Project Report: AbsenceBench 2.0 Replication and Cross-Domain Extension

Context: Trial Project for Conceptualization Lab (Project 5)

Author: Zekai Tong (University of Chicago)

Date: January 16, 2026

Repository: https://github.com/zkt02/AbsenceBench-2.0.git

Trial Decisions & Technical Summary.

(1) I introduced Recipes as a low-constraint procedural domain to test whether thinking mode improves instruction following while risking verbatim fidelity under strict line-level exact match; to isolate composition effects, I also ran a counterfactual composition re-weighting analysis that adjusts the zero-omission prevalence and clarifies how it shapes the interpretation of Sections 3.2–3.3.

(2) I treated thinking mode as a controlled intervention rather than comparing many models, prioritizing mechanism attribution and error analysis under an otherwise fixed evaluation pipeline.

(3) Because GitHub PRs and Numerical replicate closely, I interpret the large Poetry gap as domain- and setup-sensitive rather than a global implementation error, and I audit it along three axes: output scaling limits (scale penalty), strict exact-match sensitivity (EM penalty via qualitative cases), and execution or split-statistics differences that can disproportionately affect long-output settings.

The Double-Edged Sword of Reasoning: Analyzing Cross-Domain Discrepancies in AbsenceBench Replication and Extension

1 Introduction

AbsenceBench evaluates an LLM's ability to detect what is missing by comparing an original input with a modified version and requiring the model to output the omitted elements. In this project, I (1) replicate benchmark results using gemini-2.5-flash in both non-thinking and thinking modes on the three original AbsenceBench domains, and (2) extend the evaluation to a new domain, Recipes, constructed from 500 Kaggle samples under the same protocol. The remainder of this report focuses on explaining cross-domain performance differences with measurable evidence and analyzing model sensitivity to the amount and structure of omissions, with particular attention to how reasoning-enhanced LLMs trade off verbatim retrieval against logical reconstruction under strict line-level exact-match scoring. I address these questions with a reproducible evaluation pipeline and targeted analyses (distribution audits, composition re-weighting, and a qualitative error taxonomy), which together highlight coding, quantitative reasoning, and creative failure-mode identification.

Domain	Baseline (Non-thinking)	Thinking Mode	Delta (Δ)
Recipes	39.87	73.45	+33.58
GitHub PRs	26.47	40.50	+14.03
Poetry	16.32	17.20	+0.88
Numerical	91.91	97.10	+5.19

Table 1: reports the micro-F1 scores (in percentage points) of gemini-2.5-flash under baseline (non-thinking) and thinking modes across four domains, along with the absolute difference $\Delta = \text{Thinking} - \text{Baseline}$(positive values indicate improvement under thinking).

On the three original datasets, the replicated results on Numerical and GitHub PRs largely match those reported in the paper, and are even slightly higher in terms of micro-F1. However, the Poetry results are substantially lower than those reported in the paper. This discrepancy motivates Section 2, Analysis of the Poetry Performance Discrepancy, which investigates evaluation sensitivity and domain-specific failure modes with both quantitative audits and qualitative evidence.

2 Analysis of the Poetry Performance Discrepancy

This large discrepancy suggests that the Poetry setting may be especially sensitive to implementation details and domain-specific factors. In the following sections, I analyze several plausible explanations point by point and provide supporting evidence where possible.

2.1 Evidence of Sub-linear Scaling and the "Scale Penalty"

A detailed audit of the Poetry runs indicates that the low score is primarily driven by a sub-linear scaling mismatch between omission-set size and the model's realized output volume.

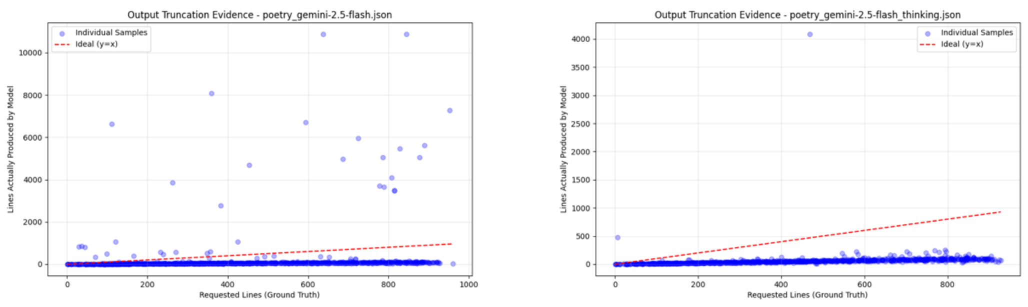

Figure 1: Comparison of scaling trends between Requested Lines and Produced Lines across the full range. The stark divergence from the y=x parity line illustrates a fundamental inability of the model to scale its output volume proportionally with the task complexity.

As shown in Figure 1, Produced Lines is positively correlated with Requested Lines, but the response is far from proportional: the gap to the $y = x$ parity line widens as omission size increases, implying a slope well below 1.0.

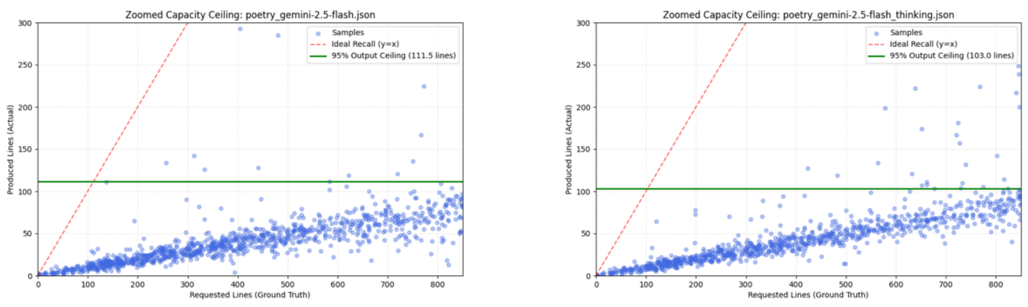

Figure 2: Zoomed analysis of the output capacity ceiling (0–300 lines). The horizontal clustering of data points below the P95 threshold (~103 lines) demonstrates a systemic "glass ceiling" that persists regardless of the reasoning mode or computational effort.

Figure 2 further shows a practical output ceiling near the 100-line range, which appears similar across baseline and thinking modes. Together, these results suggest that many Poetry instances exceed the model's effective capacity to enumerate all missing lines in a single response.

This induces a direct scale penalty in evaluation. Recipes typically have small omission sets (often <10 lines), where near-complete coverage is feasible. By contrast, Poetry has an average request size of 452.4 missing lines. Even if the model produces 50 to 80 lines, the achievable recall is mechanically bounded:

$$\text{Recall} = \frac{|\text{Correctly Predicted Missing Lines}|}{|\text{True Missing Lines}|} \approx \frac{50\text{--}80}{452.4} \approx 11\%\text{--}18\%$$

This recall dilution directly suppresses micro-F1 by increasing false negatives. Notably, the thinking configuration incurs substantial internal computation in my runs (Poetry-average 12,610 thinking tokens) without overcoming the output ceiling, which is consistent with an output-capacity bottleneck as the dominant limiting factor.

2.2 Exact-Match Error Penalty: Qualitative Case Evidence

A second contributor to the low Poetry score is an exact-match (EM) penalty that converts near-correct recoveries into zero-credit errors. Because the EM evaluator is all-or-nothing at the line level, even minor surface-form deviations, including grammatically improving punctuation or modernizing phrasing, can cause an otherwise correct line to be scored as incorrect.

Comparing wrongly_identified_lines with unidentified_lines shows that the model often recovers the semantic content of missing lines but fails EM due to recurring surface-form changes, most commonly punctuation over-normalization (e.g., ID 639), lexical modernization or synonym swaps (e.g., ID 0), and small start-word deviations. As illustrated in Table 2, these edits systematically inflate false negatives and depress micro- F1.

Failure Type	Ground Truth	Model Response	EM Outcome
Punctuation correction	Thy murder Brown William, …	Thy murder, Brown William, …	Full line counted wrong
Modern synonym swap	… white-fire insect,	… white-fire creature,	Full line counted wrong
Start-word deviation	Read this Song of Hiawatha!	To this Song of Hiawatha!	Full line counted wrong

Table 2: Micro-case comparisons illustrating how strict line-level exact match (EM) penalizes near-correct Poetry recoveries. Each example shows a minimal surface-form deviation between the ground truth and the model output that results in the entire line being scored as incorrect under EM.

2.3 Discrepancy in Execution Environment and Sample Statistics

In this replication, GitHub PRs and Numerical achieve micro-F1 scores broadly consistent with the original paper, and in some cases slightly higher. This alignment makes a global implementation error unlikely and suggests that the large Poetry gap is not explained by model capability or prompt compliance alone. Beyond Poetry's intrinsic difficulty, I therefore consider execution-environment constraints and evaluation-distribution differences that can disproportionately affect long-output regimes.

First, platform-level generation limits and rate constraints can materially change outcomes even with identical code. Provider defaults and deployment settings may differ across environments (e.g., Vertex AI vs. Google AI Studio). Because Poetry often requires very long responses, with an average request size of 452.4 missing lines, any hard cap on response length, timeout, or truncation would mechanically reduce recall and depress micro-F1. Consistent with this possibility, early runs with an overly large batch_size triggered repeated throttling and temporary account suspension, indicating that provider-side enforcement can become binding in this pipeline, even if it does not directly diagnose truncation.

Second, sample composition and length sensitivity can shift the aggregate score. My audit shows performance degrading as omission size increases, implying a strong dependence on the omission-length distribution. If the paper's evaluation emphasized shorter Poetry instances that remain below the observed output ceiling, while my replication includes a broader, heavier-tailed omission-size distribution, then the reported averages can diverge substantially under the same model and prompt. This motivates reporting length-stratified results and summarizing omission-size statistics for the evaluated split.

Finally, post-processing and normalization policy may contribute. The evaluation is sensitive to surface-form variation, and strict exact match penalizes minor punctuation edits and small lexical modernizations (Section 2). If the original pipeline used a different normalization policy prior to exact matching, such as punctuation handling, case folding, or whitespace canonicalization, its reported micro-F1 could be higher than a strict character-level comparison. This can be tested by re-scoring under alternative normalization settings and measuring the fraction of the gap they explain.

3 Analysis of Performance Differences on the Recipes Dataset

3.1 Sensitivity to Omission Rate and the Bimodal Polarization of F1

In this section, I replicate the visualization procedure from Figure 5 of the original paper, *AbsenceBench: Language Models Can't Tell What's Missing*, to analyze omission-rate sensitivity on the Recipes dataset.

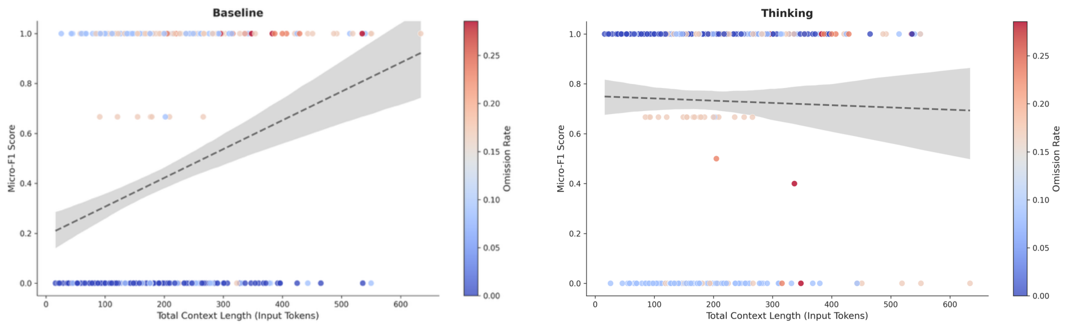

Figure 3: Recipes domain micro-F1 as a function of total context length (estimated tokens), with points colored by omission rate, shown for baseline (non-thinking) and thinking modes. Across both modes, low omission rates (blue) exhibit a pronounced bimodal pattern with mass at both $F1 = 0.0$ and $F1 = 1.0$, while higher omission rates (red) are more likely to cluster near $F1 = 1.0$. This highlights increased fragility under few omissions and the near all-or-nothing outcome distribution in this domain.

Figures 3 shows a clear pattern in the color-coded scatter plots, where deep blue denotes low omission rates and deep red denotes high omission rates. In the Baseline mode, performance is generally and positively associated with the omission rate: a dense cluster of low-omission (deep-blue) points lies on the $F1 = 0.0$ baseline, indicating frequent collapse when only a few lines must be recovered, whereas higher-omission instances are more likely to migrate toward the $F1 = 1.0$ region. Notably, low-omission instances also exhibit a bimodal tendency, with a visible mass near $F1 = 1.0$, consistent with the near all-or-nothing behavior in this domain. In the Thinking mode, while this positive association persists, it is notably less pronounced than in the Baseline, with instances exhibiting a more dispersed distribution across various omission levels. This observation, particularly the Baseline's behavior, directly supports the conclusion that the model is more fragile under few omissions, consistent with the broader omission-rate sensitivity patterns discussed in the original study.

A notable property of Recipes is the extreme bimodal, near all-or-nothing distribution of F1, with scores concentrated near 0.0 or 1.0 and few intermediate values. I attribute this polarization to the interaction of two factors. First, recipe instances typically contain only a small number of procedural steps, so the omission set is often tiny, with the number of missing lines N frequently less than 3. Under such small denominators, even a single miss or mismatch produces a discontinuous jump in precision and recall, leaving little room for partial credit. Second, the benchmark's strict exact-match scoring at the line level magnifies these effects: because recipe steps are short and stylistically uniform, minor surface-form deviations such as whitespace or punctuation differences can flip an otherwise correct line into an incorrect one. Together, these properties act as an amplifier, yielding the observed "all-or-nothing" outcome in Figure 3. Notably, the same small-steps structure also gives rise to another phenomenon of practical interest, which I analyze further in Section 3.2.

3.2 When Thinking Helps: A Comparison of Recipes and PRs in Zero-Omission Scenarios

From a computational perspective, GitHub PRs and Recipes are among the most structurally similar domains in AbsenceBench, as both are procedural sequences where the task is to recover omitted steps. Under thinking mode, both domains improve, but the magnitude and likely mechanisms differ.

On GitHub PRs, thinking increases micro-F1 from 26.5% to 40.5% (a 53% relative gain). On Recipes, the corrected micro-F1 increases from 39.9% to 73.5% (an 84% relative gain). In Recipes, this larger gain is closely tied to the dataset's small step count (Section 3.1), which produces a substantial fraction of zero-omission cases, accounting for 39.8% (199/500) of the samples. In these cases, the improvement is driven primarily by instructional precision: in my runs, thinking mode produced an empty output in 199/199 zero-omission examples, whereas the baseline produced a non-empty, explanatory response in 199/199 cases (0/199 compliant). This is consistent with the interpretation that reasoning can act as a self-check that verifies the absence of targets before producing output, which is crucial under strict exact-match scoring.

Behavior Type	Github Prs Baseline	Github Prs Thinking	Recipes Baseline	Key Characteristics
Perfect Silence	0 (0.0%)	1 (0.7%)	6 (3.0%)	Full compliance with null output
Pure Explanation	39 (28.7%)	56 (41.2%)	187 (94.0%)	Natural language only (e.g., "No missing lines found")
Structural Noise	18 (13.2%)	18 (13.2%)	0 (0.0%)	Formatting debris (brackets, spaces)
Redundant Code	79 (58.1%)	61 (44.9%)	12 (6.0%)	Mistakenly retrieved existing lines

Table 3: Comparative distribution of model behaviors in zero-omission scenarios

A crucial observation from Table 3 is that, in zero-omission scenarios, the dominant failure modes differ by domain: Recipes is primarily limited by instructional compliance, whereas GitHub PRs more often exhibits logical identification errors. In the Recipes baseline, the model correctly recognizes that no steps are missing in 94.0% of cases (categorized as "Pure Explanation"), yet still produces a non-empty response, indicating difficulty adhering to the negative constraint of remaining silent. In other words, the failure mode is dominated by prompt non-compliance, consistent with a conversational-response bias.

In contrast, the GitHub PRs baseline exhibits a much higher rate of "Redundant Code" (58.1%), reflecting genuine retrieval errors where existing code is incorrectly emitted as missing. This points to a domain-level difference: natural-language procedural tasks (Recipes) primarily challenge self-restraint and instruction following, whereas code-dense environments (PRs) more often elicit active retrieval mistakes. Accordingly, the main contribution of thinking mode in Recipes is to enforce silence as a regulatory mechanism, while in PRs it functions more as a filter that reduces hallucinated code and shifts the model toward verified confirmation.

Behavior Type	Baseline F1	Thinking F1
Perfect Silence	0.0	100.0
Pure Explanation	100.0	100.0
Structural Noise	95.9	94.7
Redundant Code	74.9	82.9
Average Total F1	84.9	91.6

Table 4: PRs micro-F1 scores by error/behavior category under baseline (non-thinking) and thinking modes, reported as percentage-point values. Notably, even in zero-omission cases where the correct behavior is "Perfect Silence," PRs still attains high F1 under both modes, including when the model makes qualitatively severe mistakes such as producing Redundant Code. This reflects the PRs scoring regime's tolerance to redundant outputs in long-line settings.

Importantly, the magnitude of the baseline gap is also shaped by how the repository's evaluation script penalizes redundant output. In the GitHub PRs implementation, the instance-level score is computed approximately as $1 - \mathrm{FP/Total\_Lines}$. Because PR contexts often contain dozens to hundreds of lines, false positives introduced by Pure Explanation, Structural Noise, or Redundant Code are partially diluted by a large denominator, which reduces the apparent penalty for violating negative constraints. By contrast, Recipes lacks such a tolerance buffer: for zero-omission samples, any non-silent output (explanatory text, formatting noise, or repeated steps) constitutes a complete violation and collapses the micro-F1 score to 0. Therefore, the substantially lower baseline micro-F1 in Recipes is largely attributable to an evaluation regime that makes instructional non-compliance maximally visible, whereas similar verbosity can be partially diluted in PRs. This asymmetry suggests that higher baseline scores in PRs are not necessarily indicative of stronger negative-constraint adherence, but may partly reflect the metric's lower sensitivity to conversational redundancy in long-line settings.

3.3 Semantic Reconstruction vs. Verbatim Fidelity: Why Thinking Hurts Non-Zero-Omission Recipes

To isolate dataset-composition effects, I re-weighted Recipes so its zero-omission (ZO) rate matches the 15.3% observed in GitHub PRs. Under this composition-matched setting, the Recipes baseline rises to 56.1% and the thinking score drops to 62.7%, reducing the relative gain from 84.2% to 11.8% (now close to the 14.03 percentage-point gain observed for GitHub PRs under thinking mode). This suggests the original surge is partly driven by high ZO density: thinking reliably enforces silence on ZO cases but can hurt exact-match fidelity on non-zero instances via semantic reconstruction, whereas PRs gains appear less composition-dependent in this setup.

To ground this explanation, a closer inspection of non-zero-omission Recipes outputs corroborates the mechanism: thinking-mode errors are often driven by semantic reconstruction that breaks literal alignment under strict exact-match scoring. Across the 500 Recipes samples, I observe 37 cases (7.4%) where the thinking-mode response is semantically faithful yet fails exact match because the model reformats or re-segments the missing content. In contrast, such failures are rare in GitHub PRs, where syntax and formatting are rigidly constrained. Representative examples are shown in Table 5.

Sample ID	**Ground Truth (or Baseline Recovery)**	**Thinking Output (F1 = 0.0)**	**Failure Explanation**
ID 27	Heat the butter... Stir in the onion... Stir in flour and salt... (single line)	Heat the butter... / Stir in the onion... / Stir in flour and salt... (split into three lines)	Structural splitting: the model decomposes a compound instruction into multiple steps, breaking line alignment.
ID 38	Squeeze lime juice... Allow shaker to rest for 1 to 2 minutes. (continuous text)	Squeeze lime juice... Pour tequila... Cover shaker... Allow shaker to rest...	Listification: the model converts descriptive text into an explicit step list, changing formatting and content boundaries.
ID 105	Combine 2 teaspoons sugar and cinnamon in a small bowl; sprinkle evenly over muffin tops.	Combine 2 teaspoons sugar and cinnamon in a small bowl;/ sprinkle evenly over muffin tops.	Atomization: the model separates action nodes into finer-grained steps, altering the granularity of the recovered lines.

Table 5: Representative semantic reconstructions in Recipes that fail line-level exact match

The error accounting further clarifies how thinking improves PRs and why, within Recipes, its benefits are concentrated in zero-omission cases while non-zero-omission instances can degrade under strict exact match. In PRs, baseline outputs contain substantial irrelevant content (FP = 31,428), and thinking sharply reduces false positives to 12,924 (a 58.8% decrease), indicating that reasoning acts as a filter that suppresses spurious code and improves precision. In Recipes, the task is already low-noise (baseline FP = 1), so there is little room for “denoising” gains. Instead, thinking reduces true positives from 325 to 261 and increases false negatives, consistent with an “overthinking” pattern where the model optimizes for readability or procedural clarity rather than literal reproduction, which is penalized under exact match.

More broadly, these findings suggest that reasoning can improve internal reconstruction while worsening verbatim fidelity under strict exact-match scoring. In domains where evaluation requires line-level literal mirroring, reporting both strict EM and a normalization-aware variant may better separate semantic recovery from formatting-driven errors.

4 Conclusion

This report evaluates gemini-2.5-flash on AbsenceBench across four domains and yields three main findings about reasoning-enhanced LLM behavior.

First, I observe an output-capacity bottleneck in long-retrieval tasks. In the Poetry domain, the model exhibits a persistent response-length ceiling near the 100-line range, even with increased reasoning, which mechanically limits recall on large omission sets.

Second, the benefits of thinking mode are domain-dependent and sensitive to dataset composition. In Recipes, much of the aggregate gain is driven by improved instructional compliance in zero-omission cases, whereas in GitHub PRs reasoning primarily reduces spurious retrieval, lowering false positives in a code-dense setting.

Third, I find a trade-off between semantic reconstruction and verbatim fidelity under strict exact-match scoring. Reasoning can induce reformatting and re-segmentation that harms literal alignment, and evaluation design can further modulate these effects through denominator-driven tolerance in long-text settings versus strict penalties in short-text ones.

Overall, thinking mode tends to favor logical reconstruction over literal mirroring. Future replications and benchmarks would benefit from (i) explicitly reporting generation constraints and truncation rates and presenting length-stratified results by omission size, and (ii) reporting both strict exact-match and lightweight normalization-aware variants (e.g., whitespace and punctuation canonicalization) to separate semantic recovery from formatting-driven errors while still penalizing conversational redundancy.