SUPPLEMENTARY INFORMATION: THE LIMITS OF EXACT STATE TRACKING IN LANGUAGE MODELS

Tianxiang Dai
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
txdai@stanford.edu

Jonathan A. Fan
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
jonfan@stanford.edu

Supplementary Note 1: Supplementary Results Visualizations

This supplementary information file collects the supporting analysis and result-focused visualizations that accompany the main manuscript. It includes mathematical bounds for the adaptive ladder, a comprehensive audit of instruction-following collapse at the sequence boundary, and the overview figures used to summarize Stable Counting Capacity (SCC), token efficiency, context-length behavior, and family-level patterns. It also provides the full suite of benchmark correlation panels and adaptive search panels for representative closed and open-source models.

Supplementary Note 2: Prompt Templates and Generation Settings

The benchmark harness uses the following exact system instruction string:

```
You count items exactly. Return only one integer with no words, punctuation, or explanation.
```

The user prompt template is:

```
How many occurrences of "{item_label}" are in this sequence?

{sequence}
```

Here `{item_label}` is the counted symbol label (for example `a`) and `{sequence}` is the generated repeated-token sequence with the chosen delimiter. For Google models, the preferred path sends the system instruction separately as a developer/system instruction and sends the user string above as the sole content block. If that provider path is not available, the harness falls back to an inline Google prompt formed by concatenating:

```
{system_instruction}

{user_prompt}
```

Across all APIs, the benchmark used deterministic prompting with the fixed textual templates above. The shared harness settings were:

- RNG seed for the search procedure: `7`
- Initial output cap: `4096` tokens
- Retry output cap: `32768` tokens
- Preflight count: `8`
- Adaptive search parameters: initial length `32`, jitter fraction `0.2`, samples per length `16`, parallel trials per batch `15`, tolerance fraction `0.05`, step fraction `0.1`, maximum searched length `20000`

If a response terminated because of the output cap and still did not contain a parseable integer, the harness doubled the active token budget until either a parseable integer was produced or the retry ceiling of `32768` tokens was reached. This retry logic was identical across the provider-specific pathways above.

Supplementary Note 3: Mathematical Bounds on Guessing Exploits

A critical feature of the $\pm 20\%$ fuzzy boundary combined with the relative MAE metric is that it structurally prevents a model from artificially minimizing error via a generalized prior guess.

Suppose an architecture loses sequence cohesion but recognizes its approximate sequence depth. In an attempt to minimize error, it outputs a statistically optimal fixed guess c for the continuous range $X \sim \mathcal{U}(L(1-\alpha), L(1+\alpha))$. The expected Mean Absolute Percentage Error (MAPE) for a constant guess c over this uniformly distributed span is

$$\mathbb{E}[\text{MAPE}_{\text{guess}}] = \frac{1}{2\alpha L} \int_{L(1-\alpha)}^{L(1+\alpha)} \frac{|c-x|}{x} dx. \tag{S1}$$

The optimal guess c^* that minimizes this integral is the geometric mean of the distribution boundaries, $c^* = L\sqrt{1-\alpha^2}$. Substituting this optimal guess yields the absolute minimum theoretical expected relative error achievable without genuine token tracking:

$$\min_c \mathbb{E}[\text{MAPE}_{\text{guess}}] = \frac{1-\sqrt{1-\alpha^2}}{\alpha}. \tag{S2}$$

For our defined fuzzy boundary of $\alpha = 0.20$, this lower limit evaluates to $\approx 10.1\%$. Alternatively, if the model guesses the distribution center L, the expected error is slightly higher at $\approx 10.2\%$.

An architecture employing the optimal fixed-guess strategy over the $\mathcal{U}(0.8L, 1.2L)$ bounded uniform distribution yields an expected MAPE of $\mu \approx 10.10\%$ with a standard deviation of $\sigma \approx 5.86\%$. At a passing threshold of $\tau = 5\%$, evaluating only a single trial ($K = 1$) introduces a 24.6% probability that a guessing model randomly passes. By scaling the batch size to $K = 16$, the Z-score for a guessing model reaching the $\tau = 5\%$ threshold is -3.48. This establishes a strict false-positive rate of approximately 0.025%.

Supplementary Note 4: Audit of Instruction-Following Failures

In addition to the exact numerical failure modes discussed in the main text, models exceeding their stable sequence capacity frequently experience a severe collapse in instruction following. To systematically quantify this, we conducted an audit flagging every trial where the raw response violated the strict single-integer formatting constraint (evaluated via the regular expression `^\s*[+-]?\d+\s*$`). Cases were categorized from the raw response text, even when the benchmark parser managed to successfully extract a fallback integer from the sequence.

Out of 9,797 total evaluation trials scanned across the benchmark, 501 responses (5.1%) were flagged for containing extraneous text, formatting artifacts, or failing to produce a number entirely. Despite this instruction failure, the evaluation parser successfully extracted a valid integer in 305 of these cases (60.9%). This degradation broadly affected 44 distinct model variants, comprising 286 cases from closed-source APIs and 215 cases from open-weight models.

Supplementary Table 1 details the distribution of these failure modes. The most common deviation was unsolicited step-by-step counting or explanation, where models abandoned zero-shot generation and attempted to bypass internal state loss via explicit text generation. This was followed closely by completely blank or whitespace outputs. Supplementary Table 2 highlights the specific models most frequently exhibiting these deviations. Notably, log inspection reveals another sub-category among reasoning-optimized and coding models: exceeding the maximum output token limit (recorded in inference logs as `null failures`). When pushed beyond their stable capacity during the adaptive ladder expansion, these models tend to "overthink," falling into infinite latent reasoning loops without ever emitting a final parsed answer. For example, `gpt-5.2-codex` recorded 17 `null failures` (out of 129 trials) as the sequence expanded to a center length of L=256. Similar breakdowns occurred for `gpt-5.1-codex-mini` (7 failures at L=64), `gpt-5.1-codex-max` (2 failures at L=64), and `o1` (1 failure at L=128).

When models undergo structural state collapse, they frequently regress to pre-training priors or exhibit complete contextual disassociation. In severe cases, models not only fail to output the correct integer but hallucinate an entirely unrelated instruction sequence, or trap themselves in infinite reasoning loops.

For instance, following state loss, the `deepseek-r1` model occasionally defaulted to generating a `</think>` tag followed by an unprompted, hallucinated chain-of-thought trace attempting to solve unrelated calculus or algebra problems. Similarly, `gemini-3-flash-preview` attempted to write Python code to parse the string, and

Supplementary Table 1: **Category breakdown of non-single-number responses.**

Category	Cases	Share	Affected models	Integer still parsed
Step-by-step counting or explanation	266	53.1%	11	266
Blank / whitespace	141	28.1%	31	0
Token-limit exhaustion (Overthinking)	27	5.4%	4	0
Prompt echo / copied sequence	28	5.6%	2	1
Answer with extra prose / instruction leakage	20	4.0%	7	20
Code / markdown formatted output	10	2.0%	2	10
Reasoning-tag or hidden-CoT markup	7	1.4%	1	7
Text with no parseable number	1	0.2%	1	0
Number with stray punctuation / formatting	1	0.2%	1	1

Supplementary Table 2: **Models most frequently exhibiting instruction-following collapse.**

Model	Flagged cases	Integer still parsed	Dominant category
claude-sonnet-4-6	143	143	Step-by-step counting
claude-opus-4-6	73	73	Step-by-step counting
mistralai/devstral-2512	28	3	Prompt echo
claude-sonnet-4-5-20250929	26	26	Step-by-step counting
deepseek-v3.2-speciale	21	0	Blank / whitespace
deepseek-r1-distill-llama-70b	18	5	Blank / whitespace
gpt-5.2-codex	17	0	Token-limit exhaustion
mistral-7b-instruct-v0.1	17	17	Answer with extra prose
glm-5.1	17	1	Blank / whitespace
deepseek-r1	13	7	Hidden-CoT markup
mixtral-8x7b-instruct	12	12	Code / markdown output
qwen3-14b	10	0	Blank / whitespace
gemini-3-flash-preview	7	7	Code / markdown output
gpt-5.1-codex-mini	7	0	Token-limit exhaustion
deepseek-r1-distill-qwen-32b	7	0	Blank / whitespace

`mistralai/devstral-2512` simply echoed the infinite sequence indefinitely. Models like `gpt-5.2-codex` and `gpt-5.1-codex-mini` frequently encountered token-limit exhaustion, where their latent scratchpads filled with unknown traces until the inference API forcibly terminated the generation. Representative snippets are detailed in Supplementary Table 3.

Supplementary Note 5: Full Model Catalogue

Supplementary Table 4 lists the full model catalogue together with the corresponding developer or family, parameter metadata, nominal context window, architectural class, and measured Stable Counting Capacity (SCC).

Supplementary Table 4: **Full model catalogue.** Nominal context window and parameter metadata are taken from the run manifests when available. The architecture label is inferred from published parameter metadata: models with distinct total and active parameter counts are marked as MoE; models with only a single parameter count are marked as Dense; unavailable cases are left as "–".

Model Name	Developer/Family	Parameter Count	Nominal Window Context	Architecture	Measured SCC
claude-4-opus [**?**]	Anthropic	–	–	–	2176
claude-4.1-opus [**?**]	Anthropic	–	–	–	1920
claude-4.5-opus [**?**]	Anthropic	–	–	–	1536
claude-4.5-sonnet [**?**]	Anthropic	–	–	–	896
claude-4.6-opus [**?**]	Anthropic	–	–	–	768
gpt-5.4 [**?**]	OpenAI	–	–	–	608
claude-4-sonnet [**?**]	Anthropic	–	–	–	544

Continued on next page

Model Name	Developer/Family	Parameter Count	Nominal Window	Context	Architecture	Measured SCC
claude-4.6-sonnet [?]	Anthropic	–	–		–	512
claude-3.7-sonnet [?]	Anthropic	–	200,000		–	352
claude-3.7-sonnet (thinking) [?]	Anthropic	–	200,000		–	352
gemini-3-pro-preview [?]	Google	–	–		–	352
gemini-3.1-pro-preview [?]	Google	–	–		–	352
gemini-3-flash-preview [?]	Google	–	–		–	304
gpt-5.3-codex [?]	OpenAI	–	–		–	272
claude-4.5-haiku [?]	Anthropic	–	–		–	176
gpt-5.2 [?]	OpenAI	–	–		–	176
claude-3.5-haiku [?]	Anthropic	–	200,000		–	144
gpt-5.4-mini [?]	OpenAI	–	–		–	144
gemini-2.5-pro [?]	Google	–	–		–	136
gemini-3.1-flash-lite-preview [?]	Google	–	–		–	128
gpt-5.2-codex [?]	OpenAI	–	–		–	128
gpt-5.1 [?]	OpenAI	–	–		–	120
kimi-k2-0905 [?]	Moonshot	–	262,144		MoE	112
gpt-5 [?]	OpenAI	–	–		–	112
llama-4-maverick [?]	Meta	17B	1,048,576		MoE	96
gpt-4.1 [?]	OpenAI	–	–		–	96
gemini-2.0-flash-001 [?]	Google	–	1,048,576		–	88
kimi-k2 [?]	Moonshot	32B active	131,072		MoE	88
kimi-k2-thinking [?]	Moonshot	–	262,144		MoE	88
gpt-4o [?]	OpenAI	–	–		–	88
o3 [?]	OpenAI	–	–		–	88
o1 [?]	OpenAI	–	–		–	80
qwen3.5-397b-a17b [?]	Qwen	397B total / 17B active	262,144		MoE	80
deepseek-r1-0528 [?]	DeepSeek	671B total / 37B active	163,840		MoE	72
gemini-2.0-flash-lite-001 [?]	Google	–	1,048,576		–	72
gemini-2.5-flash [?]	Google	–	–		–	72
gpt-4.1-mini [?]	OpenAI	–	–		–	72
qwen3-235b-a22b-2507 [?]	Qwen	235B total / 22B active	262,144		MoE	68
gemma-4-31b-it [?]	Google	31B	–		Dense	64
gpt-5-mini [?]	OpenAI	–	–		–	64
glm-5.1 [?]	Z.ai	–	202,752		–	64
glm-4.5 [?]	Z.ai	–	131,072		MoE	60
claude-3-haiku [?]	Anthropic	–	–		–	56
deepseek-chat-v3-0324 [?]	DeepSeek	685B	163,840		MoE	56
gemini-2.5-flash-lite [?]	Google	–	–		–	56
gemma-4-26b-a4b-it [?]	Google	26B total / 4B active	–		MoE	56
minimax-01 [?]	MiniMax	456B	1,000,192		Dense	56
minimax-m2.1 [?]	MiniMax	10B active	196,608		Dense	52
mistral-small-2603 [?]	Mistral	119B	262,144		Dense	52
mixtral-8x22b-instruct [?]	Mistral	176B total / 39B active	65,536		MoE	52
kimi-k2.5 [?]	Moonshot	–	262,144		–	52
gpt-4.1-nano [?]	OpenAI	–	–		–	52
qwen3-235b-a22b-thinking-2507 [?]	Qwen	235B total / 22B active	131,072		MoE	52
qwen3-next-80b-a3b-thinking [?]	Qwen	80B total / 3B active	131,072		MoE	52
qwen3.5-122b-a10b [?]	Qwen	122B total / 10B active	262,144		MoE	52
gpt-5-nano [?]	OpenAI	–	–		–	48
gpt-5.4-nano [?]	OpenAI	–	–		–	48
o3-mini [?]	OpenAI	–	–		–	48
qwen3-30b-a3b-instruct-2507 [?]	Qwen	30B total / 3B active	262,144		MoE	48
qwen3-next-80b-a3b-instruct [?]	Qwen	80B total / 3B active	262,144		MoE	48
glm-4.7 [?]	Z.ai	–	202,752		–	48
gpt-5.1-codex-max [?]	OpenAI	–	–		–	44
qwen-2.5-72b-instruct [?]	Qwen	72B	32,768		Dense	44
qwen3-235b-a22b [?]	Qwen	235B total / 22B active	131,072		MoE	44
qwen3-30b-a3b [?]	Qwen	30B total / 3B active	40,960		MoE	44
glm-4.5-air [?]	Z.ai	–	131,072		MoE	44
minimax-m2 [?]	MiniMax	10B active	196,608		Dense	40
devstral-2512 [?]	Mistral	123B	262,144		Dense	40
gpt-4 [?]	OpenAI	–	–		–	40
gpt-5.1-codex-mini [?]	OpenAI	–	–		–	40
qwen3-coder-30b-a3b-instruct [?]	Qwen	30B total / 3B active	160,000		MoE	40
qwen3-coder-next [?]	Qwen	80B total / 3B active	262,144		MoE	40
qwen3.5-27b [?]	Qwen	27B	262,144		Dense	40
gemma-3-27b-it [?]	Google	27B	–		Dense	36
llama-3.1-70b-instruct [?]	Meta	70B	131,072		Dense	36
minimax-m2.5 [?]	MiniMax	–	196,608		–	36
mistral-small-24b-instruct-2501 [?]	Mistral	24B	32,768		Dense	36
gpt-4-turbo [?]	OpenAI	–	–		–	36
gpt-4o-mini [?]	OpenAI	–	–		–	36

Continued on next page

Model Name	Developer/Family	Parameter Count	Nominal Window	Context	Architecture	Measured SCC
glm-4.6 [?]	Z.ai	–	204,800		–	36
glm-5 [?]	Z.ai	–	202,752		–	36
deepseek-r1 [?]	DeepSeek	671B total / 37B active	64,000		MoE	32
deepseek-r1-distill-llama-70b [?]	DeepSeek	70B	131,072		Dense	32
deepseek-v3.2-speciale [?]	DeepSeek	–	163,840		–	32
gemma-3-12b-it [?]	Google	12B	–		Dense	32
gemma-3-12b-it [?]	Google	12B	131,072		Dense	32
llama-3.3-70b-instruct [?]	Meta	70B	131,072		Dense	32
minimax-m2.7 [?]	MiniMax	–	196,608		–	32
devstral-small [?]	Mistral	24B	131,072		Dense	32
ministral-14b-2512 [?]	Mistral	14B	262,144		Dense	32
mistral-small-3.2-24b-instruct [?]	Mistral	24B	128,000		Dense	32
gpt-3.5-turbo [?]	OpenAI	–	–		–	32
qwen3-14b [?]	Qwen	14.8B	40,960		Dense	32
qwen3-32b [?]	Qwen	32.8B	40,960		Dense	32
qwen3.5-35b-a3b [?]	Qwen	35B total / 3B active	262,144		MoE	32
qwq-32b [?]	Qwen	32B	131,072		Dense	32
deepseek-chat [?]	DeepSeek	–	163,840		–	0
deepseek-chat-v3.1 [?]	DeepSeek	671B total / 37B active	32,768		MoE	0
deepseek-r1-distill-qwen-32b [?]	DeepSeek	32B	32,768		Dense	0
deepseek-v3.1-terminus [?]	DeepSeek	–	163,840		–	0
deepseek-v3.2 [?]	DeepSeek	–	163,840		–	0
deepseek-v3.2-exp [?]	DeepSeek	–	163,840		–	0
gemma-2-27b-it [?]	Google	27B	8,192		Dense	0
gemma-3-27b-it [?]	Google	27B	131,072		Dense	0
gemma-3-4b-it [?]	Google	4B	131,072		Dense	0
gemma-3n-e4b-it [?]	Google	4B	32,768		Dense	0
llama-3-70b-instruct [?]	Meta	70B	8,192		Dense	0
llama-3-8b-instruct [?]	Meta	8B	8,192		Dense	0
llama-3.1-8b-instruct [?]	Meta	8B	16,384		Dense	0
llama-3.2-3b-instruct [?]	Meta	3B	80,000		Dense	0
llama-4-scout [?]	Meta	17B	327,680		MoE	0
phi-4 [?]	Microsoft	14B	16,384		Dense	0
ministral-3b-2512 [?]	Mistral	3B	131,072		Dense	0
ministral-8b-2512 [?]	Mistral	8B	262,144		Dense	0
mistral-7b-instruct-v0.1 [?]	Mistral	7.30BB	2,824		Dense	0
mistral-nemo [?]	Mistral	12B	131,072		Dense	0
mistral-small-3.1-24b-instruct [?]	Mistral	24B	128,000		Dense	0
mixtral-8x7b-instruct [?]	Mistral	56B total / 14B active	32,768		MoE	0
voxtral-small-24b-2507 [?]	Mistral	24B	32,000		Dense	0
qwen-2.5-7b-instruct [?]	Qwen	7B	32,768		Dense	0
qwen-2.5-coder-32b-instruct [?]	Qwen	32B	32,768		Dense	0
qwen3-30b-a3b-thinking-2507 [?]	Qwen	30B total / 3B active	131,072		MoE	0
qwen3-8b [?]	Qwen	8.20BB	40,960		Dense	0
qwen3-coder [?]	Qwen	480B total / 35B active	262,144		MoE	0
qwen3.5-9b [?]	Qwen	9B	256,000		Dense	0
glm-4.7-flash [?]	Z.ai	30B	202,752		Dense	0

Supplementary Note 6: Heterogeneous Nested Key-Path Tracking Assay

To explicitly test whether the collapse of structural state maintenance extends beyond one-dimensional linear accumulation and zero-entropy prompts, we introduced a heterogeneous nested-tracking assay. This task evaluates a model's capacity for hierarchical depth-tracking and multi-slot variable binding across sequences populated with diverse semantic distractors, testing the generalization of "higher-layer abstraction".

Task Design

Models are presented with a sequence of syntactically complex records. Each record contains a `KEY` token, a deeply nested `PATH` field utilizing alternating bracket types alongside randomly generated distractor filler tokens, and a `SIDE` field containing two random tokens. All tokens are drawn from a fixed pool of twenty 2-letter capitalized strings (e.g., `BX`, `CU`, `TV`). A record is defined as a "match" if and only if the `KEY` token is identical to the deepest leaf token nested inside the `PATH` brackets.

The system prompt and a representative snippet are as follows:

```
Each record has a KEY token and a nested PATH field. A record counts as a
match when the KEY token is identical to the deepest token inside PATH. Count
```

Supplementary Table 3: **Representative examples of non-single-number responses.** For brevity and formatting, lengthy outputs have been truncated, and non-Latin scripts (e.g., Hindi, Chinese) generated during instruction leakage have been translated and bracketed.

Category	Model	Parsed	Raw Response Snippet
Step-by-step counting	claude-opus-4-6	415	I need to count all the "a" items in this sequence. Let me count by groups of 10: Row 1: a, a, a...
	claude-sonnet-4-5	25	I'll count each "a" in the sequence: a, a, a, a, a, a... 25
Blank / whitespace	deepseek-r1	–	[Empty response / whitespace only]
	deepseek-v3.2	–	[Empty response / whitespace only]
Prompt echo	devstral-2512	–	a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a...
Extra prose / leakage	gemini-2.5-flash	313	[Output in Hindi: The number of "a" characters in this sequence is 313.]
	minimax-m2.7	32	32 Be sure to dominate your code.
Code / markdown	mixtral-8x7b-instruct	30	30 ‘‘‘`python` ewline `s = "a, a, a..."` ewline `print(len(s.split(',')))` ewline ‘‘‘
	gemini-3-flash	20	Wait, the text *doesn't* end there in my count...
Reasoning-tag markup	deepseek-r1	2	`</think>` [Output in Chinese: Okay, let's reason step-by-step. 1. Understand the problem: There are n boxes...]
Stray punctuation	gemini-3-flash	500	500$

```
how many matching records are in the following sequence. Return only one integer.

R000{KEY=BX; PATH=[FW<{TV}>GY]; SIDE=(CU|DV)} ; R001{KEY=LM; PATH=(HZ[<LM>]JQ); SIDE=(WX|YZ)} ; ...
```

Because the sequence is interspersed with random two-letter capitalized distractor tokens and varied bracket geometries (which cycle sequentially between `[{left}<{leaf}>{right}]`, `({left}[<{leaf}>]{right})`, and `<{left}({leaf}){right}>`), the model cannot rely on pure 1D token distances or homogeneous attention sink dynamics. To successfully track the count, the network must correctly parse the hierarchical nesting in parallel, evaluate the conditional equality for each record, and mathematically synthesize the cumulative sum of true matches over the entire sequence context.

Evaluation Constraints

We quantified structural stability on this task using the identical adaptive randomized ladder methodology utilized for the linear counting assay. However, to maintain logical constraints, the center sequence length L dictates the *target number of true matches* rather than the overall number of sequence records.

For each evaluation tier L, we sample target match counts $x_k \sim U(0.8L, 1.2L)$. To guarantee the sampled target never mathematically exceeds the length of the prompt, the evaluation system dynamically populates the sequence with exactly x_k matching records, and then appends a proportional volume of negative (non-matching) distractor records. Specifically, the amount of negative records is scaled as $\max(2, \lceil 0.5 \times x_k \rceil + 2)$. The positive and negative records are then randomly shuffled.

Results

Evaluating a cohort of frontier API models on this nested task (initiating at a minimal target length of $L = 32$ matches), we found that the performance boundary collapses immediately for the vast majority of architectures. If a model fails to satisfy the accuracy threshold at the minimal initial tier, it lacks a valid stable lower bound to interpolate; therefore, we bypass the binary search and record its Stable Counting Capacity (SCC) categorically as < 32.

As shown in Supplementary Table 5, the added computational burden of structural variable binding ablates the state maintenance capacity previously observed in the linear homogeneous assay. These results validate that the abrupt

capability cliff represents a generalized fragility in higher-layer abstraction rather than an anomaly specific to simple token counting. Scatter plots are shown in Supplementary Figure 12.

Supplementary Table 5: **Stable Counting Capacity (SCC) on the Heterogeneous Nested Key-Path Match Assay.** The parameter L denotes the target number of true matches. Models failing to accurately compute the matches at the minimal initial search tier ($L = 32$) are recorded with an SCC of < 32.

Model	Stable Count (SCC)	Stable Lower Bound	Unstable Upper Bound
gpt-5.4	416	416	448
gpt-5.4-mini	< 32	< 32	32
gpt-5.2	< 32	< 32	32
gpt-5.1	< 32	< 32	32
gpt-4o	< 32	< 32	32
gpt-4o-mini	< 32	< 32	32
gpt-4.1	< 32	< 32	32
gpt-4.1-mini	< 32	< 32	32

Supplementary Note 7: Anatomy of Latent Reasoning Breakdowns

To definitively verify that token-limit exhaustion and massive token expenditures in reasoning-optimized architectures constitute genuine state collapse rather than the artificial truncation of a valid algorithm, we extracted the hidden reasoning traces (e.g., contents within <think> tags) from evaluated open-weight models. Because the target sequence lengths evaluated near the stable boundary rarely exceed $L = 500$, a 2:1 or 3:1 token-to-count mapping would comfortably fit within a small fraction of the 32,768 retry token limit. Therefore, extreme token usage serves as a highly reliable diagnostic signature of structural hallucination.When the latent progress variable degrades, reasoning models exhibit distinct failure modes, ranging from infinite generative traps to severe over-complication of trivial tracking variables. Below are two representative examples extracted from the evaluation logs:

Deepseek R1-0528 on length=96 expresses significant overthinking (1915 reasoning tokens), even though it provided the correct answer:

```
First, the user has given me a sequence that consists entirely of the letter "a" repeated
multiple times.  I need to count how many times "a" appears in this sequence.

The sequence is written as:  "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a"

I should count the number of "a"s.  Since it's a sequence separated by commas, each "a"
is an individual element, so I need to count how many times "a" is listed.

One way to do this is to count the number of commas and add one, because each comma
separates two elements, so the number of elements is one more than the number of commas.

Let me look at the sequence.  It starts with "a" and ends with "a", and there are commas
between them.

I could also copy the sequence and count the characters, but since it's all "a"s with
commas, I need to be careful not to count the commas.

A better approach might be to count the number of "a"s directly.

Since the sequence is long, I can group them or find a pattern.

Let me see how many "a"s are in each line or something, but it's given as a single string
in the message.
```

In the user's message, it's written as a continuous string with commas. Let me copy it here to work with it.

The sequence is: a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a

Now, to count, I can think of this as a string and count the occurrences of the substring "a".

But in the string, there are commas and spaces, so I need to count only the "a"s, not the punctuation.

Since each "a" is separated by a comma and a space, I can count the number of such separations.

Let me count how many times the pattern "a, " appears, but that might miss the last one or something.

A standard way is to split the string by commas and then count the number of elements, each of which should be "a".

I can do that in my reasoning.

Since the model supports reasoning traces, I should show my steps.

Let me imagine splitting the sequence by commas.

The sequence has commas between each "a", so if I remove all commas and spaces, I should have a string of "a"s, and the length of that string is the count.

But there are spaces after commas? In the sequence, it's written with commas and spaces, like "a, a, a", so between each "a" there is ", " which is comma and space.

In the user's message, it's "a, a, a, ..." so yes, there is a space after each comma except possibly the last one, but since it ends with "a", no comma after last.

So, to count the number of "a"s, I can count the number of commas and add one.

Let me count the number of commas first.

I can look at the sequence and count how many commas there are.

Since it's repetitive, I can count in groups.

For example, let me see if I can find how many "a"s are listed before a line break or something, but in the text, it's all in one line in the message, I think.

In the user input, it's wrapped, but I should consider it as a single string.

Perhaps I can copy the string and use a method to count.

But since I'm an AI, I can process it textually.

Let me write the sequence as a string: "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a"

Now, to count the "a"s, I can split this string by ", " which is comma and space.

If I split by ", ", each element should be "a", and the number of elements is the count.

Let me think: for example, "a, a, a" split by ", " gives ["a", "a", "a"], which has 3 elements.

Similarly, if it ends with "a", no issue.

In this case, the string ends with "a", so splitting by ", " should give a list of "a"s.

The last part is "a, a" but since it ends with "a", when I split "a, a" by ", ", I get ["a", "a"].

Yes.

So, for the whole sequence, I can count how many times ", " appears, and add one for the last "a".

Number of separators is number of ", " which is one less than the number of elements.

Easier to split conceptually.

Since all elements are "a", I just need the count.

I can count the number of commas, as each comma corresponds to one separator.

In the string, each comma is followed by a space, but for counting, the number of commas equals the number of separators between "a"s.

For n "a"s, there are (n-1) commas.

So, if I find the number of commas, I can add one to get n.

Similarly, I can count the number of "a"s directly.

Let me try to count the commas.

Looking at the sequence, I can group the "a"s.

For example, I see that the sequence is written with multiple "a"s in a row, but in the text, it’s continuous.

I could count the characters.

The string has "a" followed by ", " for each additional "a", except the last one has no comma.

So, the total length of the string can be used, but I need to know how many "a"s and how many ", " pairs.

Each "a" takes one character, and each ", " takes two characters (comma and space), but for the first "a", no prefix, and for the last "a", no suffix.

Let k be the number of "a"s.

Then, the total number of characters in the string is: for the first "a": 1 char, then for each subsequent "a", there is ", " (2 chars) plus "a" (1 char), so per additional

"a", 3 chars are added.

Let's think:

- Start with one "a": length 1

- Add second "a": add ", a" which is 3 characters, total length 4 for two "a"s: "a, a" is 4 characters? "a" is 1, ", " is 2, "a" is 1, but "a, a" is positions: a , space a, so 4 characters.

"a, a" has 4 characters: index 0: 'a', 1: ',', 2: ' ', 3: 'a'

Similarly, for three "a"s: "a, a, a" has characters: a, comma, space, a, comma, space, a 2014 that's 7 characters.

Generally, for k "a"s, the string has:

- k occurrences of 'a'

- (k-1) occurrences of ','

- (k-1) occurrences of ' ' (space)

So total characters: k + (k-1) + (k-1) = k + k - 1 + k - 1 = 3k - 2

Total characters = number of 'a' + number of commas + number of spaces.

Since each comma is followed by a space, number of commas = number of spaces = (k-1)

So total characters = k + (k-1) + (k-1) = 3k - 2

Therefore, if I can find the total number of characters in the sequence string, I can solve for k.

But I need the actual string.

In the user's message, the sequence is given, and I can count the characters.

Since it's text, I can copy it and count.

But as an AI, I can process the input string.

Let me look at the input: "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a"

Now, to count the characters, I can see that it starts from the first "a" and ends with the last "a", and there are multiple groups.

I could count how many "a"s are listed by noticing the pattern.

Another idea: the sequence is written with "a, " repeated, but the last one is just "a" without comma.

In the string, "a, " appears multiple times, and then ends with "a".

So, if I count how many times the substring "a, " appears, that will give me the number of "a"s that are followed by a comma, which is all except the last one.

Then, I add one for the last "a".

For example, in "a, a, a", "a, " appears twice (for the first and second "a"), and then the third "a" is separate.

In "a, a, a", the substrings are: positions, but "a, " as a substring occurs at index 0 and index 3 if we consider the string.

"a, a, a" - length 7, "a, " at index 0: "a, " and at index 3: "a, " but index 3 is the second "a" with comma, but for the second "a", it is part of ", a" but "a, " is the "a" followed by comma and space.

In "a, a, a", the first "a, " is from index 0 to 2: characters 'a', ',', ' '

Then from index 3 to 5: 'a', ',', ' ' but that's the second "a, ".

Then index 6 is 'a', no comma.

So "a, " appears twice, for the first two "a"s, and there is one more "a" at the end.

So number of "a, " occurrences is k-1 for k "a"s.

Then k = number of "a, " + 1

Similarly, in this case.

But I need to count how many times "a, " appears in the string.

Since the string is long, I can count the total number of characters or find another way.

Perhaps I can see that the sequence is given in the message, and I can count the commas.

Let me count the commas in the sequence.

I can break it into smaller parts.

For example, in the sequence, it seems like there are multiple "a, " groups.

I can count how many "a"s are in each line of the user's message, but the message might be wrapped differently for different users, so I should not rely on that.

In the user input, it's presented as a single block, but in text, it's wrapped.

To be precise, I should consider the string as given.

Let me copy the string and count the commas.

The sequence: let's list it with indices or something.

I can use the fact that the sequence has 96 "a"s or something? I think I need to count.

Another thought: the user might have intended a certain number, but I should count from what's given.

Perhaps I can count the number of "a"s by splitting the string in my mind.

Let me imagine the string: it starts with "a, " repeated many times, and ends with "a".

Now, I can count how many times ", " appears.

Each ", " separates two "a"s, so number of ", " is one less than number of "a"s.

So if I find the number of ", " substrings, that will be the number of separators.

Then number of "a"s = number of separators + 1.

Similarly, I can count the number of commas, since each comma has a space after, but counting ", " is fine.

Let me try to count the occurrences of ", " in the string.

I can look for the pattern.

Since the sequence is uniform, I can count the total length.

Let me estimate the length of the string.

The sequence is: from the beginning to end.

I can count the number of "a"s by noticing that in the text, it's written with 16 "a"s per row or something, but I don't know the formatting.

In the user's message, as I see it, it's displayed with multiple "a, " per line, but for counting, I should ignore line breaks.

So, let's consider the raw string.

Perhaps I can use Python or something, but since I'm reasoning, I'll do it manually.

Let me count the commas.

I'll go through the sequence and count how many commas there are.

Start from the left:

First, there is a group of "a, " but let's count commas.

Position: after each "a" except the last, there is a comma.

So, I can count the commas by scanning.

I see that the sequence has commas between each pair.

To make it easier, I can count how many "a, " are there.

Each "a, " includes one "a" and the separator for the next, but it's messy.

Let's think of the string as a list.

I can split the string by ", " and the number of parts is the number of "a"s.

For example, "a, a, a" split by ", " gives ["a", "a", "a"], 3 parts.

Similarly, for the whole string.

So, if I can find how many elements when split by ", ", that's the count.

Now, to do that, I need the string.

Let me write the string: "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a"

Now, I can count the number of times "a, " appears, but "a, " is not at the end.

The string ends with "a", so "a, " appears for all but the last "a".

So, number of "a, " occurrences is equal to the number of "a"s that have another "a"
after, which is total "a"s minus one.

Then, the last "a" is alone.

But to find the number, I can count the occurrences of "a, ".

Each "a, " is a substring of three characters: 'a', ',', ' '.

In the string, "a, " appears multiple times.

Let me count how many times "a, " is present.

I can count the number of commas, as each comma is preceded by "a" and followed by space,
but for counting, number of commas is fine.

Let's count the commas in the sequence.

I'll go through the string mentally.

First part: "a, a, a, a, a, a, a, a, a, a," that's 10 "a"s, so 9 commas? Let's see:
"a, a, a, a, a, a, a, a, a, a" Ž2014 from start to here, there are 10 "a"s and 9 commas.

But in the sequence, it's continuous.

Perhaps I can count in groups of 10 or something.

Let's look at the sequence as written.

In the user's message, it starts with "a, a, a, a, a, a, a, a, a, a," which is 10 "a"s,
but then it continues.

After that, it says " a, a, a, a, a, a, a, a, a, a," but there might be spaces or not,
but in terms of elements, it's the same.

To be precise, let's count the total number of "a" entries.

I can count the number of commas and add one.

Let me count the commas.

I'll imagine the string and count every comma I see.

Start: position after first "a": comma, then space, then "a", then comma, etc.

But I can count the commas only.

From the beginning: comma after first a, after second, and so on.

Since it’s all the same, I can find the total length.

Let me copy the string and count the characters.

The string is: let’s take it as given.

I can see that the sequence has 96 "a"s because it’s written with 16 groups of 6 or something, but I need to verify.

Perhaps the user copied and pasted a fixed number.

But I should count from the input.

Another idea: in the message, the sequence is from "a, a, a" to the end, and it has a certain number.

Let’s count the number of "a"s by splitting.

I can use the fact that the string has a pattern.

Let me write the sequence without spaces for counting, but that might not help.

I can count the number of times the comma appears.

Let’s do it systematically.

I’ll count the commas in the first few "a"s to find a pattern.

The sequence begins: "a, a, a, a, a, a, a, a, a, a," that’s 10 "a"s with 9 commas? No.

"a, a, a, a, a, a, a, a, a, a" ˘2014 let’s count the elements: 1.a, 2.a, 3.a, 4.a, 5.a, 6.a, 7.a, 8.a, 9.a, 10.a ˘2014 so 10 "a"s, and between them, 9 commas.

But in the string, it’s "a, a, a, ..." so for 10 "a"s, there are 9 ", " pairs, but each ", " includes the comma and space.

For counting commas, there are 9 commas for 10 "a"s.

Then the sequence continues: after the first 10, it says " a, a, a, a, a, a, a, a, a, a," but there is a space before "a" because of the comma from previous, but it’s the next "a".

In the full string, it’s continuous, so after the first 10, there is a comma from the 10th? Let’s see the text.

The user wrote: "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a"

Now, to count, I can count how many "a"s there are by noticing that it is listed with no other characters, so each "a" is separate.

I can count the number of items in the list.

Since it’s a sequence of "a" separated by commas, the number of "a"s is equal to the number of commas plus one.

So, let’s count the commas.

I can count the commas in groups.

For example, in the string, I see that there are many "a, " which I can count.

Each "a, " corresponds to one "a" except the last one.

But let’s count the occurrences of "a, ".

The substring "a, " appears multiple times.

In the sequence, "a, " is repeated.

Let me count how many times "a, " appears.

From the start: "a, " (1), then "a, " (2), and so on.

But at the end, it’s "a" without ", ".

So, for example, if there are m occurrences of "a, ", then there are m+1 "a"s, because the last "a" is not followed by ", ".

In the string, "a, " appears for the first to the second-last "a", so number of "a, " is k-1 for k "a"s.

Then k = number of "a, " + 1

So I need to count how many times the substring "a, " appears in the string.

Let’s do that.

I can look at the string and count the "a, " instances.

Since the string is long, I can count in sections.

First, let’s take the beginning: "a, a, a, a, a, a, a, a, a, a," but this is 10 "a"s, but "a, " appears 9 times for the first 9, and the 10th is "a" with comma? No.

In "a, a, a, a, a, a, a, a, a, a" the last part is "a" without comma, but in the string it’s "a, a, a, a, a, a, a, a, a, a" so for each "a, " except the last "a" is not followed by ", ".

In this substring of 10 "a"s, it is "a, a, a, a, a, a, a, a, a, a" which can be seen as 9 "a, " and one "a" at the end.

But "a, " is a substring that appears 9 times in this 10-"a" group.

Similarly, in the full sequence.

But the full sequence has more.

Let’s count the total number of "a, ".

I can count the number of commas, as each comma is part of a ", " after an "a".

Each comma indicates one "a, " except for the last "a" which has no comma.

Number of commas equals number of "a, " because each comma is preceded by "a" and

followed by space, so "a, " is present for each comma.

For each comma, there is an "a, " substring ending at that point.

For example, in "a, a, a", there are two commas, and "a, " at index 0 and index 3.

So number of commas = number of "a, " occurrences.

Then, for k "a"s, number of commas = k-1

Number of "a, " = k-1

Then k = number of commas + 1

Or k = number of "a, " + 1

So I can count either.

Let's count the commas in the sequence.

I'll go through the string and count every comma.

Start from the beginning:

After the first "a", there is a comma (1)

Then after second "a", comma (2)

Third, comma (3)

Fourth, comma (4)

Fifth, comma (5)

Sixth, comma (6)

Seventh, comma (7)

Eighth, comma (8)

Ninth, comma (9)

Tenth, comma (10) Ž014 but for the tenth "a", in the first group, it might be different, but in the string, it's continuous.

In the sequence, the first few are: "a, a, a, a, a, a, a, a, a, a," so after the 10th "a", there is a comma, because it says "a, a, a, a, a, a, a, a, a, a," and then it continues with " a, a, ..." so after the 10th "a", there is a comma, and then space, then next "a".

So for the 10th "a", there is a comma after it, because it is not the last.

In fact, in the entire sequence, only the last "a" has no comma after it; all others have a comma.

So, to count commas, I can count all commas in the string.

Let's count the commas.

I can group the sequence into lines or something, but since it’s wrapped, I’ll count based on the text.

From the user’s message, I can see that it is written with 16 "a"s per line or something, but let’s not assume.

I can count the total number of characters.

Let me assume the string is as given.

I can count the number of "a"s by splitting the string in my mind.

The sequence has "a" followed by ", a" repeated.

Each ", a" adds one more "a".

But the first "a" is alone.

So, the string is: "a" + (", a" repeated (k-1) times)

For k "a"s.

In this case, the string starts with "a" and then has multiple ", a" for the subsequent ones.

In the string, it is "a, a, a, ..." so it can be seen as "a" followed by (k-1) times of ", a".

But ", a" is three characters: comma, space, a.

Then the total string is "a" + (k-1) * ", a"

Length: 1 + 3*(k-1) = 3k - 3 +1 = 3k -2, same as before.

But to find k, I need the length.

Let’s count the length of the string.

The sequence string: let’s count the characters.

I can count the number of commas and add one.

Let’s count the commas.

I’ll count in groups of 10 for simplicity.

First 10 "a"s: "a, a, a, a, a, a, a, a, a, a" but this has 10 "a"s and 9 commas? No.

"a, a, a, a, a, a, a, a, a, a" Ž014 let’s write it out: characters: a , space a , space a , ... but for 10 "a"s, it is: a, , , a, , , a, , , ... but each pair "a, " except the last.

For n "a"s in a row with separators, the string for n items has (n-1) commas and (n-1) spaces.

For example, n=10: "a, a, a, a, a, a, a, a, a, a" has 10 "a"s, 9 commas, 9 spaces.

Similarly, in the full sequence, it continues.

After that, the next part is " a, a, a, a, a, a, a, a, a, a," but there is a space because of the previous comma, but it's the next "a".

In the full string, after the first 10 "a"s, there is a comma (from the 10th "a"), then space, then "a" for the 11th, and so on.

So for the full count, let's list the commas.

I can count how many times ", " appears, as each ", " corresponds to one separator between "a"s.

Number of ", " is equal to the number of intervals between "a"s, which is k-1 for k "a"s.

Then k = number of ", " + 1

So let's count the number of ", " substrings in the string.

Each ", " is two characters: comma and space.

In the string, ", " appears between each pair of "a"s.

For example, in "a, a, a", ", " appears twice.

Similarly here.

So for the sequence, let's count the number of ", ".

I can do it by sections.

Look at the sequence: it starts with "a, a, a, a, a, a, a, a, a, a," which has 9 ", " because there are 10 "a"s, 9 intervals.

Then it continues with " a, a, a, a, a, a, a, a, a, a," but " a" has a space, but the ", " is already counted or not.

In the string, after the first 10 "a"s, it is "...a, a, a,..." so the comma and space after the 10th "a" is part of the ", " for the interval between 10th and 11th.

Then for the next 10 "a"s, from 11 to 20, there are 9 ", " between them, but the first ", " for 10-11 is already included.

Let's think of the whole string.

The string has a sequence of "a" separated by ", ".

So the number of ", " is the number of times it appears.

From the beginning: the first ", " after first a, second ", " after second a, and so on, up to the second last a.

For k "a"s, there are (k-1) ", ".

In the text, I can count how many times ", " occurs.

Since the sequence is long, I can count the total number of characters and solve.

Let me count the characters in the string.

The string is: let's take it from the user input.

I can see that in the message, the sequence has 96 "a"s because it’s 6 lines with 16 "a"s each or something, but let’s verify.

Perhaps I can copy the text and count, but as AI, I’ll estimate.

Another idea: the user said "this sequence" and it’s given in the prompt, so I can count the items.

Let’s list the "a"s with numbers.

1. a

2. a

3. a

4. a

5. a

6. a

7. a

8. a

9. a

10. a (first 10)

11. a

12. a

13. a

14. a

15. a

16. a

17. a

18. a

19. a

20. a (second 10)

21. a

22. a

23. a

24. a

25. a

26. a

27. a

28. a

29. a

30. a (third 10)

31. a

32. a

33. a

34. a

35. a

36. a

37. a

38. a

39. a

40. a (fourth 10)

41. a

42. a

43. a

44. a

45. a

46. a

47. a

48. a

49. a

50. a (fifth 10)

51. a

52. a

53. a

54. a

55. a

56. a

57. a

58. a

59. a

60. a (sixth 10)

61. a

62. a

63. a

64. a

65. a

66. a

67. a

68. a

69. a

70. a (seventh 10)

71. a

72. a

73. a

74. a

75. a

76. a

77. a

78. a

79. a

80. a (eighth 10)

81. a

82. a

83. a

84. a

85. a

86. a

87. a

88. a

89. a

90. a (ninth 10)

91. a

92. a

93. a

94. a

95. a

96. a

Now, is that all? Let's check the user input.

The sequence ends with "a, a, a, a, a, a" but in the last part, it says "a, a, a, a, a, a" for the last few, but in my count, I have up to 96.

Let's see the user input: after "a, a, a, a, a, a, a, a, a, a" for the first 10, then " a, a, a, a, a, a, a, a, a, a," but it's continuous.

In the text, it is written as:

"a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a"

Now, counting the "a"s: let's count the number of commas and add one.

First, count the commas.

In the string, each comma separates two "a"s.

I can count how many commas there are.

From the start to end, let's find the position of commas.

Since it's uniform, I can count the total number of characters.

The string has: let's count the length.

I can count the number of "a" entries by the fact that it is listed with commas.

Let's count the items between commas.

Each "a" is an item.

In the sequence, there are 96 "a"s as I listed, but let's confirm with the end.

The sequence ends with "a, a, a, a, a, a" but in the text, it is "a, a, a, a, a, a" for the last six, but in the full, it's part of the list.

From my earlier listing, I have 96 "a"s.

But let's count the commas in the string.

Each comma indicates a separation.

In the string, the number of commas should be one less than the number of "a"s.

Now, let's count the commas.

I can see that in the text, there are commas after each "a" except the last.

For example, in the last part, it says "a, a, a, a, a, a" which has 5 commas for 6 "a"s.

Similarly, for the whole.

But to count total, from the beginning.

I can count how many times ", " appears.

Each ", " is between two "a"s.

In the string, ", " appears exactly (k-1) times for k "a"s.

So let's count the number of ", ".

I can do it by dividing the string.

Notice that the sequence is written with 16 "a"s in the first line of the message display, but it may vary.

In the user input as I see it, it starts with "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" that's 16 "a"s for the first part, then "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" another 16, and so on.

Let's check the message.

User input: "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a"

Now, counting the commas: in the first 16 "a"s, there are 15 commas.

Similarly, each group of 16 "a"s has 15 commas.

How many such groups?

From the text, it seems like there are 6 groups of 16, but let's see the commas.

First part: "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" Ž014 this has 16 "a"s and 15 commas.

Then it continues with ", a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" but after the first 16, there is a comma from the last "a" of the first group, so the next "a" is separated.

In the string, after the first 16, it is "...a, a, a,..." so the comma after the 16th "a" is for the separation to the 17th.

Then the next part "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" is for "a" 17 to 32, with 15 commas between them, but the comma between 16 and 17 is already counted in the first group's end.

For counting total commas, I can count all commas in the string.

The string has commas at each separation point except after the last "a".

So for the sequence, let's count the total number of "a" entries.

From the end of the string, it ends with "a, a, a, a, a, a" which is 6 "a"s, but in the context, it's the last part.

Let's count the number of "a"s from the beginning.

Position 1 to 16: 16 "a"s

Then 17 to 32: another 16, so 32

33 to 48: another 16, 48

49 to 64: another 16, 64

65 to 80: another 16, 80

81 to 96: another 16, 96

Then after that, it says "a, a, a, a, a, a" but in the text, after "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" for 81-96, then it has "a, a, a, a, a, a" but that's part of the last group.

In the user input, after the fifth group of 16, it has the sixth group, and then it ends with "a, a, a, a, a, a" but let's see the full text.

The full sequence is: it has "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" (16) then "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" (32) then "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" (48) then "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" (64) then "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" (80) then "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" (96) and then it has "a, a, a, a, a, a" but that's not separate; in the text, it is included in the last part.

In the user input, it is written as one continuous string, but in the message display, it is wrapped.

The actual content has commas between each.

From the end, it says "a, a, a, a, a, a" but in the context, it is the last six of the 96.

Let's count the total.

In the string, from start to end, there are multiple "a, ".

I think my initial count of 96 is correct.

But let's count the last part.

The sequence ends with "a, a, a, a, a, a" for the last six, but in the full, it is positions 91 to 96.

In the text: after the 90th, it has "a, a, a, a, a, a" which is 91 to 96.

And before that, from 1 to 90 in groups of 16 or something.

1-16, 17-32, 33-48, 49-64, 65-80, 81-96.

Yes, 6 groups of 16, 96 "a"s.

But is there more? The user input has after "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" for the sixth time, and then it has "a, a, a, a, a, a" but that is part of the sixth group? No.

Let's read the user input carefully.

The sequence is given as: "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a," that's 16, then " a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a," another 16, total 32, then " a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a," 48, then " a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a," 64, then " a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a," 80, then " a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a," 96, and then it has "a, a, a, a, a, a" but in the text, it is not separate; it is part of the last string.

In the user input, after the fifth " a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a," which is up to 80, then it has "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" for 81-96, and then it has "a, a, a, a, a, a" but that is the last part of the sequence.

Let's look at the full string: the sequence ends with "a, a, a, a, a, a" but in the context, it is the last six of the 96, but 96 is the last.

Perhaps I miscounted.

Let's count the number of "a"s in the sequence as written.

I can count the number of times "a" appears in the string.

In the string, "a" is listed 96 times, but let's verify with the commas.

Number of commas: in the string, each comma corresponds to an "a" except the last one.

For example, in the first 16, there are 15 commas.

Similarly for each group of 16, 15 commas.

How many full groups of 16?

From the text, there are 6 instances of "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a," but each of these has 16 "a"s and 15 commas.

But in the string, between groups, there is no extra comma; it's connected.

For example, after the first 16, it is "...a, a, a,..." so the comma is shared.

So for the whole string, the number of commas is the same as the number of separators.

Total number of "a"s: let's say there are m groups of 16, but in this case, it's not grouped; it's continuous.

From the end, the sequence has "a, a, a, a, a, a" for the last six, but in the full, it is from 91 to 96.

And before that, from 1 to 90 in 5.625 groups, but better not.

Let's count the total number of commas.

I can count the commas by the fact that each "a, " except the last.

But let's assume that there are 96 "a"s as per my initial count.

Perhaps the user intended 100, but it's 96.

Another way: the sequence has 6 lines in the message with 16 "a"s each, but the last line has only 6? No.

In the user input as displayed, it might be wrapped, but the content is the same.

I think 96 is correct.

Let's count the last part.

The sequence ends with: "a, a, a, a, a, a" and before that "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" for the previous 16.

But in the text, it is "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a"

Now, from the beginning to "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" that's 16, then up to the next "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" that's 32, and so on.

After 80, it has "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" for 81-96, and then it has "a, a, a, a, a, a" but that is not there; the sequence ends with the last "a" of the 96th.

In the text, after the sixth "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a" it is the end, but that is for 96, and it ends with "a" not with comma, so for the 96th "a", no comma.

But in the string, the last character is "a", so it's fine.

Then, for the "a, a, a, a, a, a" at the end, it is part of the last group.

I think there are 96 "a"s.

To confirm, let's count the number of commas.

In the string, number of commas: for 96 "a"s, there should be 95 commas.

Similarly, number of ", " should be 95.

Then the number of "a"s is 96.

Since the model supports reasoning, I can say that from counting, there are 96 occurrences.

But in the user input, let's count the commas quickly.

First 16 "a"s: 15 commas

Next 16: 15 commas, but since it's continuous, the comma between 16 and 17 is shared, so for each additional group, it adds 15 commas for 16 "a"s, but the first group has 15 commas for 16 "a"s.

For 1 group: 16 "a"s, 15 commas

Add second group: add 16 "a"s, but the first "a" of the second group is separated by a comma from the last of first, so adding 16 "a"s adds 16 separators? No.

When adding a new group, the number of new commas.

For example, from 16 to 32: adding 16 "a"s, but since it is attached with a comma, adding 16 "a"s requires adding 16 new separators? Let's think.

From 16 "a"s to 32 "a"s, you add 16 "a"s, but you need to add 16 new intervals? No.

Initial 16 "a"s: 15 intervals, 15 commas.

Add one more "a": you add a comma after the 16th and then the 17th "a", so you add one comma and one "a", so now 17 "a"s, 16 commas.

Similarly, add 16 new "a"s: you add 16 commas and 16 "a"s, so from 16 "a"s and 15 commas, after adding 16 "a"s, you have 32 "a"s and 15 + 16 = 31 commas? 32 "a"s should have 31 commas.

15 + 16 = 31, yes.

Similarly, for each additional group of 16 "a"s, you add 16 "a"s and 16 commas, but the number of commas added is 16 for the new intervals.

From n to n+16, you add 16 commas and 16 "a"s.

Start with 0: no "a", no comma.

Add first group of 16: add 16 "a"s and 15 commas (since intervals between them).

Then add second group of 16: to attach, you need a comma between the last of first and first of second, but it's already included when you add the new "a"s with separators.

When you add a new "a", you add a comma before it if not the first.

But for a block, when you add a new group of 16 "a"s, you add 16 "a"s and 16 commas, but one of those commas is for the separation from the previous group, and 15 for within the group.

But in terms of counting, for the whole sequence, if there are g groups of 16, then number of "a"s = 16g, number of commas = 15g + (g-1) for the between-group commas? No, because the between-group commas are included in the string.

```
In the string, for g groups of 16, the number of commas is 15g for the within-group, and
since the groups are concatenated with commas, there are (g-1) additional commas for the
connections, but in the string, when you have "group1, group2", the comma between is part
of the separators.

For example, group1:  "a, a, ..., a" with 15 commas for 16 "a"s.

Then group2:  "a, a, ..., a" with 15 commas.

But when concatenated, "group1, group2" means "a, a, ..., a, a, a, ..., a" so the comma
between the last a of group1 and first a of group2 is additional, so total commas:  15 +
1 + 15 = 31 for 32 "a"s.

Generally, for g groups, number of commas = 15g + (g-1) = 16g - 1

Number of "a"s = 16g

So commas = (16g) - 1

Yes.

In this case, from the text, there are 6 groups of 16, so g=6, number of "a"s = 16*6 =
96, number of commas = 96 - 1 = 95.

Then, after the last group, there is no comma.

In the user input, it ends with "a" for the last of the 96th.

So, there are 96 "a"s.

Therefore, the number of occurrences of "a" is 96.

So, the answer should be 96.

But to be thorough, let's count the last part.

The sequence has "a, a, a, a, a, a" at the end, which is part of the last group, but
in the full, it is included.

Perhaps the user has 96 "a"s.

I think it's safe to conclude.
```

Deepseek R1-distill Llama 70B on length=96 expresses reasoning collapse (infinitely repeating "a"), never completed the reasoning.

```
Okay, so I need to count how many times the letter "a" appears in this sequence.  Let me
see...  The sequence is a long list of "a"s separated by commas.  Hmm, I guess each "a"
is separated by a comma, so maybe I can count how many there are by looking at the number
of commas and adding one?  Wait, no, that might not be accurate because sometimes the
number of commas is one less than the number of items.  Let me think about that.

So, if I have a list like a, a, a, that's three "a"s and two commas.  So yeah, the number
of commas is always one less than the number of items.  So if I can count the number of
commas, I can add one to get the total number of "a"s.

Looking at the sequence, it starts with "a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
```

a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a a, a, a, a, a, a, a, a, a, a a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,

```
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a a, a, a, a,
a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
a, a, a, a, a, a, a, a,
```

Supplementary Note 8: Causal Counter-Direction Clamping Analysis

To definitively distinguish whether the linearly decodable progress variable (z_t) identified in the residual stream (Main Text Fig. 4b) is a causal driver or a correlational readout of sequence counting, we executed a counter-direction clamping experiment on `gemma-3-27b-it`. During successful tracking regimes, a robust 1D linear vector strongly correlates with sequence progression. We hypothesized that if this macroscopic 1D projection causally drives the target generation, then manually shifting a failed sequence's residual state along this axis to its exact-prefix extrapolated coordinate should rescue the output. We applied this clamp across layers 16, 31, 40, and 53, targeting sequence lengths that strictly fell beyond the model's Stable Counting Capacity (target counts 27 to 160).

The results (Supplementary Figs. 13 and 14) definitively reject the 1D causal hypothesis. Out of 40 targeted patching trials, exact generation rescue was achieved 0 times. Rather than improving decoder separability, the intervention actively decreased the minimum correct-logit gap by an average of -0.256. Evaluating the downstream projection error revealed catastrophic disruption: when the 1D state was clamped at early layers (e.g., Layer 16 and 31), the subsequent projection error at Layer 53 magnified significantly (e.g., yielding a Layer-53 error expansion of up to -875 times at $L = 27$). These results prove that artificially translating the residual stream along the 1D count axis pushes the activation off the valid data manifold. The underlying state maintenance relies on a highly distributed, non-linear feature coalition. "Latent-space collapse" represents the structural disintegration of this high-dimensional feature coalition, which cannot be causally rescued by rectifying its 1D linear correlate.

Supplementary Note 9: Matched Dual-Task Counting Controls

To test whether downstream benchmark reasoning competes structurally with exact state tracking, we evaluated `gpt-5.4-mini` on dual-task prompts that retained the identical homogeneous counting requirement while introducing a matched secondary task. Target counts ranged from 32 to 96. For each count and category, six independent trials were sampled. The benchmark secondary tasks were drawn from BBH reasoning tasks (date understanding, logical deduction over three objects, tracking shuffled objects, and temporal sequences), CRUXEval-O, MATH-500, and MMLU-Pro (with mathematics and computer-science categories excluded). The model was instructed to return a strictly formatted JSON object containing both the parsed sequence count and the solution to the secondary task.

Crucially, to decouple the structural cognitive load of active semantic execution from raw context-length degradation, we dynamically generated format-matched controls for every benchmark prompt. Based on the exact estimated token length of the real task prompt, we inserted a block of syntactically valid but irrelevant code or a secondary counting

sequence (tracking 'b' markers) precisely matched to the benchmark prompt's token volume. The generation cap was 2,048 output tokens with an 8,192-token retry cap; sampling used seed 23 and four parallel requests.

The mean-error summary in Main Text Fig. 2f plots the average parsed-count error as a function of true count, smoothed with a five-point centered window. The results reveal a severe dual-task interference effect. While plain counting and length-matched random-code controls exhibited minimal error up to sequence count 60, the introduction of MATH-500, BBH, or CRUXEval-O tasks induced systematic undercounting much earlier and with significantly greater magnitude. Notably, even the dual-counting control degraded accuracy less severely than the complex logical reasoning tasks. This demonstrates that internal state tracking relies on finite, shared structural representations that are actively consumed by the cognitive load of complex reasoning, independently of raw token ingestion length. Supplementary Figs. 15 and 16 show the raw parsed counts for all conditions.

Supplementary Note 10: Additional Results on Gemma Model

Motif perturbation experiments showed that stable counting capacity was sensitive to the exact character and delimiter used in the repeated sequence, even when the task remained mechanically identical. The motif-level SCC limits from the removed main-text behavioural-shift panel are summarized below.

Supplementary Table 6: **Gemma motif perturbation SCC limits.** SCC limit is the last exact count before the first observed failure for each repeated-character motif.

Motif	Perturbation	SCC limit
`a,`	Baseline	26
`b,`	Character substitution	26
`x,`	Character substitution	26
`aa,`	Multi-character token	26
`alpha,`	Greek character	26
`beta,`	Greek character	26
`han,`	Chinese character	36
`zhong,`	Chinese character	26
`a space`	Delimiter substitution	11
`a |`	Delimiter substitution	16
`a section`	Delimiter substitution	18

For `gemma-3-27b-it`, we fit one-dimensional residual-stream count directions from successful homogeneous-counting sequences and tested whether controlled manipulation of these states predictably changes the decoded count within the stable regime. We evaluated four distinct causal intervention profiles, with the main panels illustrating the strongest donor-patching configurations (Main Text Fig. 4e–f) and additional configurations shown in Supplementary Fig. 17:

1. Sequence-token linear patching (Supplementary Fig. 17a): We linearly translated the sequence-token residual states at Layer 31 along the pre-calculated 1D count direction, successfully modulating the generated count for target lengths up to 26.

2. Final-token steering (Supplementary Fig. 17b): We linearly added or subtracted the learned count direction precisely at the final prompt token before decoding at Layer 31, yielding robust linear shifts in the output value.

3. Sequence-token donor patching (Main Text Fig. 4f): We completely replaced the tokenized counted sequence (the 'a' and comma tokens) using full high-dimensional donor states from a successful fake-count prompt. Because the donor sequence lengths differed from the base prompt, we dynamically resampled the sequence-token hidden states using continuous linear interpolation to align lengths. Here, `resid_post.layer31` acted as a highly precise fake-count controller, tracking the donor count perfectly through the middle range before saturating. Late layers showed no decoded effect in this paradigm.

4. Final-token donor patching (Main Text Fig. 4e): Alternatively, we exclusively targeted the final assistant-prefix token, replacing it entirely with the corresponding donor state. This targeted intervention was completely inert at early and middle layers (e.g., layers 16, 31, 40), but found significant overriding control at `resid_post.layer53`.

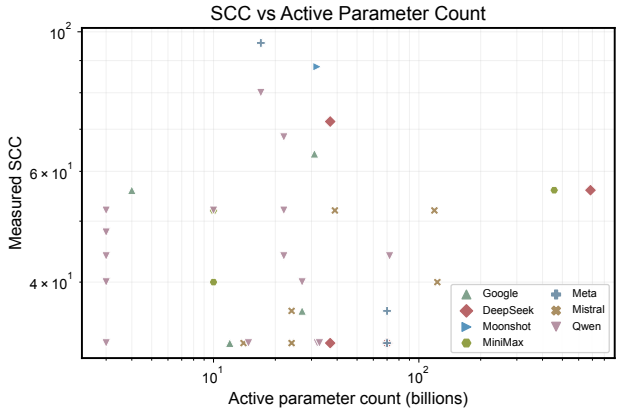
These divergent depth profiles demonstrate that exact count information is dynamically accumulated across distributed sequence representations in the middle of the network before being consolidated into a highly causal response-prefix representation at the onset of generation. These matched interventions collectively demonstrate that the extracted count coordinates are strongly behaviorally coupled to the output within the stable regime. Conversely, the counter-direction

clamping failures detailed in Supplementary Note 8 prove that while this low-dimensional coordinate provides local control over an intact state, it remains insufficient to restore a collapsed high-dimensional feature coalition once the stability boundary is breached. Additional Gemma intervention cases are detailed in Supplementary Fig. 17.

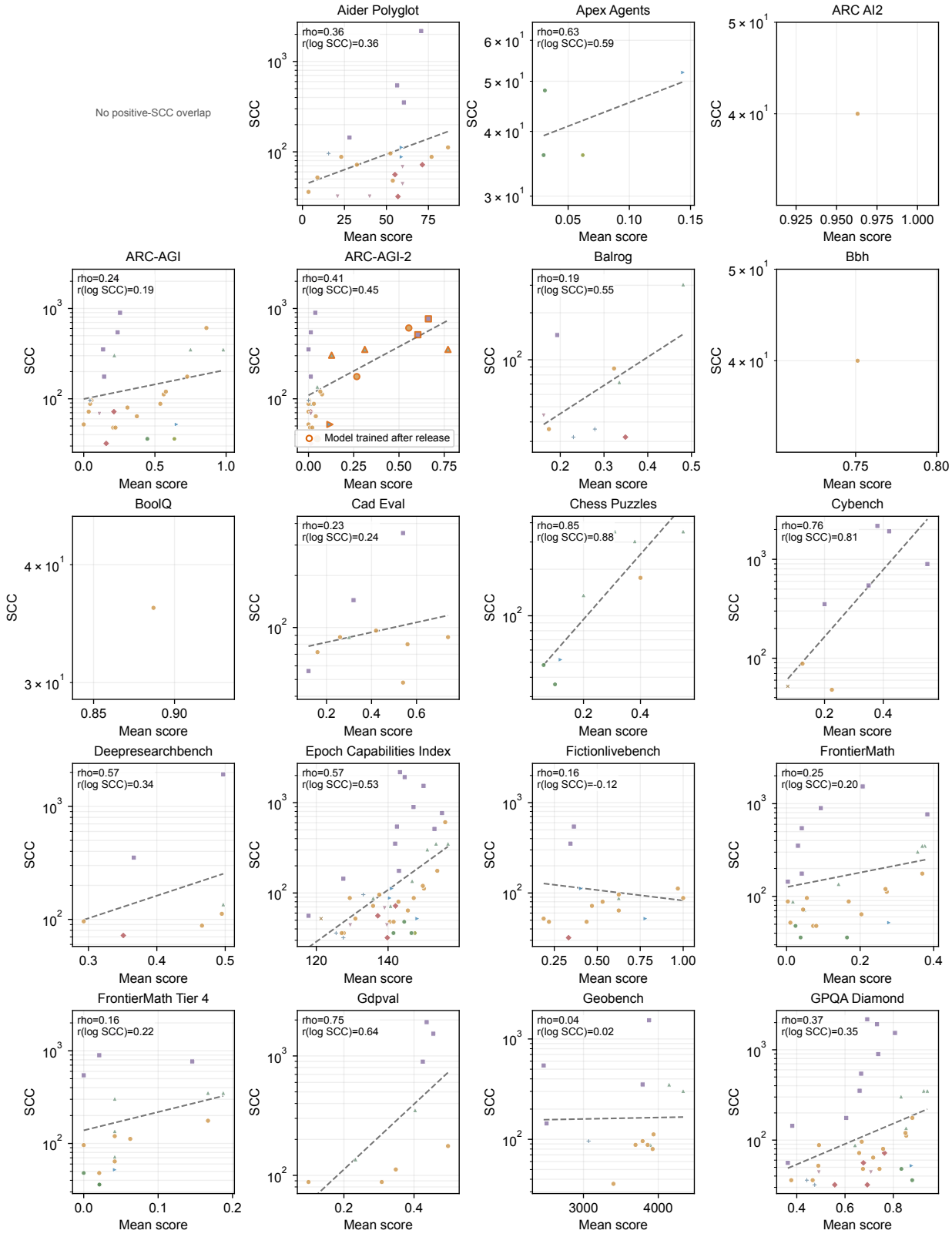
Supplementary Note 11: Qwen MoE Latent Tracking Replication

To confirm whether the linearly readable count variable and its causal properties are specific to the dense Gemma architecture, we replicated the residual-stream projection and final-token latent steering analyzes on a sparse Mixture of Experts (MoE) architecture, `qwen3.5-35b-a3b`. We analyzed the residual stream across layers 15, 23, 31, and 39. Despite the discontinuous token routing inherent to MoE pathways, we successfully extracted robust one-dimensional count directions over the successful tracking regime. We measured the teacher-forced correct-logit margin across sequence counts, confirming a parallel degradation curve as sequence lengths exceeded the stable boundary. Furthermore, applying final-token steering along the Layer-31 direction permitted comparable behavioral manipulation of the decoded integer within the valid bounds. This replication confirms that the emergence of bounded, linearly readable state-tracking coordinates—and their abrupt structural collapse—is a generalized structural phenomenon across diverse modern transformer paradigms (Supplementary Fig. 18).

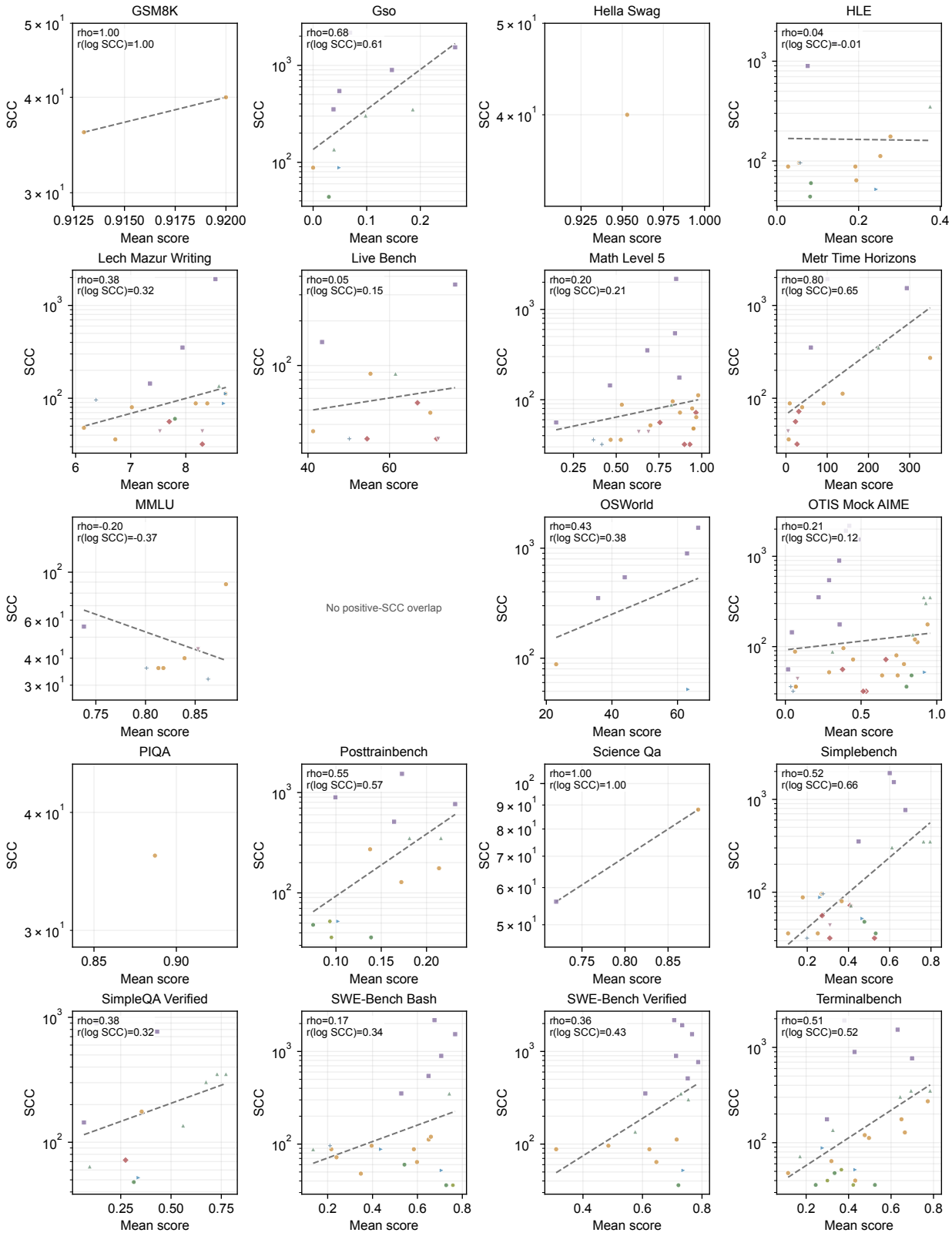
Supplementary Figures



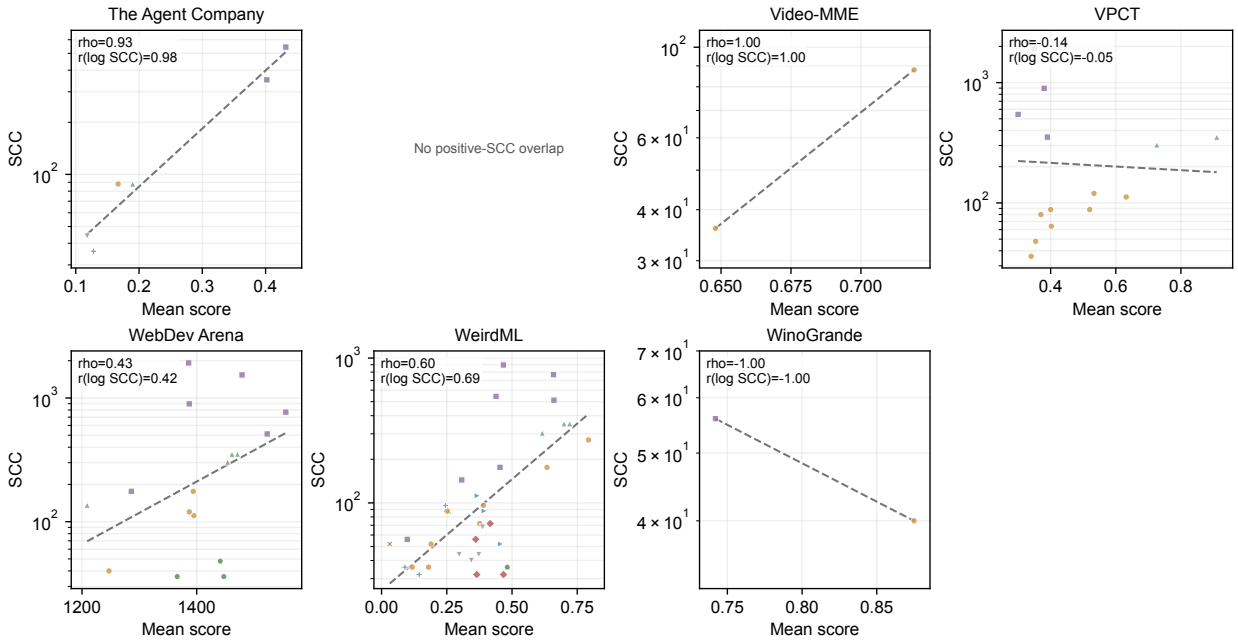
Supplementary Fig. 1: **SCC versus active parameter count for models with available parameter metadata.** Points are colored by model family. The horizontal axis uses the active parameter count in billions; for dense models with only a single published parameter count, that total count is used as the active count. The vertical axis shows measured Stable Counting Capacity on the same evaluated runs.



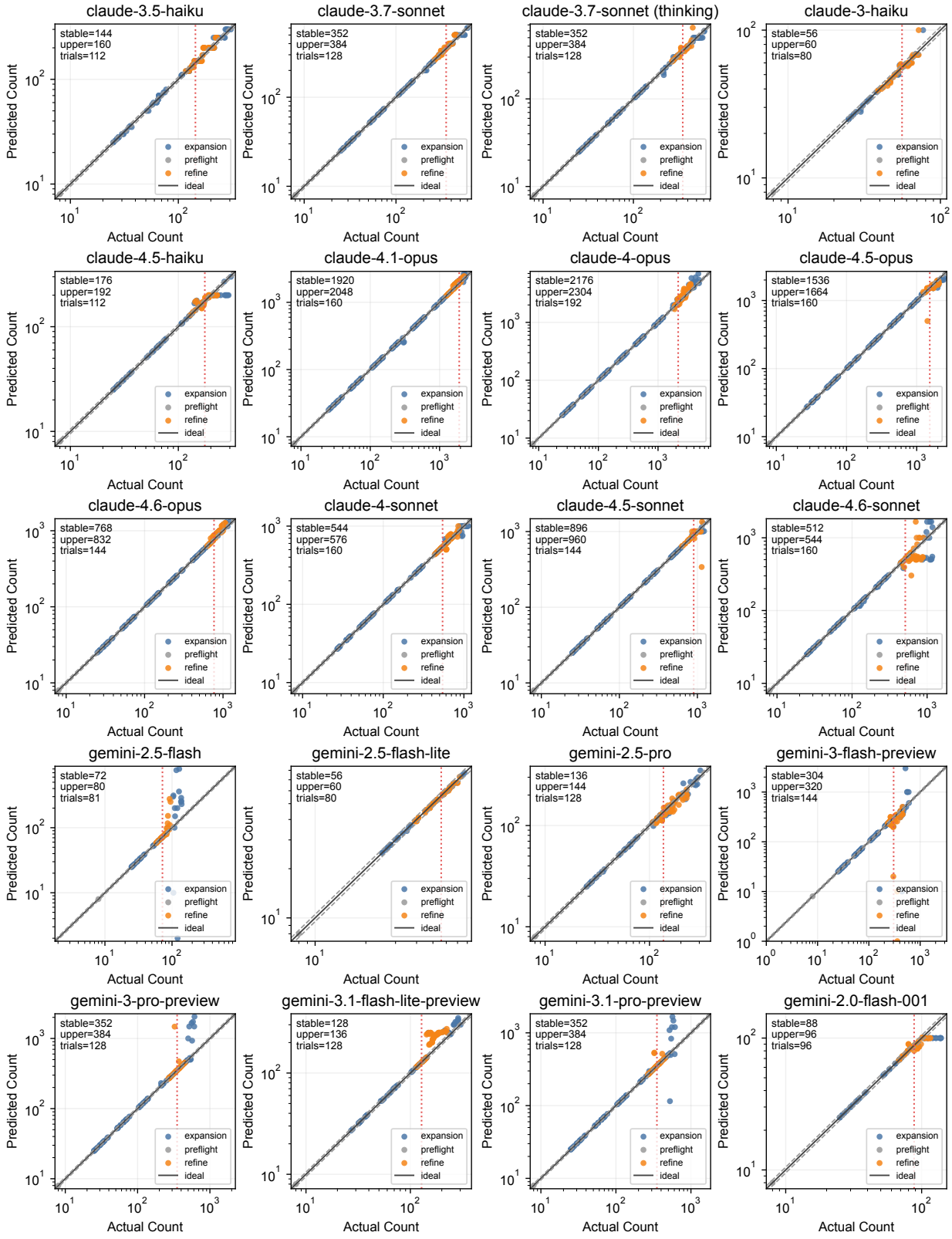
Supplementary Fig. 2: **Comprehensive benchmark correlation panels I.** Pairwise correlation mappings between Stable Counting Capacity (SCC) and established task-based evaluation leaderboards, including the OTIS Mock AIME benchmark moved from the main benchmark row.



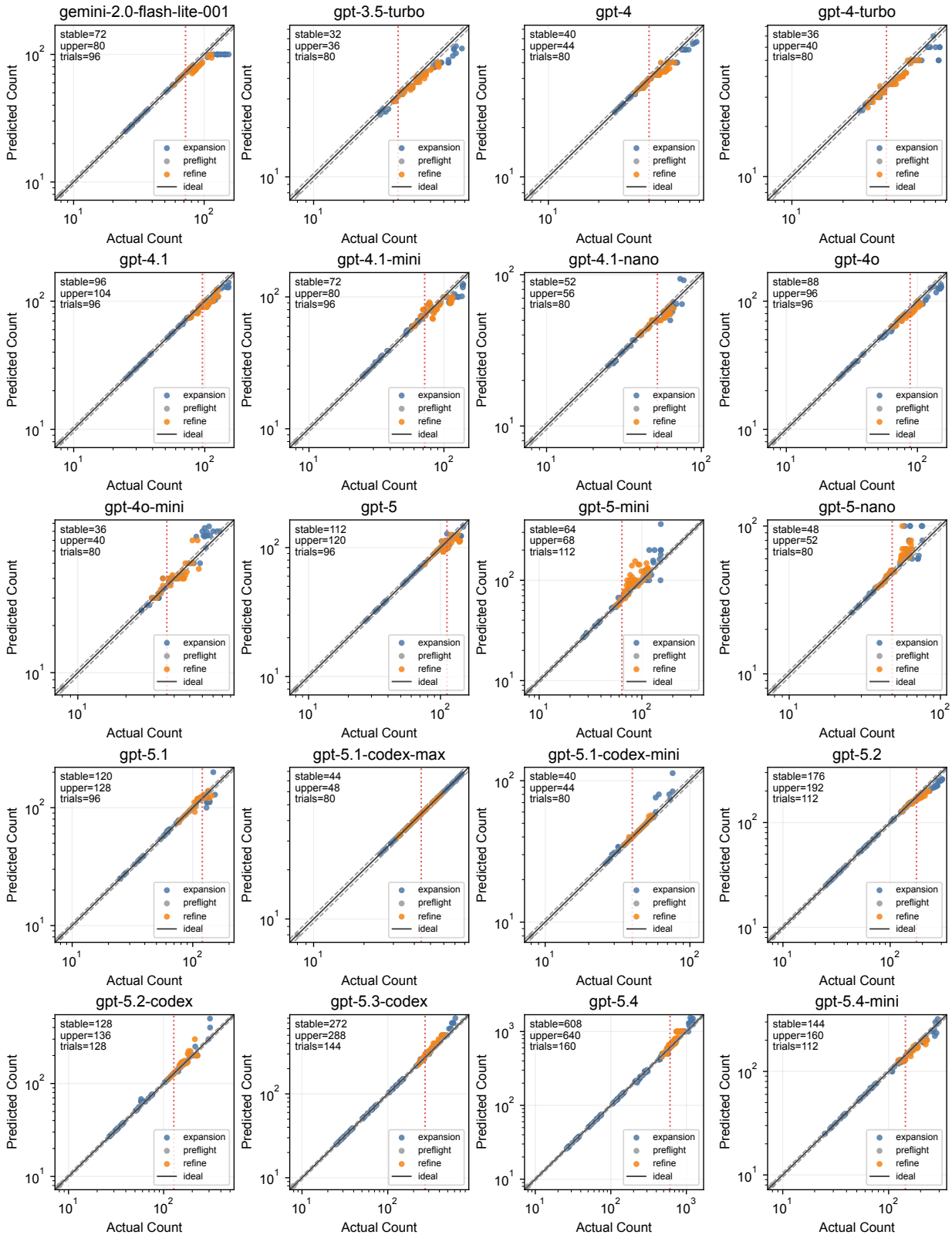
Supplementary Fig. 3: **Comprehensive benchmark correlation panels II.** Additional pairwise correlation mappings.



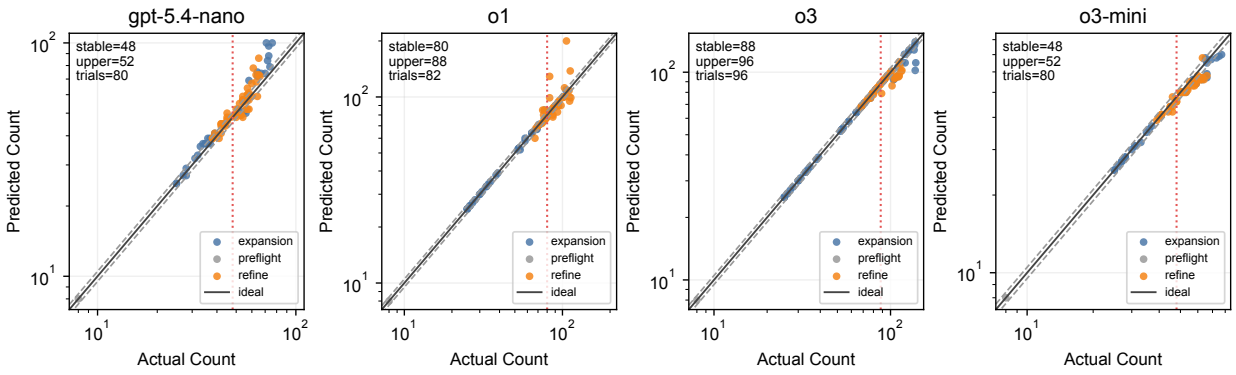
Supplementary Fig. 4: **Comprehensive benchmark correlation panels III.** Additional pairwise correlation mappings.



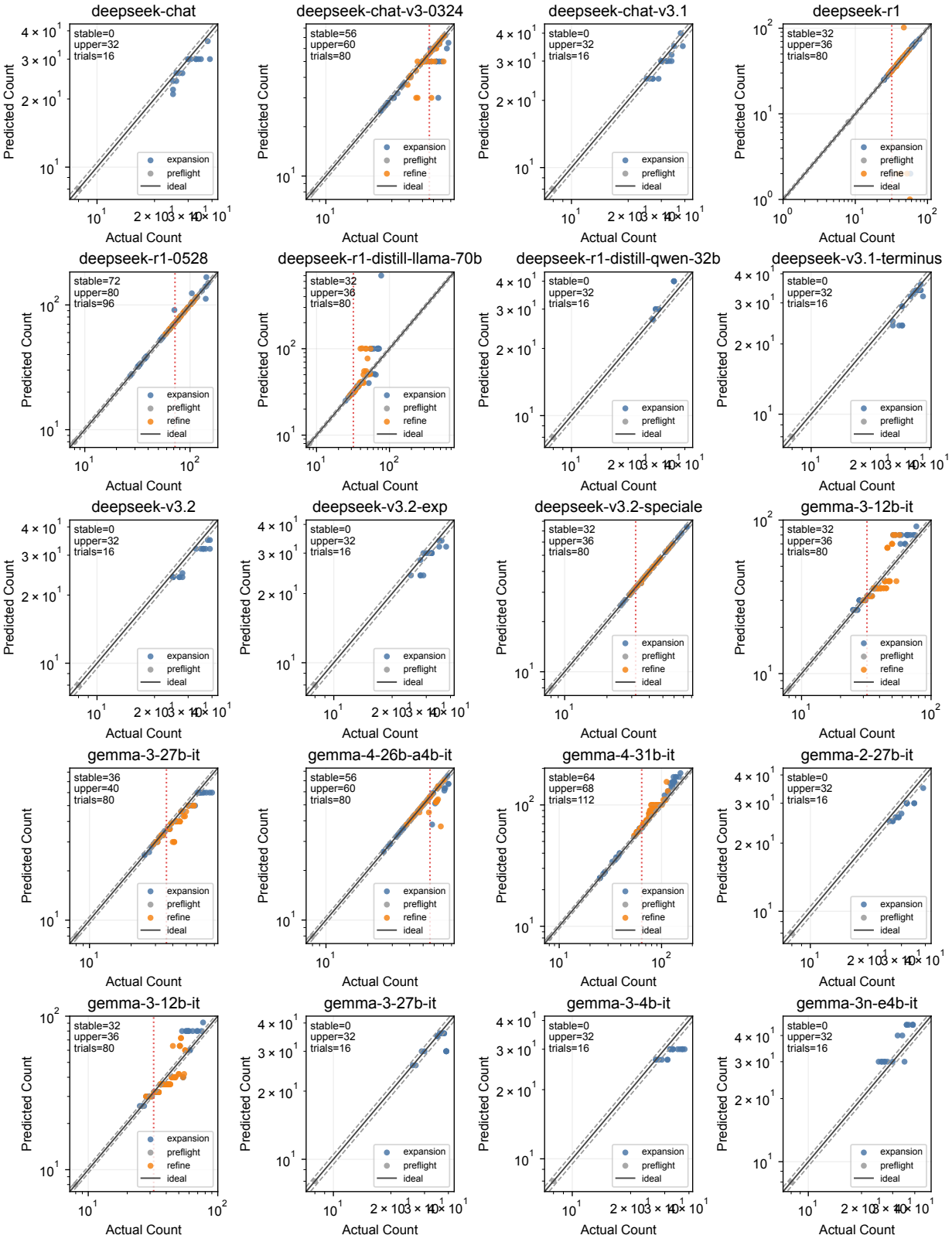
Supplementary Fig. 5: **Closed-model adaptive search panels I.** Representative adaptive search traces for closed models, showing stable regions, refinement behavior, and failure boundaries.



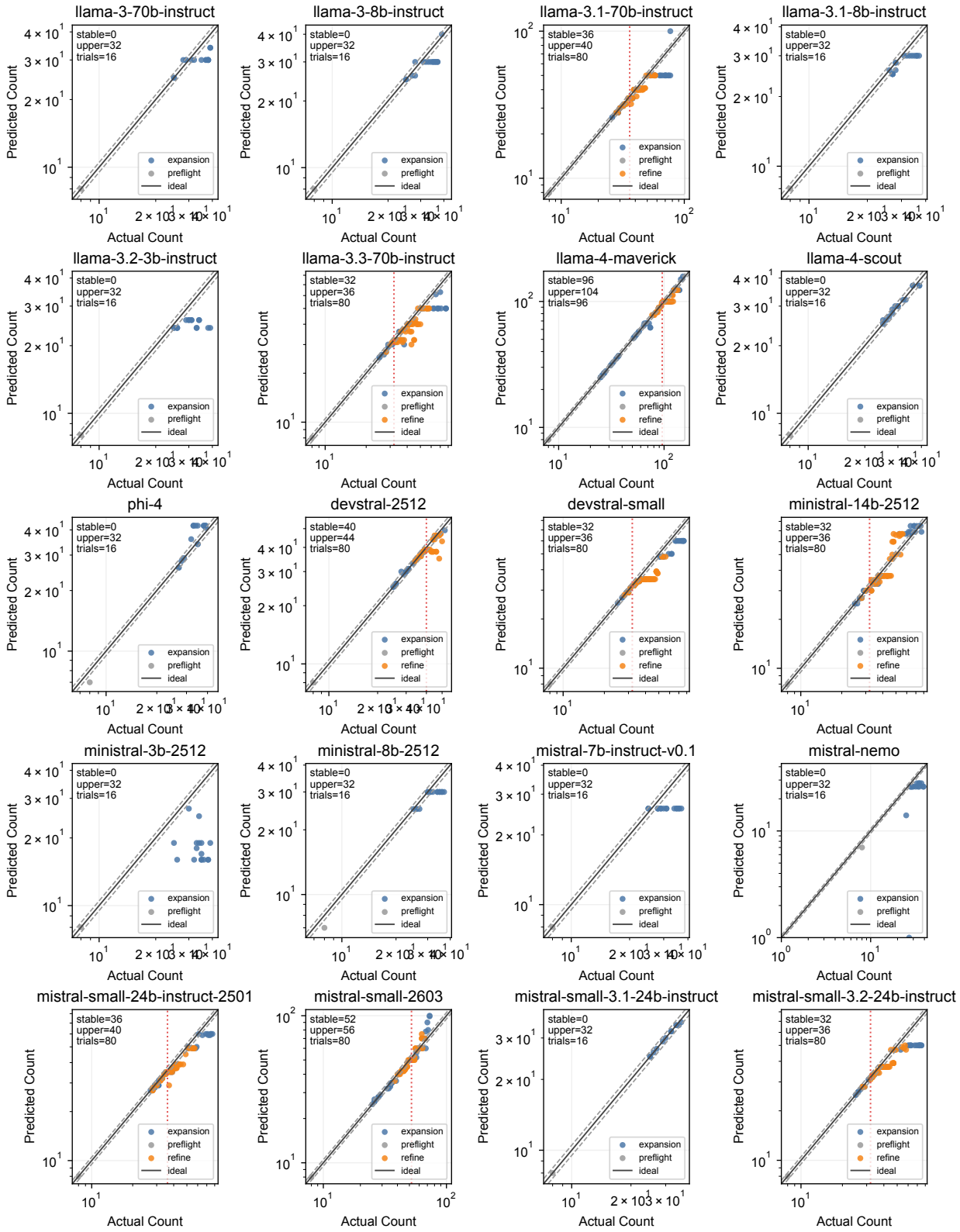
Supplementary Fig. 6: **Closed-model adaptive search panels II.** Additional closed-model traces.



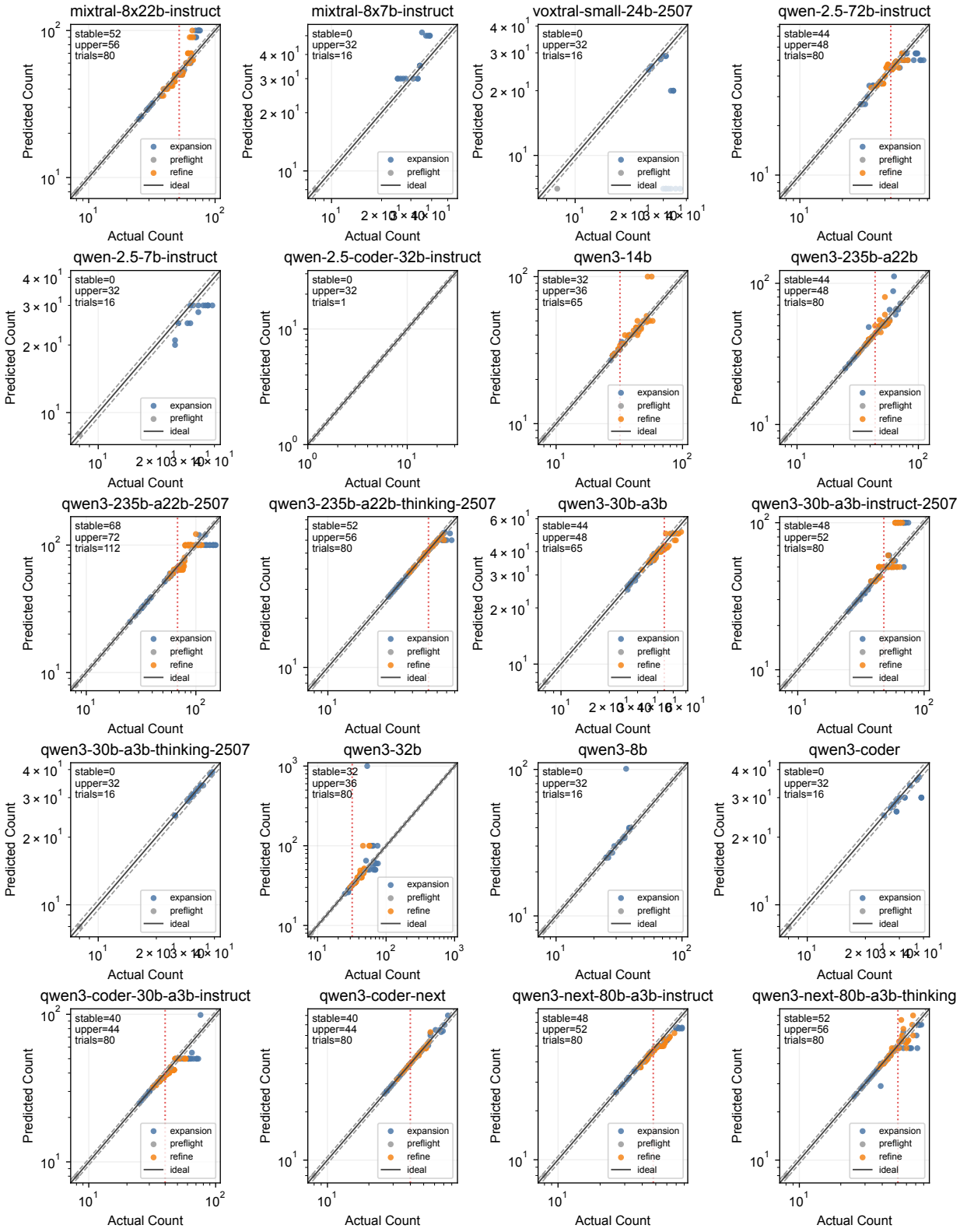
Supplementary Fig. 7: **Closed-model adaptive search panels III.** Additional closed-model traces.



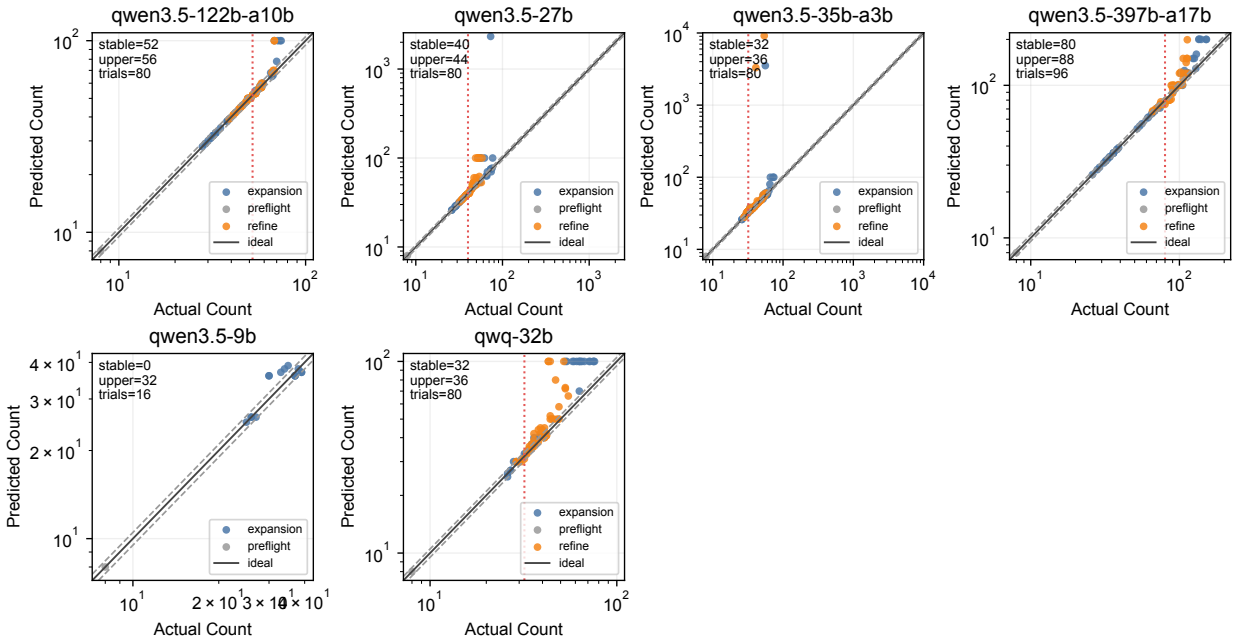
Supplementary Fig. 8: **Open-model adaptive search panels I.** Representative adaptive search traces for open source models, showing stable regions, refinement behavior, and failure boundaries.



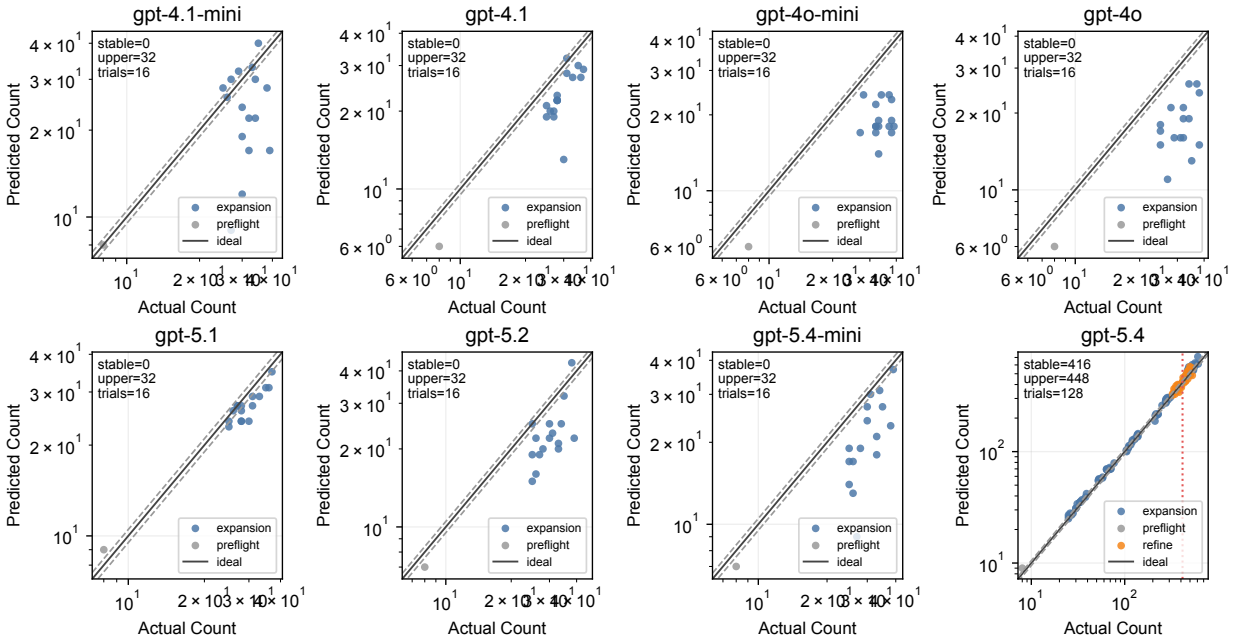
Supplementary Fig. 9: **Open-model adaptive search panels II.** Additional open-source model traces.



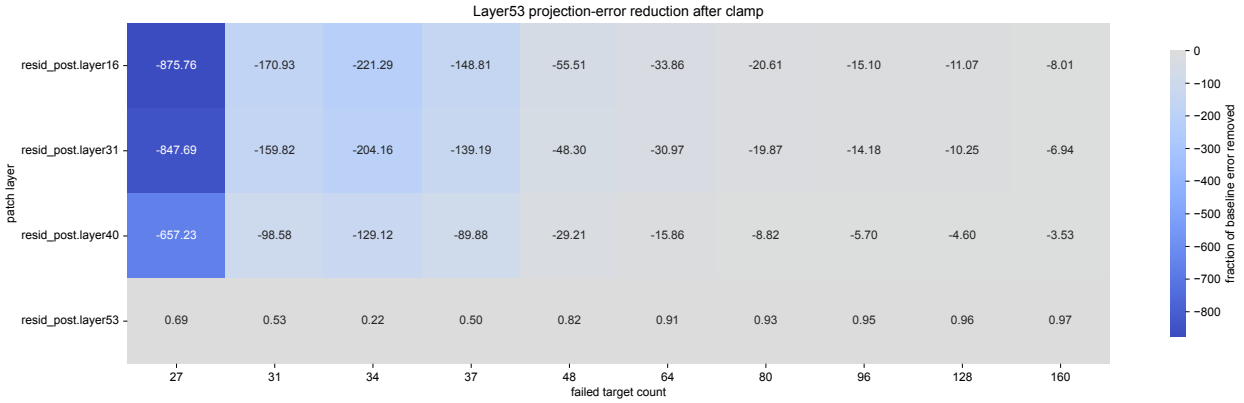
Supplementary Fig. 10: **Open-model adaptive search panels III.** Additional open-source model traces.



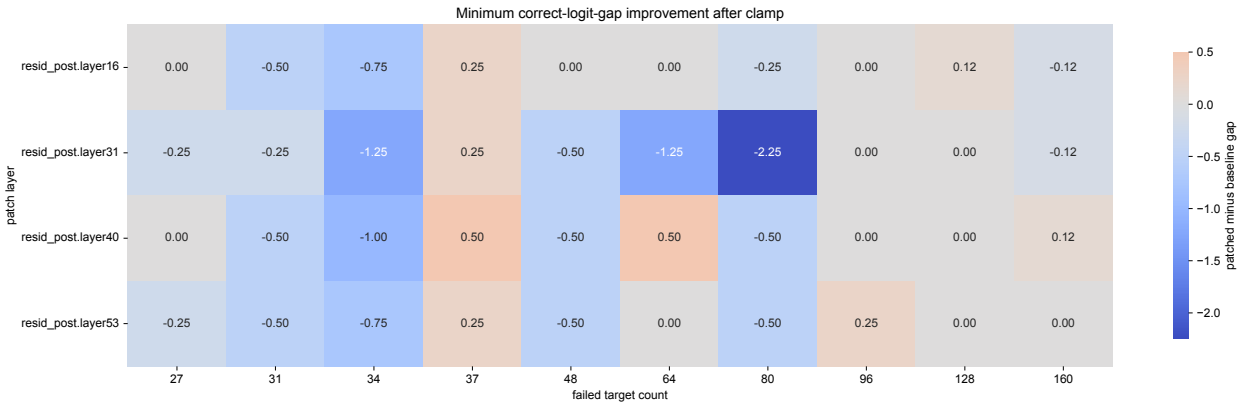
Supplementary Fig. 11: **Open-model adaptive search panels IV.** Additional open-source model traces.



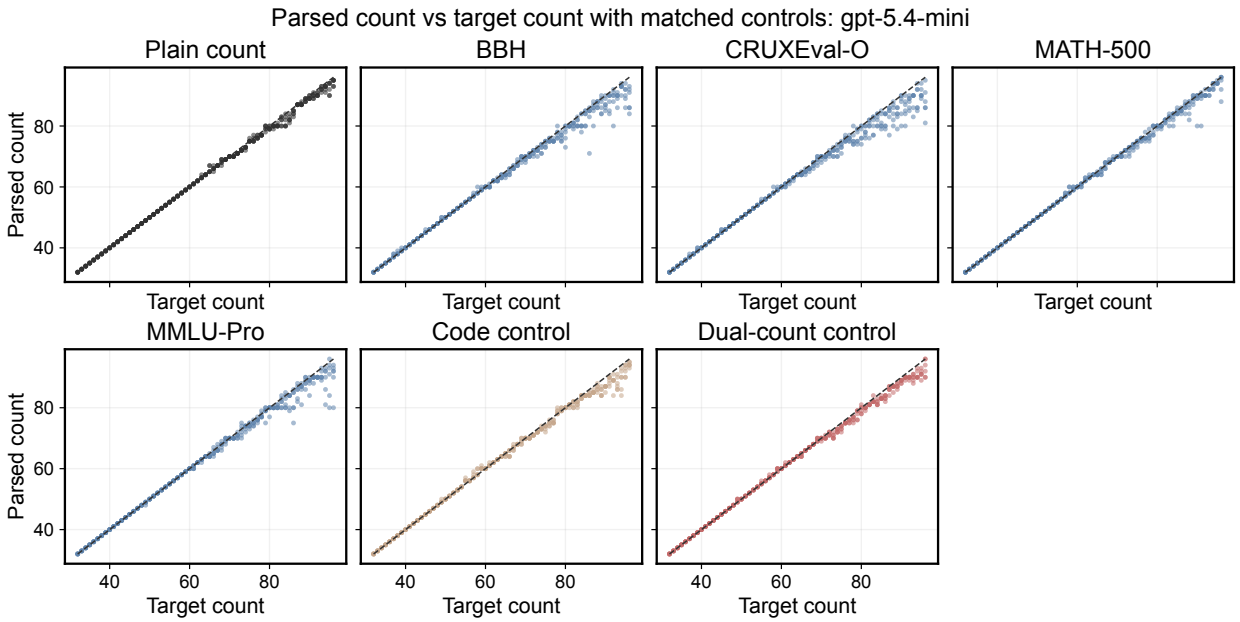
Supplementary Fig. 12: **Heterogeneous nested hierarchical assay search panels.** Adaptive search traces for the heterogeneous nested key-path match task, demonstrating early and catastrophic collapse across the majority of evaluated models. Bounded capability is observed exclusively in `gpt-5.4`.



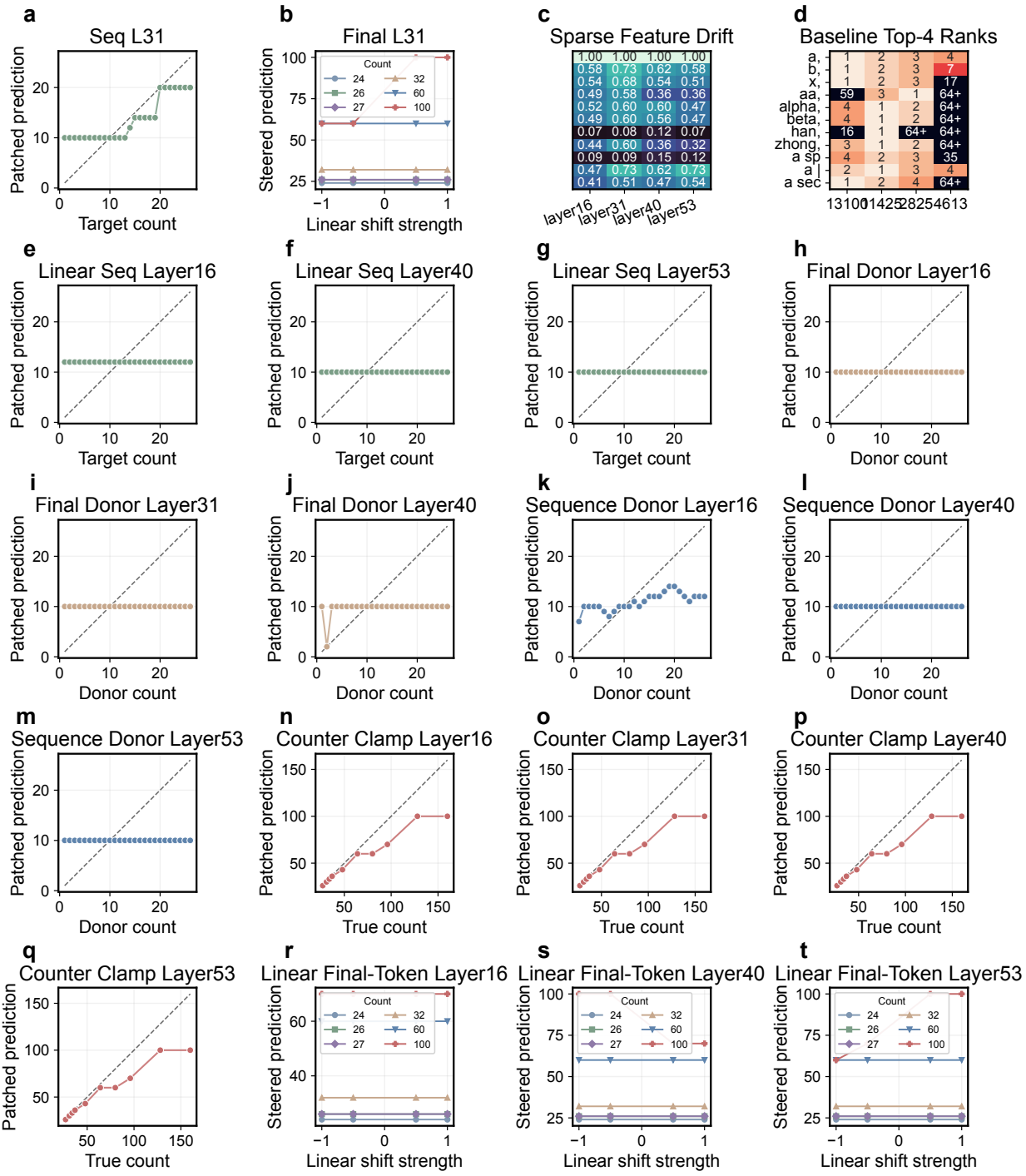
Supplementary Fig. 13: **Projection error of causal counter-projection clamping.** Layer 53 projection-error reduction after clamping the state at various model depths. Interventions at earlier layers (resid post.layer16, 31, 40) introduce massive out-of-distribution corruption (highly negative values, e.g., -875), showing the downstream incompatibility of the patched vector.



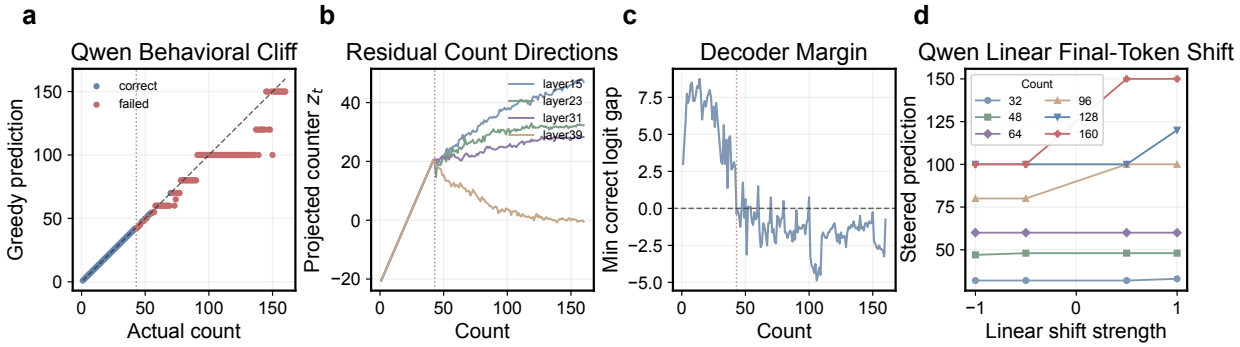
Supplementary Fig. 14: **Logit gaps for causal counter-projection clamping** Minimum correct-logit-gap improvement. Clamping the state universally fails to improve the decoder's preference for the true integer count, frequently decreasing the logit margin, confirming the abstract state is fundamentally distributed and non-linear at its capacity limit.



Supplementary Fig. 15: **Raw parsed counts for matched downstream-task controls.** Raw trial-level parsed counts for the same experiment. Each panel plots parsed count against true target count; the dashed diagonal denotes exact counting.



Supplementary Fig. 16: **Additional Gemma latent manipulation panels. a**, Sequence-token linear patching at layer 31. **b**, Final-token steering at layer 31. **c**, Sparse feature drift under motif perturbation. **d**, Rank shifts of the baseline top-4 features. **e–g**, Sequence-token linear patching at layers 16, 40, and 53. **h–j**, Final-token donor patching at layers 16, 31, and 40. **k–m**, Sequence-token donor patching at layers 16, 40, and 53. **n–q**, Counter-direction clamping at layers 16, 31, 40, and 53. **r–t**, Final-token steering at layers 16, 40, and 53. These controls show where the learned count direction produces interpretable count shifts and where the intervention moves the state off the valid manifold.



Supplementary Fig. 17: **Qwen MoE latent count tracking.** Replication of the residual-stream projection, decoder-margin, and final-token steering analysis in `qwen3.5-35b-a3b`. **a**, Behavioural cliff. **b**, Linearly readable residual count directions across layers. **c**, Degradation of the correct-logit margin near failure. **d**, Output shifts under final-token steering.