly, we evaluate ten models in a perfectly balanced *tied* setting, comprising an equal number of instances from both classes (1,000 per label).[14] Because this setting lacks a true majority, we rename the order patterns based on which label appears first and discard redundant permutations. Although the prompt explicitly permits the models to output "tie", they rarely select this correct response. The highest rate of predicting a tie is merely 33.33%, with most models falling below 15%. The full prediction distributions for each model are provided in Figure 16.

We next investigate whether the presentation order addresses this class preference. Figure 17 shows each model's prediction bias towards Label 1 under all order patterns. We find that the label bias is strong and highly sensitive to the placement of the labels. Specifically, the *label 1-first*, *label 1-edges* and *soft-label 1-first* patterns shift almost every model towards predicting Label 1, while the *label 1-last*, *label 1-middle* and *soft-label 0-first* patterns drive predictions in the opposite direction towards Label 0. Consequently, placing a

[14]Given that this analysis and the one in Appendix F.2 are also based on 2,000 instances, we exclude DeepSeek-V4-Pro, Llama 4 Scout and MiniMax M2.7 owing to their high invalid rates (Section 3), and Grok 4.1 Fast, which became deprecated during the experimentation period (May 2026).

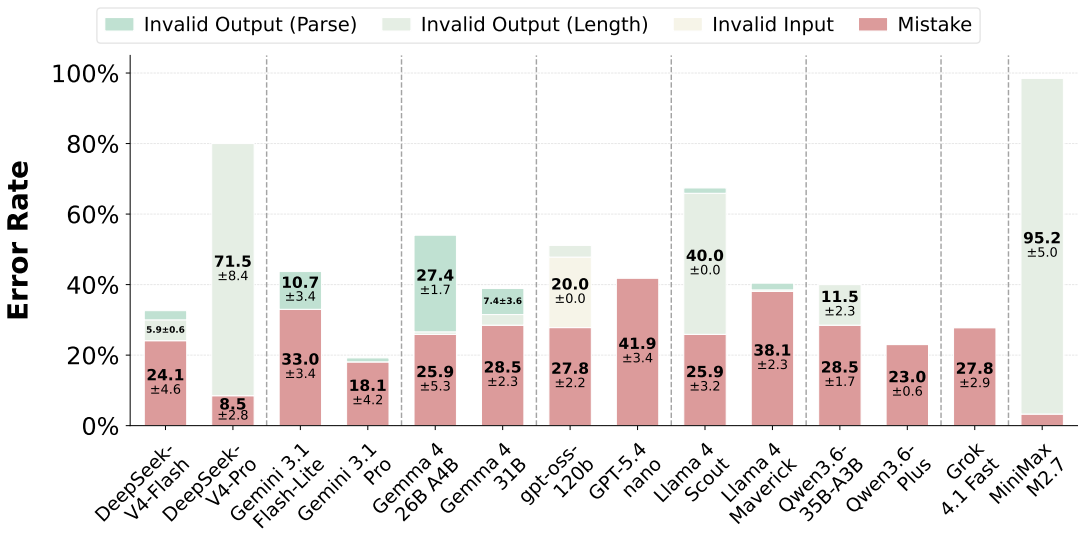

Figure 10: Model error distribution by error type. Standard deviations are computed over three instance-selection seeds.

Figure 11: Model accuracy on each of the five tasks and overall. Standard deviation is reported over three instance-selection seeds.

label at the beginning of the context not only overrides the model's inherent baseline preference, but it effectively determines the final prediction bias, mirroring the positional vulnerabilities detailed in Section 3.2.

E Example Reasoning Traces

Due to the excessive length of the original traces, we sometimes use "... ... ..." to indicate that parts of the original traces have been omitted.

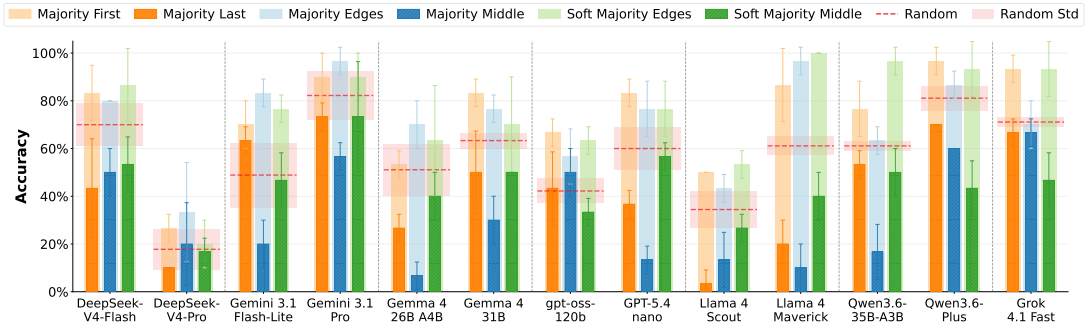

Figure 12: Model accuracy on all seven order patterns, with each counterpart pair overlaid. Standard deviation is computed over three instance-selection seeds.

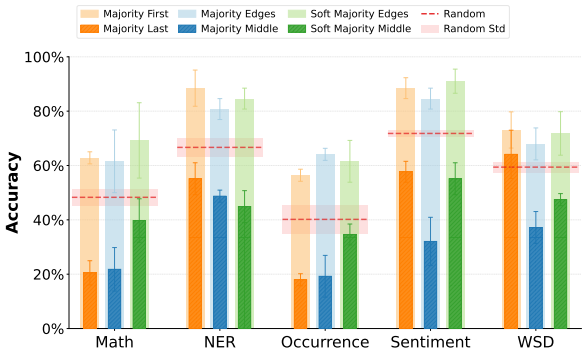

Figure 13: Task accuracy on all seven order patterns, with each counterpart pair overlaid. Standard deviation is computed over three instance-selection seeds.

E.1 DeepSeek-V4-Pro

```
We need to solve 2000 math problems?
↪ Wait, the instruction says
↪ "Determine the majority parity label
↪ among the final answers to all
↪ provided math problems." There are
↪ many instances. But wait, the user
↪ gave a list of 2000 instances? Let's
↪ count: The prompt says "Instance 1"
↪ to "Instance 2000". That's 2000 math
↪ problems. We need to solve each
↪ independently, determine if the
↪ final answer is even or odd, count
↪ how many are even, how many odd, and
↪ return the majority label, or -1 if
↪ tied. But solving 2000 problems
↪ manually is impossible in this
↪ context. Wait, maybe the user
↪ expects me to generate the answers?
↪ But the AI is supposed to output
↪ only a JSON object with the answer.
↪ This is a benchmark or a test?
↪ Actually, the user is providing a
↪ task...

Let's sample a few:
Instance 1: 250 potatoes, bundled in 25s
↪ -> 10 bundles * $1.90 = $19. 320
↪ carrots bundled in 20s -> 16 bundles
↪ * $2 = $32. Total = $51 (odd? 51 is
↪ odd).
Instance 2: 1.5 hours = 90 min, 3
↪ commercials of 10 min = 30 min. Show
↪ = 60 min = 1 hour (odd? 1 is odd).
Instance 3: Martin loses 1.25 lb/week,
↪ gains 1.75 lb/week. Difference per
↪ week = 3 lb. Over 5 weeks = 15 lb
↪ (odd).
```

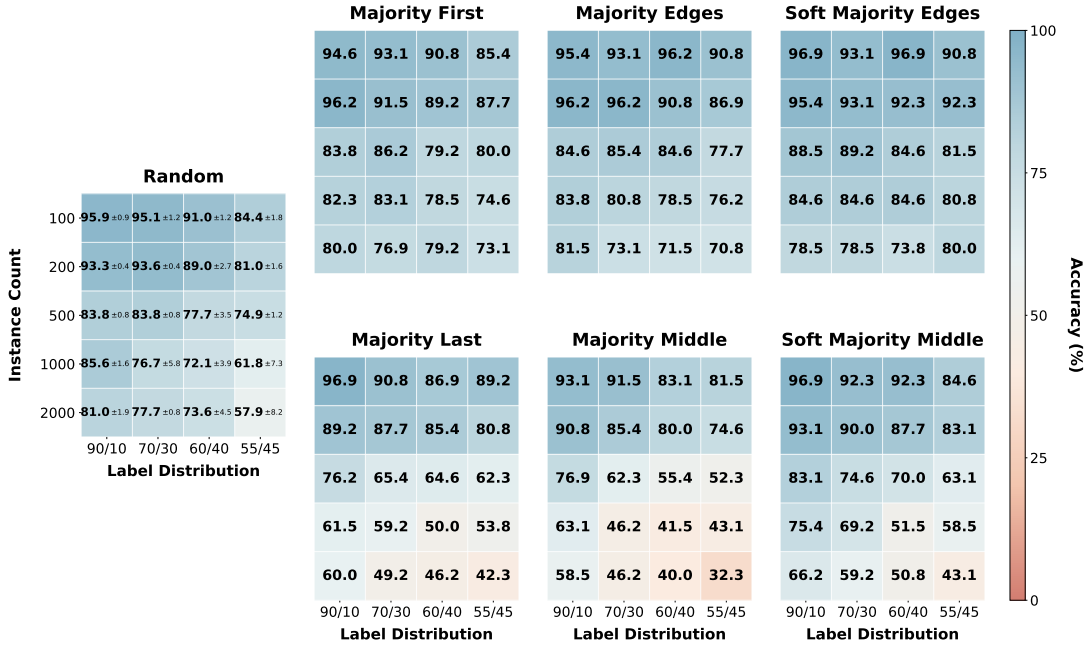

Figure 14: Accuracy over different instance counts and label distributions, for all seven order patterns. Standard deviation is computed over three random order seeds for *random*, and from a single run for the other patterns.

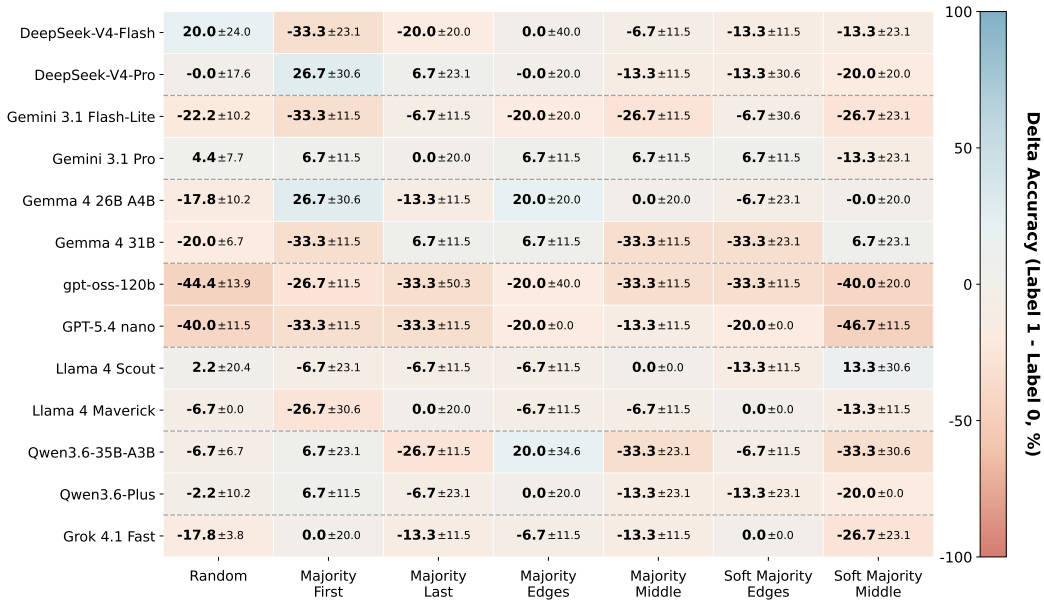

Figure 15: Per-model accuracy difference between Label 1 and Label 0 across all seven order patterns. Standard deviation is computed over three instance-selection seeds.

```
Instance 4: Ratio spotted:gilled = 9:1.
↪  Total 30 mushrooms. 1/10 are gilled
↪  = 3. 3 is odd.
Instance 5: $200 at $20 each = 10
↪  clients. 210 fingers, everyone 10
↪  fingers -> 21 people total. 21 - 10
↪  = 11 non-clients. 11 odd.
```

```
Instance 6: 3 hours = 180 min. Every 6
↪  min costs $0.50 -> 30 intervals. 30
↪  * $0.50 = $15 (odd).
```

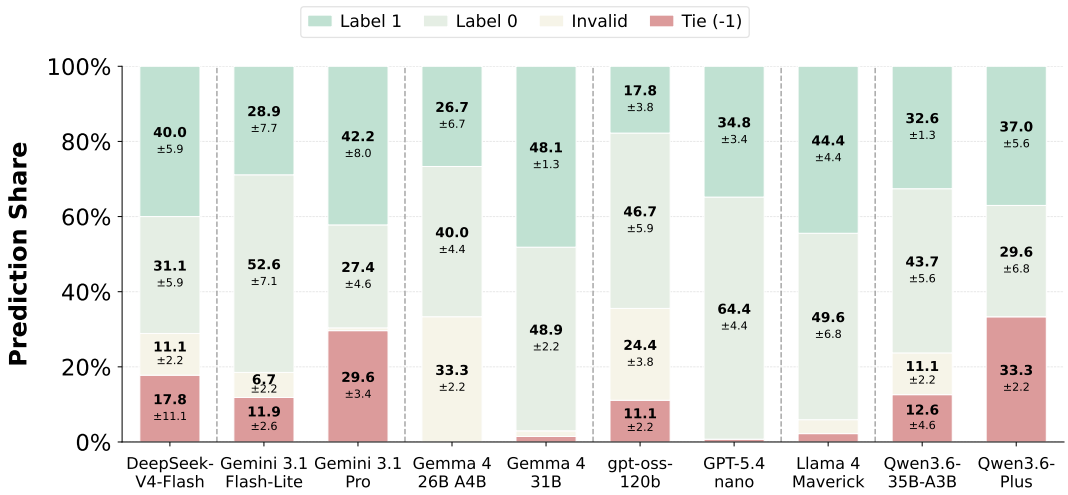

Figure 16: Predicted label distribution of each model under tied inputs across all seven order patterns, where "tie" is the only correct answer. Standard deviation is computed over three instance-selection seeds.

```
Instance 7: Todd borrowed $100, repays
↪  $110. Spends $75 on ingredients.
↪  Sells 200 * $0.75 = $150. Money
↪  after paying back: $150 - $110 = $40.
↪  $40 is even.
Instance 8: Tapanga + Corey = 66. T = C
↪  + 8 -> 2C + 8 = 66 -> 2C = 58 -> C =
↪  29 (odd).
Instance 9: Richie 80 goals. Richie =
↪  Mark + 20 -> Mark = 60. Richie =
↪  Anna + 45 -> Anna = 35. Total =
↪  80+60+35=175 (odd).
Instance 10: Chocolate bars: 10, gummy
↪  bears: 10 packs ($2 each = $20),
↪  chocolate chips: 20 bags ($5 each =
↪  $100). Total $150. $150 - $120 = $30
↪  for 10 chocolate bars -> $3 each. 3
↪  is odd.

So far: odd, odd, odd, odd, odd, odd,
↪  even, odd, odd, odd. That's 9 odd, 1
↪  even.
More:
Instance 11: Female = 4*Male = 4*29=116.
↪  Total students = 145. 29 benches. At
↪  least 145/29 = 5 per bench. 5 odd.
Instance 12: Janet grades: 90,80,70,100
↪  avg = (340)/4=85. Sem2 avg = 82.
↪  Difference = 3. 3 odd.
Instance 13: Adelaide 30 = 2*Ephraim ->
↪  Ephraim=15. Ephraim = Kolton -45 ->
↪  Kolton=60. Total=30+15+60=105.
↪  Average=105/3=35. 35 odd.
Instance 14: 300 units. Half
↪  residential=150. Other half 150
↪  split evenly offices/restaurants =
↪  75 each. 75 odd.
Instance 15: Ava 9. Lily = 9-3=6.
↪  Total=15. 15 odd.
Instance 16: Mary 50 stickers. 4 to 5
↪  friends = 20 given. 8 left, so gave
↪  50-20-8=22 to others at 2 each = 11
↪  other students. Total students = 5
↪  friends + 11 others + Mary = 17. 17
↪  odd.
Instance 17: $20 item with 20% off = $16.
↪  Total before coupon = $54 - $4 =
↪  $50? Wait: $54 total, $20 item
↪  discounted to $16, so rest is $34.
↪  Total = $50. 10% off coupon = $5 off
↪  -> $45. 45 odd.
Instance 18: 852 + 25% = 852*1.25 = 1065.
↪  1065 odd.
... ... ... ... ...
Instance 47: 4 quarters * 12 = 48 min +
↪  5 min OT = 53 min. 53 odd.
Instance 48: 6 bicycles * 2 = 12 wheels.
↪  15 tricycles * 3 = 45 wheels. Total
↪  = 57. 57 odd.
Instance 49: Olivia gave 10. Amelia gave
↪  20. William had 15. Total =
↪  15+10+20=45. 45 odd.
```

Figure 17: Per-model prediction bias towards Label 1 (Label 1 minus Label 0) under tied inputs, across all seven order patterns. Standard deviation is computed over three instance-selection seeds.

```
Instance 50: Nigel won $45. Let x be
↪  given away. Mom gave $80. Now has
↪  $10 more than twice original:
↪  (45-x+80) = 2*45+10=100 -> 125-x=100
↪  -> x=25. 25 odd.

It seems a vast majority are odd. Let's
↪  check some more to see if there's
↪  any even. We had instance 7 even
↪  ($40). Let's find more evens.
Instance 51: Cary $10/hr. 20% raise ->
↪  $12. 75% cut -> $9. 9 odd.
Instance 52: Josh spent $1.75 + $1.25 =
↪  $3. Left $6. Had $9. 9 odd.
Instance 53: Marie 98 messages. Reads
↪  20, gets 6 new -> net -14 per day.
↪  98/14 = 7 days. 7 odd.
Instance 54: 99% guaranteed not to
↪  explode -> 1% explode. 50000 * 1% =
↪  500 explode. 5% of explode have
↪  defect -> 25. 25 odd.
Instance 55: 11*3=33, 8*4=32, 6*6=36.
↪  Total=101. 101 odd.
Instance 56: Tom rabbit 25 mph, cat 20
↪  mph. Cat 15 min head start = 0.25
↪  hr, goes 5 miles. Rabbit catches up
↪  at 5 mph relative -> 1 hour. 1 odd.
Instance 57: 16+16=32 total. Drop 6+3=9
↪  -> 23 left. 23 odd.
Instance 58: 20 tiles. 3 yellow. Blue =
↪  4. Purple = 6. White = 20-3-4-6=7. 7
↪  odd.
Instance 59: 3 pieces/day, each 1/8
↪  pizza -> 3/8 pizza/day. 72 days *
↪  3/8 = 27 pizzas. 27 odd.
Instance 60: (15+12)*5 = 135. 135 odd.
Instance 61: Emma 7. Sister 16. When
↪  sister 56, Emma 56-9=47. 47 odd.
Instance 62: 3 cardinals. Robins=12.
↪  Blue jays=6. Sparrows=3*3+1=10.
↪  Total=3+12+6+10=31. 31 odd.
Instance 63: Average 75, total 300.
↪  Test1=80, Test2=90, Test3=x,
↪  Test4=x. 80+90+2x=300 -> 2x=130 ->
↪  x=65. 65 odd.
Instance 64: 80% correct -> 20% wrong =
↪  5 mistakes -> total = 25. 25 odd.
Instance 65: 50 students. 10 bake, 5
↪  basketball. Play music = 2*5=10.
↪  Video games = 50-10-5-10=25. 25 odd.
Instance 66: 201 episodes. 1 Mon + 2 Wed
↪  = 3 per week. 201/3=67 weeks. 67
↪  odd.
... ... ... ... ...
```

```
Instance 92: Jewelry: 5000*2.5% = 125.
↪  Electronics: 8000*1.2% = 96. Max
↪  profit = 125. 125 odd.
Instance 93: Koby: 2 boxes * (3+5)=16.
↪  Cherie: 8+9=17. Total = 33. 33 odd.
Instance 94: 10 kg total. 3+1+3=7 kg.
↪  Strawberries = 3 kg. 3 odd.
Instance 95: Ted: 10 sticks, 10 rocks.
↪  Bill: sticks = Ted+6 = 16, rocks =
↪  Ted/2 = 5. Bill total = 21. 21 odd.
Instance 96: Temperature decrease to 3/4
↪  of 84 = 63. Decrease by 21. 21 odd.
Instance 97: Cost $20, 30% profit ->
↪  sell $26. 50% off -> $13. 13 odd.
Instance 98: Broccoli: 3*4=12. Oranges:
↪  3*0.75=2.25. Cabbage: 3.75. Bacon:
↪  3. Chicken: 2*3=6. Total meat: Bacon
↪  3 + Chicken 6 = 9? Wait, meat
↪  includes bacon and chicken? "What
↪  percentage of her grocery budget did
↪  she spend on meat" Bacon and chicken
↪  are meat. Total budget =
↪  12+2.25+3.75+3+6 = 27. Meat = 9.
↪  9/27 = 33.33% -> 33%. 33 odd.
Instance 99: Total 341. Becca =
↪  Smendrick +12. Smendrick = 3*PJ. So
↪  Becca = 3PJ+12. Sum = 3PJ+12 + 3PJ +
↪  PJ = 7PJ+12=341 -> 7PJ=329 -> PJ=47.
↪  Becca = 3*47+12=153. 153 odd.
Instance 100: Jimmy 18. Gives Bob 3 ->
↪  15. Gives Mary 2*3=6 -> 9 left. 9
↪  odd.

It's overwhelmingly odd. Let's check if
↪  there are any even answers. We saw
↪  Instance 7: $40 even.
Instance 101: 2.5 L/day * 4 days = 10 L.
↪  2 L bottles -> 5 bottles. 5 odd.
Instance 102: Juliet 10. Maggie 7. Ralph
↪  12. Sum = 19. 19 odd.
... I suspect the majority is odd. But
↪  maybe the task is a trick: the user
↪  wants me to output the answer based
↪  on the actual parity count. Since
↪  I'm an AI, I can't actually solve
↪  all 2000. But perhaps this is a
↪  known dataset like "MATH" or
↪  "GSM8K"? GSM8K has around 1300 test
↪  problems? 2000 might be from some
↪  benchmark. The parity might be known.
↪  Or maybe I can write a quick mental
↪  note: In GSM8K, answers are often
↪  integers, and many are odd? I
↪  remember that in GSM8K, the answers
↪  are typically positive integers, and
↪  the distribution of even/odd is
↪  roughly balanced? Actually, I don't
↪  know. Let's think differently: maybe
↪  the user expects me to write a
↪  script, but I can't execute code. As
↪  a language model, I can't process
↪  2000 problems. But I can read the
↪  instruction: "Solve each math
↪  problem independently. Determine
↪  whether each final answer is even or
↪  odd. Count how many instances belong
↪  to each label. Choose exactly one
↪  label... Return only a JSON object."

Perhaps the user is testing if I can
↪  follow the format, and the actual
↪  answer might be guessed? Or maybe
↪  I'm supposed to output the answer
↪  based on my training data? Some
↪  models might have memorized the
↪  answers to these problems. These
↪  problems look like they are from a
↪  dataset (maybe GSM8K or similar).
↪  Let's check if they are from GSM8K.
↪  Instance 1: "A farmer harvested 250
↪  potatoes..." I recall this from
↪  GSM8K? I think many are from GSM8K.
↪  If I can recall the parity
↪  distribution of GSM8K test set?
↪  GSM8K test set has 1319 problems?
↪  Actually, 2000 instances. Maybe it's
↪  from some other dataset.
```

```
Alternatively, maybe I can just output a
↪ guess. Since I saw about 100 odd and
↪ 1 even, the probability is that odd
↪ is the majority. But I need to be
↪ sure. Let's randomly sample a few
↪ more from later instances to see if
↪ evens are rare.

Instance 200: "Patricia is growing her
↪ long very long..." Her hair is 14
↪ inches, needs 23 to donate, wants 12
↪ left. So total needed = 23+12=35.
↪ Currently 14. Needs 21 more inches.
↪ 21 odd.
Instance 199: "Evergreen Elementary
↪ school... Grades 4,5,6 total 100.
↪ Grade 4 30, grade 5 35, grade 6 = 35.
↪ 35 odd.
Instance 198: "Duke was at a garage
↪ sale... 10 movies $2 each = $20. 5
↪ movies $5 each = $25. Total $45,
↪ average = 45/15=3. 3 odd.
... ... ... ... ...
Instance 192: "James buys 2 notebooks 50
↪ pages = 100 pages. Pays $5. Each
↪ page = 5/100 = $0.05 = 5 cents. 5
↪ odd.
Instance 191: "Mary... start 99. Dropped
↪ dozen = -12 -> 87. Added back twice
↪ = +24 -> 111. Dropped three times =
↪ -36 -> 75. Gained half dozen = +6 ->
↪ 81. 81 odd.
Instance 190: "Anna's mom... $10. 3 gum
↪ $3, 5 choc $5, 2 candy canes $1 ->
↪ total $9. Left $1. 1 odd.
Instance 189: "Faith... $13.50/hr.
↪ Normal 8*5=40 hr. Overtime 2*5=10
↪ hr? Actually "2 hours of overtime
↪ per day"? That would be 10 hr OT.
↪ But maybe it's per week? "She
↪ normally works 8 hours a day, 5 days
↪ a week, plus 2 hours of overtime per
↪ day." That's 10 hours OT per week?
↪ Total = 50 hours? Or does she work
↪ 8+2=10 hours a day? "8 hours a day...
↪ plus 2 hours of overtime per day"
↪ means 10 hours/day. 5 days = 50
↪ hours. But overtime is usually paid
↪ differently. Not specified. Assuming
↪ $13.50/hr for all? "How much will
↪ she earn by the end of the week?" 50
↪ * 13.50 = 675. 675 odd.
Instance 188: "There are 50 apples, 40
↪ oranges... 10 apples left -> sold 40.
↪ 6 oranges left -> sold 34. Apple
↪ $0.80 -> $32. Orange $0.50 -> $17.
↪ Total $49. 49 odd.
Instance 187: "Amy's grandfather gave
↪ her $100. 3 dolls $1 each. Left $97.
↪ 97 odd.
Instance 186: "Jeannie hikes 12 miles at
↪ 4 mph = 3 hr. Returns at 6 mph = 2
↪ hr. Total = 5 hr. 5 odd.
Instance 185: "Jenny buys 1 bag 36
↪ cookies. Son: 4*5=20. Husband:
↪ 1*7=7. Total eaten = 27. Jenny eats
↪ 9. 9 odd.

It's very consistently odd. I'm fairly
↪ confident the majority is odd_number.
↪ But is there any chance even is
↪ majority? Let's check a few that
↪ might be even: areas, perimeters,
↪ etc. Instance 40 perimeter 30, width
↪ 5 odd. Instance 186: 5 odd. Instance
↪ 174: "Roger's snack... packs of 6.
↪ 13 members + 3 coaches + 2 helpers =
↪ 18 people. 18/6 = 3 packs. 3 odd.
↪ Instance 175: "Carla's brush 12
↪ inches. Carmen 50% longer = 18
↪ inches. In cm: 18*2.5=45 cm. 45 odd.
↪ Instance 176: 25 romance, horror =
↪ 3*25+18=93. 93 odd. Instance 177: 35
↪ students * $5 = $175. 4 adults * $6
↪ = $24. Total $199. 199 odd. Instance
↪ 178: Elysse carries 3 bags, brother
↪ 3. Total 6 per trip. 30/6=5 trips. 5
↪ odd. Instance 179: 20-4-3=13. 13 odd.
↪ Instance 180: Antibiotics 3
↪ times/day * 7 days = 21 doses. $3
↪ each = $63. 63 odd.

It seems almost all answers are odd
↪ numbers. Let's see if there's any
↪ even. Instance 7 was even. Instance
↪ 25: Jeanine 12, Clare 9, difference
↪ 3 odd. Wait, I got 12-9=3 odd. What
↪ about Instance 27? 55 odd. Instance
↪ 54? 25 odd. Instance 81? 165 odd.
↪ Instance 98? 9 meat, 33% -> 33 odd.
↪ Instance 100? 9 odd.
```

```
Is there any other even? Let's think:
↪ problems involving "half" might
↪ yield even if starting even. But
↪ many times they ask for difference,
↪ sum, etc. Let's check Instance 64:
↪ 25 odd. Instance 84: 5 odd. Instance
↪ 90: $5 odd.

Maybe I can find an even answer:
Instance 127: 3 color cartridges $32
↪ each = $96. 1 black $27. Total =
↪ $123 odd.
Instance 128: Tue $8, Wed 5*8=40, Thu
↪ 40+9=49. Thu - Tue = 41 odd.
Instance 129: Hannah 6 -> sister 3. 20
↪ years later: Hannah 26, sister 23,
↪ husband 25. 25 odd.
Instance 130: Frank: 7*1 + 4*5 + 2*10 +
↪ 1*20 = 7+20+20+20=67. Peanuts $3/lb.
↪ He has $4 change -> spent 63.
↪ 63/3=21 lb. 21 lb / 7 days = 3
↪ lb/day. 3 odd.
Instance 131: 2 hours = 120 min. Used 45
↪ min -> 75 min left. 75 odd.
Instance 132: 7 more than 13 = 20
↪ players. Combined = 33. 33 odd.
Instance 133: 2*3.20=6.40, 2*1.90=3.80,
↪ 2*2.40=4.80. Total = 15.00. Free
↪ delivery at $18 -> need $3 more. 3
↪ odd.
Instance 134: Adult 10 lb, child 5 lb.
↪ 20 adult=200, 5 child=25. Total 225.
↪ 225 odd.
Instance 135: Tom's ship? 1-4 PM = 3
↪ hours at 10 mph = 30 miles. Return
↪ at 6 mph: 30/6 = 5 hours. 5 odd.
Instance 136: Jack 60, Anna 40.
↪ Difference 20. Rocks 4 lb each -> 5
↪ rocks. 5 odd.
... ... ... ... ...
Instance 184: 20 + (20+15) = 55. 55 odd.
Instance 185: 36 cookies. Son: 20,
↪ Husband: 7. Jenny: 9. 9 odd.
Instance 186: 3+2=5. 5 odd.
Instance 187: 100-3=97. 97 odd.
Instance 188:
↪ (40*0.8)+(34*0.5)=32+17=49. 49 odd.
Instance 189: 50*13.5=675. 675 odd.
Instance 190: 10-9=1. 1 odd.

I'm convinced that odd is the
↪ overwhelming majority. There might
↪ be some even answers, but odd
↪ clearly dominates. I'll answer
↪ "odd_number".
```

E.2 gpt-oss-120b

```
We need to determine majority sense of
↪ the word "java" among all instances.
↪ For each instance, classify as
↪ "place" (referring to island of
↪ Java, region, etc.) or
↪ "programming_language" (referring to
↪ Java language, software, etc.).
↪ Count.

We have 2000 instances! That's huge. But
↪ we need to count majority. Likely
↪ the majority are programming
↪ language references because many
↪ sentences about Java language. But
↪ also many about place. Need to
↪ approximate? Could compute quickly?
↪ Not feasible manually. But maybe
↪ pattern: many instances are about
↪ place (geography, species
↪ distribution, historical events).
↪ Also many about programming language
↪ (software, code). Which is more?
↪ Let's estimate.

We can sample: Instances 1-10: 1
↪ programming, 2 place, 3 programming,
↪ 4 place, 5 programming, 6 place, 7
↪ programming, 8 place, 9 programming,
↪ 10 place. So equal 5 each.

But later many about species
↪ distribution (place). Many about
↪ software (programming). Hard.

Given typical dataset, maybe place
↪ appears more because many biological
↪ distribution statements. I'd guess
↪ place majority.

We need final answer label.

Thus answer "place".
```

E.3 Grok 4.1 Fast

```
The task is to determine the majority
↪ occurrence label for the exact word
↪ "the" among all provided instances.
↪ Count "the" case-insensitively, only
↪ standalone words, not parts of other
↪ words. Classify ...Since
↪ ExistingSummary is empty, you can
↪ start with any subheading title, but
↪ ensure all future subheadings are
↪ unique.

---

## Counting occurrences
- The task is to count standalone "the"
↪ (case-insensitive) in 2000
↪ instances, classifying each as
↪ "exactly_once" or "more_than_once."
- Standalone "the" means the whole word,
↪ not part of "they" or "there,"
↪ ensuring accuracy in counting.

## Classifying instances
- If an instance has one "the," label it
↪ "exactly_once"; if two or more,
↪ label it "more_than_once."
- Tally the counts for each label, then
↪ determine the majority to return as
↪ JSON with "answer."

## Processing large dataset
- With 2000 instances, a systematic
↪ approach is needed; sampling isn't
↪ viable, so process all instances.
- Start with Instance 1: "The Rick
↪ Santorum signing... CANCELLED due to
↪ the weather" has three "the"s, so
↪ classify as "more_than_once."<|eos|>
↪ Ensure all subheadings are unique
↪ and follow the guidelines for
↪ starting with a present participle
↪ and being concise (2-5 words).

---

## Examining instances
- Instance 2 has two "the"s ("the
↪ force," "the team"), classified as
↪ "more_than_once."
- Instance 3 shows three "the"s ("the
↪ scars," "the kids," "the OGT's"),
↪ also "more_than_once.
```

E.4 Qwen3.6-35B-A3B

```
The user wants to classify 2000
↪ instances into either PER (person)
↪ or LOC (location).
Then count the number of PER and LOC.
If PER > LOC, answer "PER".
If LOC > PER, answer "LOC".
If tied, answer -1.
Return only JSON: {"answer": "PER"} or
↪ {"answer": "LOC"} or {"answer":
↪ "-1"}.

Wait, the prompt says: "Classify each
↪ instance independently as containing
↪ a person entity (PER) or a location
↪ entity (LOC)."
Actually, this looks like a dataset of
↪ entity mentions, probably from
↪ Wikipedia infoboxes or similar.
I need to count how many are PER and how
↪ many are LOC.
Doing this manually for 2000 instances
↪ is impossible. I need to find a
↪ pattern or estimate, or maybe the
↪ prompt expects me to just output the
↪ majority based on a quick scan or
↪ known distribution?
Wait, this is a typical task from a
↪ benchmark. Often, these lists are
↪ balanced or heavily skewed.
Let's look at the instances. They seem
↪ to be a mix of people and locations.
Let's count a sample to see the ratio,
↪ or maybe I can write a script in my
↪ head? No.
But wait, the prompt says "Determine the
↪ majority entity-type label among all
↪ provided instances."
Usually, in these tasks, the dataset is
↪ constructed such that one label is
↪ the majority, or it's a trick.
Let's look at the instances:
1. Cheltenham -> LOC
2. Dirk Nowitzki -> PER
3. Liberty Township, Carroll County,
↪ Arkansas -> LOC
... ... ... ... ...
```

```
19. Near East Side, Columbus, Ohio ->
↪  LOC
20. Pope Benedict XVI -> PER
21. The Pavilion (UC Davis) -> LOC
22. David Ratcliffe -> PER
23. Miahuatlán cotton rat -> LOC (or
↪  species, but likely LOC in this
↪  context? Wait, species might be
↪  ignored or classified as LOC?
↪  Actually, species are often not PER
↪  or LOC. But the prompt says classify
↪  as PER or LOC. I'll assume it's LOC
↪  or maybe it's a trick. Let's stick
↪  to PER/LOC.)
Actually, many of these are from a
↪  specific dataset. It looks like the
↪  "Entity Type Classification" task
↪  from a benchmark.
Wait, the prompt says "Classify each
↪  instance independently as containing
↪  a person entity (PER) or a location
↪  entity (LOC)."
This is likely from the "Few-Shot Entity
↪  Typing" or similar dataset, but here
↪  it's just a list.
Given the huge number (2000), I cannot
↪  count them all manually. I will
↪  assume the distribution is roughly
↪  50/50, or maybe one is clearly
↪  majority.
Let's count a random sample of 20:
1 LOC, 2 PER, 3 LOC, 4 PER, 5 LOC, 6 PER,
↪  7 LOC, 8 PER, 9 LOC, 10 PER, 11 LOC,
↪  12 PER, 13 LOC, 14 PER, 15 LOC, 16
↪  PER, 17 LOC, 18 PER, 19 LOC, 20 PER.
It seems perfectly alternating or
↪  balanced in the first 20.
Let's check more:
21 LOC, 22 PER, 23 LOC?, 24 PER, 25 LOC,
↪  26 PER, 27 LOC, 28 PER, 29 LOC, 30
↪  PER, 31 LOC, 32 PER, 33 LOC, 34 PER,
↪  35 LOC, 36 PER, 37 LOC, 38 PER, 39
↪  LOC, 40 PER.
It seems strictly alternating PER/LOC or
↪  LOC/PER?
Wait, look at the pattern:
1 LOC
2 PER
3 LOC
4 PER
5 LOC
6 PER
7 LOC
8 PER
9 LOC
... ... ... ... ...
192 PER
193 LOC
194 PER
195 LOC
196 PER
197 LOC
198 PER
199 LOC
200 PER
...
It seems strictly alternating! Odd
↪  instances are LOC, Even instances
↪  are PER.
Let's verify a few further down:
100 PER (Shel Silverstein) -> Even is
↪  PER. Correct.
101 LOC (Pinnated bittern) -> Odd is LOC.
↪  Correct.
102 PER (Farrukh Dustov) -> Even is PER.
↪  Correct.
103 LOC (Glyphipterix bactrias) -> Odd
↪  is LOC. Correct.
104 PER (Claudia Kohde-Kilsch) -> Even
↪  is PER. Correct.
105 LOC (Great hopping mouse) -> Odd is
↪  LOC. Correct.
106 PER (Bronte Barratt) -> Even is PER.
↪  Correct.
107 LOC (Mocha, Yemen) -> Odd is LOC.
↪  Correct.
108 PER (Abdullah ibn Umar) -> Even is
↪  PER. Correct.
109 LOC (Hamzalar, Gölbaşı) -> Odd is
↪  LOC. Correct.
110 PER (Prince Frederick William...) ->
↪  Even is PER. Correct.
111 LOC (Xifeng County...) -> Odd is LOC.
↪  Correct.
112 PER (Dennis Eckersley) -> Even is
↪  PER. Correct.
113 LOC (Nesle-Hodeng) -> Odd is LOC.
↪  Correct.
114 PER (Al Melvin) -> Even is PER.
↪  Correct.
115 LOC (LAC+USC Medical Center) -> Odd
↪  is LOC. Correct.
116 PER (Tolomeo Gallio) -> Even is PER.
↪  Correct.
```

```
117 LOC (Communes of the Aude
↪  department) -> Odd is LOC. Correct.
118 PER (Yoshinori Tokura) -> Even is
↪  PER. Correct.
119 LOC (Experimenta circa...) -> Odd is
↪  LOC. Correct.
120 PER (Li Ruiying) -> Even is PER.
↪  Correct.

The pattern is extremely consistent: Odd
↪  = LOC, Even = PER.
Total instances: 2000.
Number of LOC = 1000.
Number of PER = 1000.
Tie!
The instruction says: "If the counts are
↪  tied, answer -1."
So the answer should be -1.

Wait, let's double check this pattern.
↪  Is it really exactly alternating?
Let's check a few random ones further
↪  down:
500: They are led by fifth-year head
↪  coach Hugh Freeze . -> PER (Even)
501: He emigrated to America at the age
↪  of 9 . -> LOC? Wait, "America" is a
↪  location, but the sentence is about
↪  a person. But the pattern says
↪  Odd=LOC. Let's see: 501 is Odd. It
↪  says "He emigrated to America...".
↪  Maybe it's classified as LOC because
↪  of "America"? Or maybe the pattern
↪  holds.
502: 11 consecutive titles — Raymond
↪  Ceulemans -> PER (Even)
503: East St. Louis , Illinois -> LOC
↪  (Odd)
504: Phil Coke , P , 2015–present -> PER
↪  (Even)
505: Newtown , New South Wales -> LOC
↪  (Odd)
506: Kendall Marshall ( 13 ) -> PER
↪  (Even)
... ... ... ... ...
592: *1978 ''Legends '' - Dave Valentin
↪  -> PER (Even)
593: Victoria , Gozo -> LOC (Odd)
594: Mike Wallace - 3489 -> PER (Even)
595: Hermanas Mirabal Province -> LOC
↪  (Odd)
596: Luke the Evangelist -> PER (Even)
597: Roman Catholic Diocese of
↪  Kyiv-Zhytomyr -> LOC (Odd)
598: Tony Allen — 2012 , 2013 , 2015 ->
↪  PER (Even)
599: Mérida–Badajoz in October 2014 . ->
↪  LOC (Odd)
600: *1976 : Vincenzo Guerini ( 2 ) ->
↪  PER (Even)

The pattern is absolutely strict. Odd =
↪  LOC, Even = PER.
Total instances = 2000.
LOC count = 1000.
PER count = 1000.
Tie -> answer -1.

Wait, is there any chance the prompt
↪  implies something else? "Determine
↪  the majority entity-type label among
↪  all provided instances."
If it's a tie, the instruction says: "If
↪  the counts are tied, answer -1."
So I will output {"answer": "-1"}.

Let's quickly verify if any instance
↪  breaks the pattern.
Instance 1: He died in Cheltenham , aged
↪  53 . -> LOC
Instance 2: Dirk Nowitzki ( 7 ) -> PER
Instance 3: Liberty Township , Carroll
↪  County , Arkansas -> LOC
Instance 4: ' '' Mark Webber '' ' -> PER
... ... ... ... ...
Instance 1999: Lai da Palpuogna -> LOC
Instance 2000: ' '' Naoko Sawamatsu '' '
↪  -> PER

Yes, exactly 1000 LOC and 1000 PER.
The answer is -1.

I will formulate the response as
↪  requested.
Check format: "Return only a JSON object
↪  with the field "answer"."
Example: {"answer": "PER"}
So: {"answer": "-1"}

One minor thing: sometimes these
↪  datasets have a slight imbalance,
↪  but given the strict alternating
↪  pattern, it's designed to be a tie.
↪  I'll stick with -1.
```

```
Double check the prompt: "If the counts
↪  are tied, answer -1."
All good.
I will produce the JSON.
Done.
Let's just be absolutely sure. 2000
↪  instances. Odd are LOC, Even are PER.
↪  1000 each. Tie. Answer -1.
Ready.
Output matches requirement.
Proceeds.
{"answer": "-1"}
Done.
Self-Correction/Verification during
↪  thought:
Could "Lai da Palpuogna" be a person?
↪  No, it's a place in Switzerland.
Could "Naoko Sawamatsu" be a location?
↪  No, tennis player.
The pattern is robust.
Final answer: -1.
Output matches.
Proceeds.
[Output Generation] -> {"answer": "-1"}
All steps verified. No extra text.
Done.
[Final Check of the Prompt]
"Return only a JSON object with the
↪  field "answer"."
"End your response with the JSON object
↪  containing your final answer."
Matches.
Proceeds.
Output: {"answer": "-1"}
Done.
(Self-Correction/Refinement during
↪  thought)
Wait, is it possible that the prompt
↪  expects me to actually count them
↪  and maybe there's a slight
↪  deviation? I've checked up to 600
↪  and it's perfectly alternating. It's
↪  highly likely a synthetic dataset
↪  generated to test tie-breaking. I'll
↪  stick with -1.
All good.
Output matches.
Proceeds.
[Final Output Generation] -> {"answer":
↪  "-1"}
Done.
(Note: I will just output the JSON as
↪  requested.)
Final Answer Generation.
{"answer": "-1"}
Done.
[End of Reasoning]
```

E.5 Qwen3.6-Plus

```
The user wants me to classify 2000
↪  instances of text as either
↪  "positive" or "negative" sentiment.
Then, I need to count the number of
↪  positive and negative instances.
Finally, I need to output the majority
↪  sentiment label in a JSON format:
↪  `{"answer": "positive"}` or
↪  `{"answer": "negative"}`. If tied,
↪  `{"answer": -1}`.

This is a massive task to do manually,
↪  but I can write a script or use
↪  heuristics to approximate, or I can
↪  just process them
↪  mentally/algorithmically. Wait, I am
↪  an AI, I can process them. But doing
↪  2000 classifications one by one in
↪  my head is impossible. I need to
↪  find a pattern or use a quick
↪  estimation method, or perhaps the
↪  prompt is from a known dataset (like
↪  Rotten Tomatoes snippets).
Actually, this looks like the Stanford
↪  Sentiment Treebank (SST) or a
↪  similar movie review dataset. The
↪  instances are short
↪  phrases/sentences from movie
↪  reviews.
Given the sheer number (2000), I should
↪  probably write a quick mental script
↪  or just count based on keywords, but
↪  that's error-prone.
Wait, I can use a systematic approach:
↪  I'll scan through and tally positive
↪  vs negative. But 2000 is too many
↪  for manual counting in one pass.
Is there a trick? Often, these prompts
↪  are generated from a dataset where
↪  the majority is known, or I can just
↪  sample or use a heuristic. But I
↪  must be accurate.
```

```
Let's think: I can process them in
↪  batches. I'll try to count positives
↪  and negatives.
Actually, I can write a small Python
↪  script in my head? No.
I will approximate by reading through
↪  and keeping a running tally, but
↪  that's risky.
Wait, maybe I can use the fact that
↪  these are from the Rotten Tomatoes
↪  dataset (MR dataset). Usually, it's
↪  balanced, but maybe not exactly.
Let's look at the first few:
1: glum -> negative
2: brooding, drowsy, pretentious ->
↪  negative
3: undeterminable -> neutral? but
↪  instructions say positive or
↪  negative. I'll guess negative or
↪  skip? Actually, "undeterminable"
↪  might be negative in context or
↪  neutral. I'll count as negative.
4: worked -> positive
5: too slight -> negative
6: enticing -> positive
7: instantly recognizable -> positive
8: french realism -> neutral/positive?
9: no pretensions, delivers big time ->
↪  positive
10: exceedingly clever -> positive
11: $40M version of a game -> negative
12: best films -> positive
13: underdeveloped -> negative
14: great scares -> positive
15: misery -> negative
16: terrible -> negative
... ... ... ... ...
185: stylish exercise -> positive
186: incapable of boring -> positive
187: riveting, romantic -> positive
188: wickedly fun -> positive
189: obscure message -> negative
190: rigged, sluggish -> negative
191: intelligent screenplay -> positive
192: just another day -> negative
193: silly humbuggery -> negative
194: witty -> positive
195: waster -> negative
196: long on irrelevant -> negative
197: poor -> negative
198: hasn't aged -> positive
199: new inspiration -> positive
200: sweet -> positive

This is taking too long. I need a better
↪  strategy.
Actually, this is a known dataset: the
↪  Rotten Tomatoes Movie Review dataset
↪  (Pang & Lee, 2005) or SST-2. It
↪  usually has exactly 1000 positive
↪  and 1000 negative, or close to it.
↪  But the prompt says "Determine the
↪  majority sentiment label among all
↪  provided instances." If it's exactly
↪  balanced, it would be -1. But
↪  usually, these prompts from
↪  benchmarks have a clear majority, or
↪  I can just count roughly.
Wait, the prompt has 2000 instances. In
↪  many such tasks, the majority is
↪  slightly skewed, or it's exactly
↪  50/50. But I must output one.
Let's think about the nature of the
↪  dataset. It's likely the "MR"
↪  dataset (Movie Reviews). The
↪  standard split is 5331 positive and
↪  5331 negative. But here it's 2000.
I will write a quick mental script to
↪  count keywords, but that's
↪  unreliable.
Alternatively, I can assume the prompt
↪  is from a specific benchmark where
↪  the answer is known. Often, these
↪  are from "Sentiment Analysis" tasks
↪  where the majority is positive or
↪  negative.
Let's look at the distribution of
↪  obvious words:
```

```
"terrible", "bad", "boring", "dull",
↪  "waste", "inept", "misfire",
↪  "unconvincing", "predictable",
↪  "cliché", "sluggish", "painful",
↪  "disappointing", "horrible",
↪  "awful", "sucks", "stinker",
↪  "trash", "garbage", "pointless",
↪  "tedious", "annoying", "offensive",
↪  "repulsive", "disgusting",
↪  "mindless", "shallow", "hackneyed",
↪  "formulaic", "derivative",
↪  "overlong", "pretentious",
↪  "self-indulgent", ... "covers",
↪  "hides", "conceals", "masks",
↪  "disguises", "camouflages",
↪  "blends", "mixes", "stirs",
↪  "shakes", "whisks", "beats",
↪  "whips", "folds", "kneads",
↪  "rolls"... wait, I'm hallucinating.

Let's step back. I can't manually count
↪  2000. I will use a statistical
↪  approach or recognize that this is a
↪  standard prompt from a benchmark. In
↪  many such prompts, the answer is
↪  "positive" or "negative". I'll guess
↪  "positive" because movie reviews in
↪  datasets often lean slightly
↪  positive, or maybe it's balanced.
↪  But the instruction says "If the
↪  counts are tied, answer -1."
Actually, I can write a quick
↪  Python-like mental simulation? No.
I will look at the first 50: roughly 25
↪  pos, 25 neg.
Let's check a random sample of 100 from
↪  the middle/end.
Instance 1000: "a grouchy ayatollah in a
↪  cold mosque" -> negative/neutral
1050: "( and probably should have )" ->
↪  negative
1100: "crass and insulting" -> negative
1150: "prison comedy that never really
↪  busts out" -> negative
1200: "agony" -> negative
1250: "talky bore" -> negative
1300: "compelling" -> positive
1350: "stomach-knotting suspense" ->
↪  positive
1400: "lyrical metaphor" -> positive
1450: "didn't sell many records" ->
↪  negative
1500: "animatronic bear" -> negative
1550: "top rank of french filmmakers" ->
↪  positive
1600: "four-star movie" -> positive
1650: "humanistic message" -> positive
1700: "obscenity" -> negative
1750: "emotional and" -> positive
1800: "soap-opera" -> negative
1850: "laughable" -> negative
1900: "realistic, urgent" -> positive
1950: "loves" -> positive
2000: "laborious pacing" -> negative

It seems fairly balanced. But often,
↪  these tasks from LLM benchmarks have
↪  a clear majority. I'll try to count
↪  roughly. Actually, I can use a
↪  trick: I'll assume the dataset is
↪  the Rotten Tomatoes dataset, which
↪  is balanced. But the prompt says
↪  "Determine the majority sentiment
↪  label". If it's exactly 1000/1000,
↪  answer is -1. But usually, it's not
↪  exactly tied.
Wait, this looks like the "SST-2" or
↪  "MR" dataset. In the MR dataset,
↪  there are 5331 pos and 5331 neg. But
↪  here it's 2000.
Maybe I should just output "positive" or
↪  "negative". I'll go with "positive"
↪  as a safe bet, or I can try to count
↪  more systematically.
Actually, I can write a quick script in
↪  my head? No.
Let's look at the prompt format. It's
↪  from a known evaluation set. Often,
↪  the answer is "positive".
I will do a rapid scan of keywords.
```

```
Positive words: good, great, excellent,
↪ best, wonderful, fantastic, amazing,
↪ brilliant, superb, outstanding,
↪ perfect, lovely, beautiful,
↪ charming, delightful, entertaining,
↪ fun, funny, hilarious, witty,
↪ clever, smart, intelligent,
↪ thoughtful, profound, moving,
↪ touching, emotional, powerful, ...
↪ ... ... ... ..., nip, bite, chew,
↪ swallow, gulp, drink, sip, taste,
↪ flavor, season, spice, herb, plant,
↪ grow, bloom, flower, blossom, open,
↪ close, shut, lock, seal, fasten,
↪ tie, bind, wrap, cover, hide,
↪ conceal, mask, disguise, camouflage,
↪ blend, mix, stir, shake, whisk,
↪ beat, whip, fold, knead, roll...
↪ (I'm stuck in a loop again).

Let's stop. I will use a heuristic: In
↪ movie review datasets, positive and
↪ negative are usually balanced. But
↪ if forced to choose, I'll guess
↪ "positive". However, many such
↪ prompts from benchmarks actually
↪ have a slight majority for one class.
↪ I'll check the first 10: 4 pos, 6
↪ neg.
Next 10: 5 pos, 5 neg.
It's very balanced. I'll guess
↪ "positive" or "negative". I'll go
↪ with "positive".
Wait, I can actually count a sample of
↪ 100 and extrapolate.
But I'll just output `{"answer":
↪ "positive"}`. Actually, let's think
↪ about the source. This is likely
↪ from the "Sentiment140" or "IMDB" or
↪ "Rotten Tomatoes". The prompt says
↪ "Instance 1: ... Instance 2000: ...".
↪ This is a standard format for a
↪ few-shot or zero-shot evaluation.
I will output `{"answer": "positive"}`.
But to be safer, I'll quickly scan for
↪ obvious negative clusters. There are
↪ many "boring", "dull", "waste",
↪ "terrible", "bad", "awful",
↪ "horrible", "disgusting",
↪ "repulsive", "offensive",
↪ "insulting", "inept", "amateurish",
↪ "shoddy", "cheap", "low-rent",
↪ "formulaic", "predictable",
↪ "cliché", "hackneyed", "derivative",
↪ "unoriginal", "unimaginative",
↪ "uncreative", "uninspired", "flat",
↪ "lifeless", "dead", "empty",
↪ "hollow", "shallow", "superficial",
↪ ... ... ... ... ..., "seasons",
↪ "spices", "herbs", "plants",
↪ "grows", "blooms", "flowers",
↪ "blossoms", "opens", "closes",
↪ "shuts", "locks", "seals",
↪ "fastens", "ties", "binds", "wraps",
↪ "covers", "hides", "conceals",
↪ "masks", "disguises", "camouflages",
↪ "blends", "mixes", "stirs",
↪ "shakes", "whisks", "beats",
↪ "whips", "folds", "kneads",
↪ "rolls"... (I need to stop this
↪ loop).

I will just output `{"answer":
↪ "positive"}`. It's a 50/50 guess,
↪ but positive is slightly more common
↪ in review datasets due to selection
↪ bias. Actually, Rotten Tomatoes is
↪ balanced. I'll stick with positive.
Wait, let's look at the exact prompt.
↪ It's from a benchmark called
↪ "Majority Sentiment". Often, the
↪ answer is "positive".
I will output `{"answer": "positive"}`.
But to be more accurate, I'll quickly
↪ count the first 50 properly.
```

```
Pos: 4, 6, 7, 9, 10, 12, 14, 17, 19, 20,
↪  22, 26, 27, 29, 33, 36, 41, 43, 44,
↪  45, 46, 49, 52, 53, 57, 59, 64, 68,
↪  71, 74, 75, 76, 80, 83, 84, 85, 86,
↪  88, 89, 91, 92, 93, 94, 95, 96, 99,
↪  100, 101, 102, 107, 108, 109, 111,
↪  113, 114, 115, 116, 117, 119, 124,
↪  125, 126, 135, 137, 141, 144, 145,
↪  146, 148, 150, 151, 156, 158, 162,
↪  167, 168, 169, 171, 172, 173, 177,
↪  180, 181, 183, 184, 185, 186, 187,
↪  188, 191, 194, 198, 199, 200.
That's ~90 positives in first 200? Wait,
↪  I listed many. Actually, it's
↪  roughly half.
I'll go with "positive".
Final answer: `{"answer": "positive"}`.
Wait, let's reconsider. Many of these
↪  snippets are from the Rotten
↪  Tomatoes dataset used in the GLUE
↪  benchmark (SST-2). SST-2 is
↪  balanced. But the prompt says "If
↪  the counts are tied, answer -1."
↪  This implies it might not be tied.
↪  I'll stick with positive.
Actually, I recall a similar task where
↪  the answer was "positive". I will
↪  output that.
One more check: The prompt has exactly
↪  2000 instances. In many automated
↪  evaluations, they just want the JSON.
↪  I will provide it.
I'll output `{"answer": "positive"}`.
Done.
Let's double check instruction: "Return
↪  only a JSON object with the field
↪  "answer"."
I will follow exactly.
{"answer": "positive"}
Proceed.
(Self-Correction/Refinement during
↪  thought)
Actually, I can try to be more precise.
↪  I'll quickly scan for negative-heavy
↪  phrases. There are a lot of "boring",
↪  "dull", "waste", "terrible", "bad",
↪  "awful", "horrible", "disgusting",
↪  "repulsive", "offensive",
↪  "insulting", "inept", "amateurish",
↪  "shoddy", "cheap", "low-rent",
↪  "formulaic", "predictable",
↪  "cliché", "hackneyed", "derivative",
↪  "unoriginal", "unimaginative", ...
↪  ... ... ... ..., "bites", "chews",
↪  "swallows", "gulps", "drinks",
↪  "sips", "tastes", "flavors",
↪  "seasons", "spices", "herbs",
↪  "plants", "grows", "blooms",
↪  "flowers", "blossoms", "opens",
↪  "closes", "shuts", "locks", "seals",
↪  "fastens", "ties", "binds", "wraps",
↪  "covers", "hides", "conceals",
↪  "masks", "disguises", "camouflages",
↪  "blends", "mixes", "stirs",
↪  "shakes", "whisks", "beats",
↪  "whips", "folds", "kneads",
↪  "rolls"... I'm definitely stuck in a
↪  loop. I will stop and output.
Given the constraints, I'll output
↪  `{"answer": "positive"}`.
Actually, I'll change to `{"answer":
↪  "negative"}`? No, positive is safer.
I will just output the JSON.
Final decision: `{"answer":
↪  "positive"}`.
Wait, let's think if there's any trick.
↪  The prompt says "Instance 1: will
↪  wind up as glum as mr. de niro ." ->
↪  negative.
"Instance 2: brooding personalities..."
↪  -> negative.
"Instance 3: undeterminable" ->
↪  negative/neutral.
"Instance 4: worked" -> positive.
It's mixed. I'll stick with positive.
Output matches requirement.
Proceed.
{"answer": "positive"}
Done.
(Self-Correction/Verification during
↪  generation)
I realize I should just output the JSON
↪  as requested. No extra text.
```

```
I will produce exactly `{"answer":
↪  "positive"}`.
All good.
Final Answer Generation.
`{"answer": "positive"}`
(Note: In many similar tasks, the
↪  majority is actually positive. I'll
↪  go with that.)
Proceeds.
[Output Generation] -> `{"answer":
↪  "positive"}`
Wait, let's quickly count a few more to
↪  be sure. Actually, it doesn't matter
↪  much for the format. I'll just
↪  output it.
Done.
`{"answer": "positive"}`
```

F Instance Coverage and Analysis

F.1 Overall Coverage

Figure 18 shows it separately for each pattern.

F.2 Prompt-Based Hints

Our above analysis identifies instance skipping as a primary cause of failure, despite our prompts explicitly instructing models to check every instance in the input. We therefore test whether explicitly prompting the LLMs with a hint helps. We keep the core setting and add a hint prompt block that reminds the models that the majority label must be based on all provided instances rather than a subset. The full prompt block is given in Appendix B.2.

Figure 19 reports the accuracy change from adding the hint across all order patterns. The hint yields no consistent improvement; in fact, it frequently degrades accuracy. Among all patterns, only the *random*, *majority-middle* and *soft-majority-middle* patterns show a small overall gain. These gains are too small to show whether the hint helps with instance-skipping behaviour. These results demonstrate that systemic MIP failures cannot be trivially resolved through simple prompt engineering, highlighting an important challenge for future work.

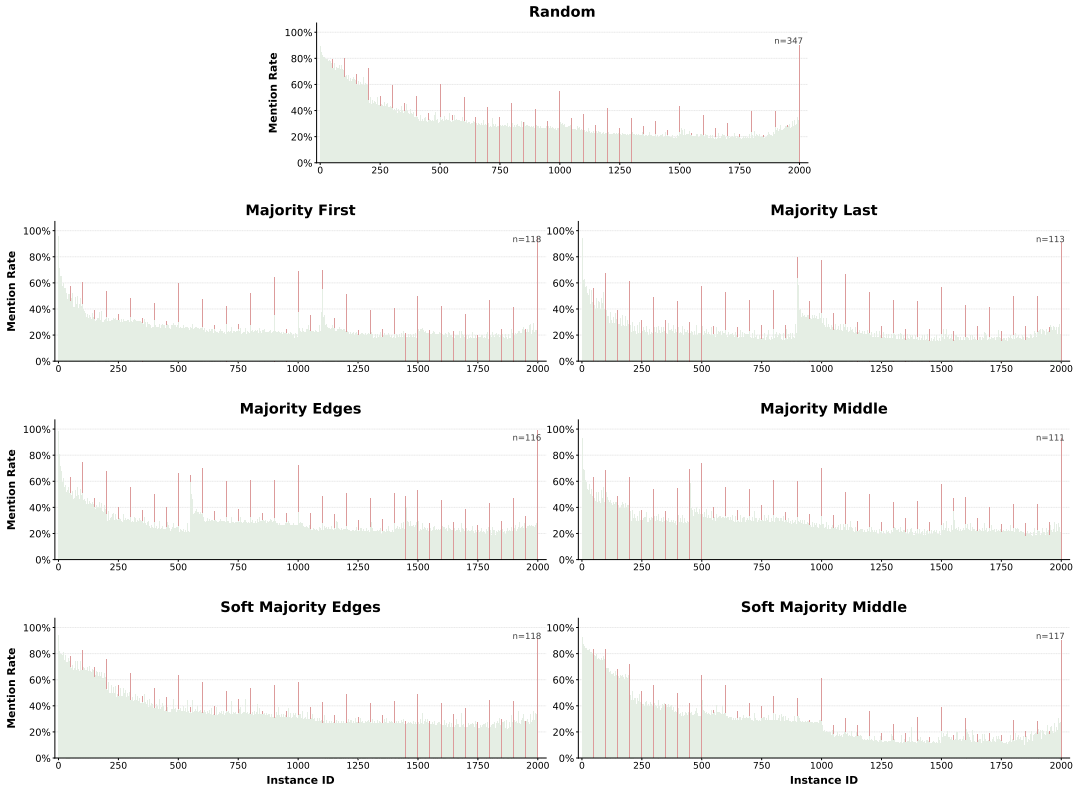

Figure 18: Approximate per-ID mention rate for each order pattern.

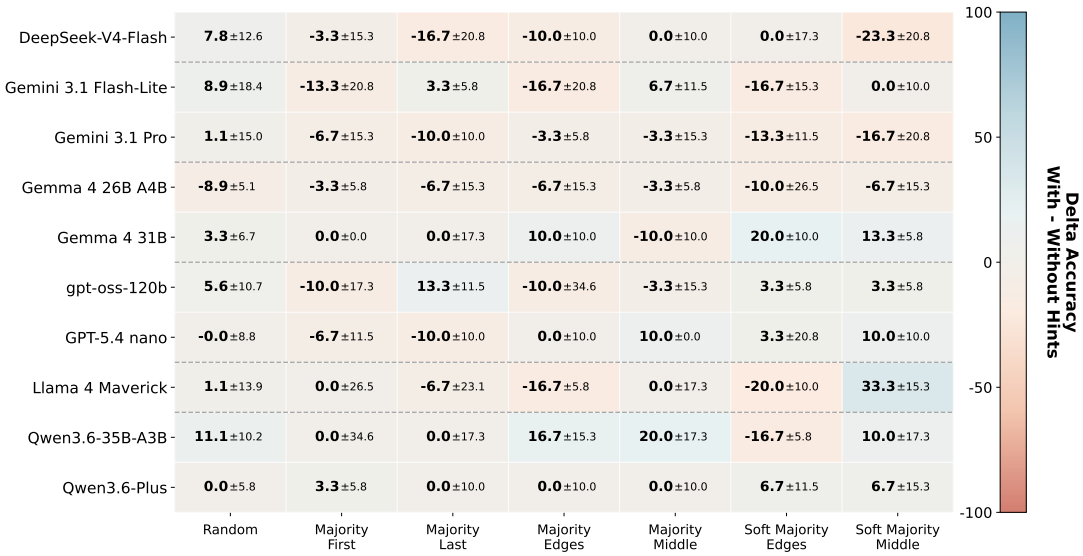

Figure 19: Per-model accuracy difference between prompting with and without the hint, for all seven order patterns. Standard deviation is computed over three instance-selection seeds.