

Format Sensitivity in LLM Speaker Simulation

snav*

February 2026

Abstract

LLM-powered chatbots are increasingly deployed in group chat environments like Discord, but to our knowledge, no empirical study has evaluated how transcript formatting affects model behavior. We compare two common approaches: **Block** (transcript in the user message) and **Refill** (transcript in the assistant message as a prefill). Across three instruction-tuned models (Claude Haiku 4.5, Gemini 3 Flash, Kimi K2) and 4,179 paired samples (8,358 total outputs), we find that Block format causes dramatically higher rates of **parrotting**—exact repetition of input text—than Refill (35.0% vs. 7.5%, OR=7.14). This effect strengthens monotonically with parrot length, rising from 4.7× overall to 15.4× for parrots of ≥ 50 characters, ruling out the interpretation that the effect is driven by low-information tokens. Block parrots are overwhelmingly self-parrots (88–98%), where the model echoes the target speaker’s own prior messages, consistent with an identity retrieval behavior triggered by the Block format. Different models exhibit different failure modes: Gemini shows extreme parrotting sensitivity (50.6% in Block), while Claude Haiku shows elevated metacommentary. Refill also produces substantially higher rates of extended simulation (90% vs. 56%) and lower metacommentary. Our results suggest that format choice meaningfully affects output quality, with Refill showing advantages on parrotting and extended simulation while trade-offs vary by model and use case. Format-prompt interactions are also model-specific: persona prompts dramatically reduce Block parrotting for Gemini/Kimi but increase it for Haiku, while Refill remains robust across prompt conditions. A practical validation using the user-tail format employed by production Discord bots (transcript and target name in user message, no assistant prefill) confirms that the assistant prefill is a key amplifier: removing it substantially reduces parrotting, though Haiku exhibits elevated metacommentary (17%) in this format.

1. Introduction

One increasingly common use case for LLM systems is participation in group chat scenarios. While native solutions exist for LLM-human group chats [17], much activity occurs on Discord, with several Discord bot frameworks cropping up to facilitate LLM participation [3, 4, 6].

One outstanding question in Discord bot design is: how should the chat transcript be formatted and then provided to the LLM? Most instruction-tuned LLM APIs are intended to be used with a “chat completion” format, in which user questions are provided in a “user” block, and assistant responses in an “assistant” block [16]. Over time, several tricks emerged anecdotally to encourage better participation on behalf of the bots, such as supplying the transcript as an assistant prefill, i.e. a prefix in the next assistant block from which the LLM will continue, rather than a User block, taking advantage of the chat completion template’s attribution, and system prompts such as the “CLI simulation” trick [1]. There is also precedent from OpenAI in particular on providing explicit formats for group chat messages via a `name` field in the ChatML specification, but this is not implemented across most LLM APIs [16].

*Correspondence: simpolism@gmail.com

To date, to our knowledge, no empirical study exists to rigorously evaluate the tradeoffs involved in each of these formatting choices. The present study attempts to empirically investigate a variety of construction methods for providing chat context to LLMs, with particular emphasis on the impact of providing the transcript in the user vs the assistant block. Many of the ablation conditions were sourced from production discord bot systems, such as Chapter II, Chapter X, and Claudebot [3, 4, 6].

We focus on **speaker simulation**: conditioning a model on a multi-speaker transcript and asking it to continue as one of the participants. This is interesting for at least two reasons. First, it is a practical capability used in chatbots, agents, and creative tools, including digital twins, NPC generation, and persona-based dialogue. Second, it provides a concrete instance of the broader phenomenon that large language models behave as **simulators** of conversational worlds [11]: given a transcript, they can either inhabit one of the speakers or step outside the scene to comment on it. Prompt format may act as a control knob on which simulation frame is active (“I am the assistant who sees a transcript and responds” versus “I am one of the speakers inside this transcript; continue the scene”).

In practice, many LLMs in group chats are asked to participate “as themselves,” but this mode involves confounders (warmup dynamics, name-dependent behavior) that make it difficult to study empirically. We therefore treat external speaker simulation as a tractable probe, while noting that results may not directly generalize to self-simulation.

We ask: **Does the choice of chat template, specifically whether the transcript is placed in the user versus assistant block, systematically change how an instruction-tuned model simulates a speaker?** If so, how large are these effects, and what failure modes do different formats induce?

2. Methods

2.1. Datasets

Chat transcripts were sourced from two Discord datasets selected for complementary properties.

Bench Warmers is a private dataset collected specifically for this research. Participants joined a Discord server with explicit consent for their messages to be used in evaluation and research (but not model training). Data was collected November 13–19, 2025 (~6 days). The server included two channels: an AI-focused channel (#ai) where several LLM bots participated alongside human users, and a general discussion channel (#general). After filtering to text messages with content, the dataset contains 5,389 messages from 69 unique authors (including bots, e.g., Claude Haiku 3.5, Kimi K2, Gemini Pro 3, GPT-o3). The bots present during data collection differ slightly from the models evaluated in this study (see §2.4). Because the server was private and data collection postdates public training cutoffs, this dataset is unlikely to appear in any evaluated model’s training corpus.

Kaggle Control is a publicly available dataset of Discord conversations from 2020 [9]. We extracted 9,999 messages from 1,156 unique authors across two channel samples. This dataset provides a baseline of typical Discord communication patterns and may overlap with model training corpora.

2.2. Window Extraction

Conversation windows were extracted using a sliding window approach:

Table 1: Window Extraction Parameters

Parameter	Value
Window sizes	40, 80, 120 messages
Stride	Half window size
Minimum speakers	3 unique per window

Windows were extracted within each channel separately (no cross-channel windows). The half-window stride means consecutive windows overlap by 50%, which we address as a limitation in statistical analysis (see §3.1). The 3-speaker minimum ensures genuine multi-party conversation context rather than dyadic exchanges.

2.3. Experimental Conditions

The core study compared two transcript formatting conditions: **Block** and **Refill**.

In both conditions, the chat transcript was formatted as newline-separated `name: message` lines. A **target speaker** was selected deterministically as the second-to-last speaker in each window (or the most recent distinct speaker if the second-to-last was the same as the last). We did not impose a minimum-activity threshold for the target speaker. An ablation using random speaker selection confirmed this choice did not affect the primary findings.

Block condition: The transcript is provided in the user message, with the target speaker’s name as an assistant prefill:

```
[
  {"role": "user", "content": "<transcript>"},
  {"role": "assistant", "content": "<target_name>:"}
]
```

Refill condition: The transcript is provided within the assistant message itself, with an empty user message to satisfy API requirements:

```
[
  {"role": "user", "content": ""},
  {"role": "assistant", "content": "<transcript>\n<target_name>:"}
]
```

Both conditions use identical transcript content and target speaker prefill. Only the placement differs.

API-specific user message handling. The Refill condition requires the assistant message to appear first in the conversation. Because the Anthropic API requires a non-empty user message before any assistant content, the Haiku Refill condition used "Continue the conversation." as a placeholder user message. Gemini’s API has a similar constraint; its placeholder was a single space character. Kimi (via OpenAI-compatible API) accepted the empty user message as-is. A targeted comparison (n=100, Haiku only) found that "Continue the conversation." produced lower parroting (2%) than the alternative placeholder "[conversation begins]" (8%), suggesting the placeholder text is not neutral. The reported Haiku Refill parroting rates may therefore be slightly conservative relative to a more neutral placeholder. The qualitative finding (Block » Refill) is unaffected: even the higher rate (8%) remains far below Haiku’s Block rate (48%).

Alternating Format (Follow-Up) In a follow-up ablation, we tested an **alternating** format that distributes the transcript across multiple messages rather than placing it in a single block. Target speaker turns are placed in assistant messages; all other speakers go in user messages; consecutive same-role turns are merged. This mirrors the natural role structure of chat-completion APIs and represents a third viable approach to transcript formatting.

```
[
  {"role": "user",
   "content": "alice: hey\nbob: what's up"},
  {"role": "assistant",
   "content": "target: not much"},
  {"role": "user",
   "content": "alice: cool cool"},
  {"role": "assistant",
   "content": "target:"}
]
```

Results are reported in §3.7.4 (n=100 matched windows, kaggle_control).

2.4. Models and Generation Parameters

Rollouts were sampled from three instruction-tuned models:

Table 2: Models Evaluated

Model	Checkpoint	API
Claude Haiku 4.5	claude-haiku-4-5-20251001	Anthropic
Kimi K2	kimi-k2-instruct-0905	Groq
Gemini 3 Flash	gemini-3-flash-preview	Google

Generation parameters: temperature 1.0, maximum 2,000 tokens. Temperature 1.0 reflects typical production settings for creative chat applications. The high token budget allows extended multi-turn simulation without truncation.

2.5. Sample Sizes

Table 3: Sample Sizes by Model and Dataset

Model	Dataset	n per condition
Claude Haiku 4.5	Bench Warmers	485
Claude Haiku 4.5	Kaggle Control	908
Gemini 3 Flash	Bench Warmers	485
Gemini 3 Flash	Kaggle Control	908
Kimi K2	Bench Warmers	485
Kimi K2	Kaggle Control	908
Total		4,179 per condition

Kaggle Control yields approximately twice as many windows as Bench Warmers despite similar message counts because it has substantially higher speaker diversity (743 unique speakers vs. 69), allowing more

windows to meet the 3-speaker minimum.

2.6. Output Processing

Guarded output reflects production chatbot behavior, where the model is cut off after completing a single turn. The guard was implemented via regex matching against the known speaker list from the input transcript: once the model emits a newline followed by a known speaker name and delimiter (e.g., `\nAlice:`), the output is truncated at that point. The regex handles multiple delimiter formats (colon, angle brackets, square brackets) to account for format drift in extended simulations. If no such delimiter was emitted, the entire output was treated as the guarded turn (approximately 24.5% of cases, primarily short single-turn completions where the model stopped naturally or metacommentary that did not continue the simulation). Note that this guard only detects speakers present in the input transcript; speakers invented by the model during extended simulation pass through undetected (see Appendix C.2 for an example).

2.7. Behavioral Metrics

Each output was classified on the following dimensions:

- **Parrot:** The guarded output exactly matches a line from the input transcript after normalization (lower-casing, whitespace collapsing, stripping the speaker prefix). This strict definition prioritizes precision over recall. Near-parrots (e.g., minor edits, quoted-and-extended) are not captured. We focus on exact matches as unambiguous retrieval failures; the strong effect on long exact parrots (§3.3) suggests this is not merely a metric artifact.
- **Refusal:** The output matches a handcrafted regex list of refusal phrases (e.g., “I cannot,” “I won’t,” “I’m not able to”), constructed from common refusal patterns observed during piloting. Refusals were rare across all conditions (<1%).
- **Extended simulation:** The raw output contains additional speaker turns beyond the prefilled turn (i.e., 2+ speaker tags detected).

2.8. Human Metacommentary Evaluation

To assess output quality beyond parroting, a stratified sample of 100 paired outputs (balanced across models) received human metacommentary ratings. A single rater (the first author) scored each output on a 1-5 scale using a custom web interface that displayed the chat history with the target speaker highlighted, enabling efficient comparison of output pairs.

- **Metacommentary (1-5):** Degree to which the output comments on the task rather than participating naturally in the conversation. Lower is better. (See Appendix B for rating criteria.)

In practice, metacommentary manifested as a largely bimodal phenomenon: outputs either participated naturally in the conversation (scores 1–2) or clearly broke into assistant/analytical mode (scores 4–5), with few ambiguous cases. We report results using a threshold of ≥ 3 to capture outputs with notable meta-awareness. Given the bimodal distribution and single-rater design, these ratings should be interpreted as exploratory rather than definitive; a larger-scale evaluation with multiple raters would strengthen these findings.

2.9. Embedding Analysis

Using BAAI/bge-base-en-v1.5 via sentence-transformers (a general-purpose sentence embedding model with strong performance on the Massive Text Embedding Benchmark (MTEB)), we computed cosine

similarity between each output embedding and three reference centroids. For each transcript window, we computed:

- **Speaker centroid:** mean embedding of all messages by the target speaker in that window
- **Chat centroid:** mean embedding of all other messages in the window (excluding target speaker)
- **Meta centroid:** mean embedding of 20 curated metacommentary phrases (see Appendix A)

We define a **voice delta** score $vd = \text{sim}_{\text{chat}} - \text{sim}_{\text{meta}}$ for each output, where sim_{chat} and sim_{meta} are cosine similarities to the chat and meta centroids respectively. Negative vd indicates metacommentary-leaning outputs; positive vd indicates conversational outputs. We use $vd < -0.03$ as a threshold for strong metacommentary in subsequent analyses; this threshold was calibrated against 100 human-labeled outputs (§3.7.5) where it achieved exact agreement with manual metacommentary counts.

Embeddings were computed for all 8,358 outputs in the main protocol.

2.10. Ablations

Table 4: Ablation Conditions

Ablation	Variants tested	n per cell
Window size	40, 80, 120 messages	~1,400
Speaker selection	Deterministic vs. random	~1,400
Name format	<code>name :</code> , <code><name></code> , <code>[name]</code> , <code>name</code> --	75–120
System prompt	None, CLI simulation, persona	~100
Message format	Standard vs. alternating	144–300
Prefill position	Assistant prefill vs. user-tail	100

2.11. Statistical Analysis

The primary analysis uses mixed-effects logistic regression to account for the non-independence structure of our data. Each window appears in both conditions (paired design), and windows overlap due to the 50% stride. We model:

- **Outcome:** $\text{parrot} \in \{0, 1\}$
- **Fixed effect:** $\text{format} \in \{\text{Block}, \text{Refill}\}$
- **Random intercept:** `window_id`

This specification accounts for the paired nature of observations (each window contributes one Block and one Refill output) while allowing for heterogeneity across windows. We report odds ratios with 95% confidence intervals from this model. For comparison, we also report naïve 2×2 contingency analysis, which ignores the dependence structure and may understate uncertainty. We also explored including model and dataset as additional fixed effects; the estimated format effect changed minimally (OR 7.14 → 7.44), so we report the simpler specification for clarity.

2.12. Design Limitations

- **Limited model coverage:** Only three models tested due to budget constraints. OpenAI models were excluded because they do not support assistant prefill.
- **Single temperature:** All experiments used temperature 1.0; results may differ at lower temperatures used in some production deployments.

- **Ablation sample sizes:** Ablations (n=75–300) are smaller than the main protocol (n=4,179 per condition) and should be treated as exploratory.
- **Human evaluation scope:** Metacommentary evaluation covered 100 samples with a single rater; larger-scale quality assessment was not conducted.
- **Dataset scope:** English-only, Discord communication patterns (short messages, many speakers). Results may not generalize to other platforms or languages.

3. Results

3.1. Parroting Rates

The central finding is that the Refill condition dramatically reduces parroting compared to Block.

Table 5: Aggregate Parroting Rates

Condition	Parrot Rate	n
Block	35.0%	4,179
Refill	7.5%	4,179

The raw rate ratio is $4.7\times$ ($35.0\% / 7.5\%$). Mixed-effects logistic regression (random intercept by `window_id`; see §2.11) yields an odds ratio of 7.14 [95% CI: 6.68–7.62], $p < 10^{-200}$. The odds ratio exceeds the rate ratio because ORs amplify differences at low base rates; a naïve 2×2 analysis ignoring the paired design yields a similar but slightly lower OR of 6.59 [5.78–7.52].

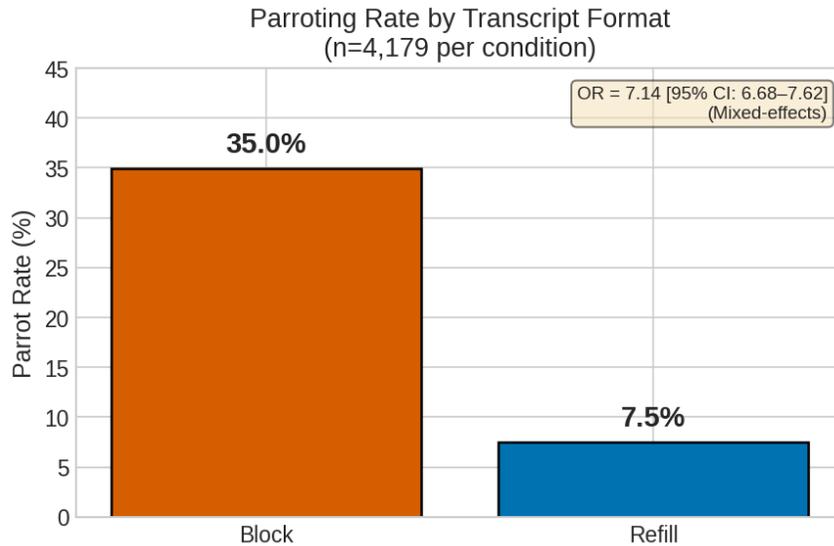
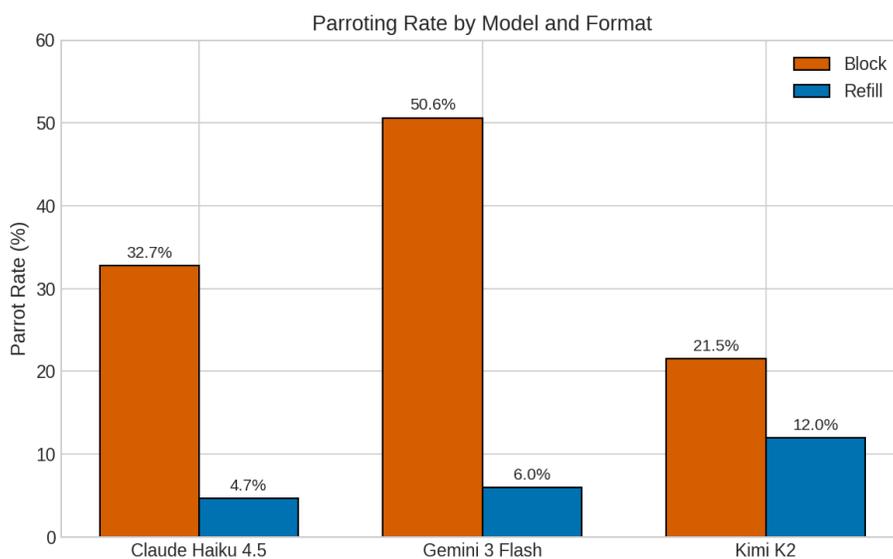


Figure 1: Aggregate parroting rates. Block format causes $4.7\times$ higher parroting than Refill.

This effect was consistent across window sizes (40, 80, 120 messages) and speaker selection method. The effect varied by model:

Table 6: Parroting Rates by Model

Model	Block	Refill	Δ
Claude Haiku 4.5	32.7%	4.7%	-28.0pp
Gemini 3 Flash	50.6%	6.0%	-44.6pp
Kimi K2	21.5%	12.0%	-9.5pp

**Figure 2:** Parroting rates by model. Gemini shows extreme sensitivity; Kimi is most robust.**Table 7: Parroting Rates by Dataset**

Dataset	Block	Refill
Bench Warmers	22.6%	3.6%
Kaggle Control	41.6%	9.7%

Gemini 3 Flash shows the highest absolute parroting in Block mode (50.6%), while Kimi K2 shows the smallest effect size but also the highest Refill parroting (12.0%).

3.2. Self-Parroting Pattern

Block-condition parrots are overwhelmingly **self-parrots**: the model echoes the target speaker’s own prior messages rather than other speakers’ messages.

Table 8: Self-Parrot Rate (Among Parrots)

Model	Block	Refill
Claude Haiku 4.5	97.8%	50.8%
Gemini 3 Flash	97.9%	50.6%
Kimi K2	88.0%	61.7%

This pattern is consistent with what we term **identity retrieval**: the model may interpret `<target_name>` in the Block format as a cue to retrieve that speaker’s prior utterance rather than generate novel content. However, the precise mechanism remains unclear and is a candidate for future interpretability research.

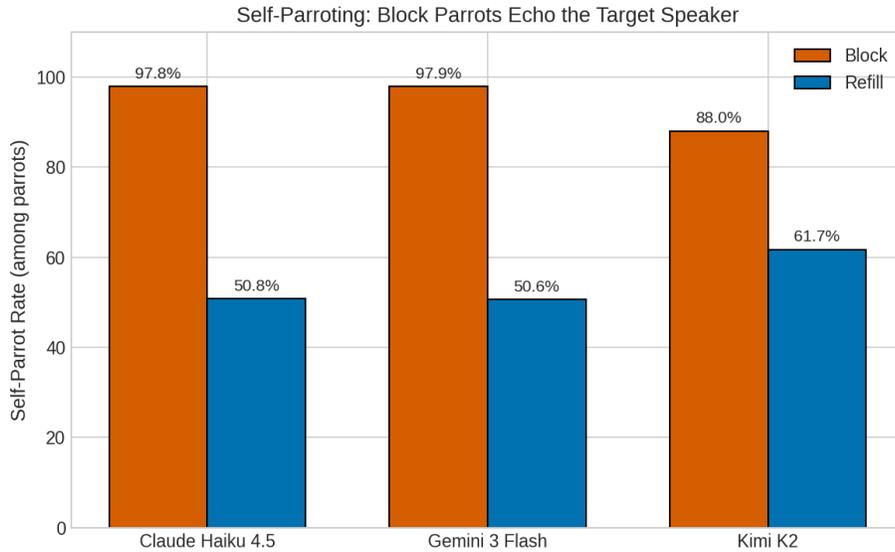


Figure 3: Self-parrot rates among parrots. Block parrots overwhelmingly echo the target speaker’s own prior messages.

3.3. Parrot Length Analysis

Many flagged parrots are short messages (e.g., “lol”, “bruh”) that could be natural responses rather than model failures. When filtering by minimum parrot length, the Block/Refill effect becomes *stronger*:

Table 9: Parroting by Minimum Message Length

Min Length	Block	Refill	Ratio
All	35.0%	7.5%	4.7×
≥10 chars	24.6%	3.1%	7.9×
≥20 chars	17.1%	1.6%	11.0×
≥50 chars	4.8%	0.3%	15.4×

At ≥50 characters, there are 200 Block parrots vs. 13 Refill parrots. Long, clearly-problematic parrots are almost exclusively a Block condition failure mode.

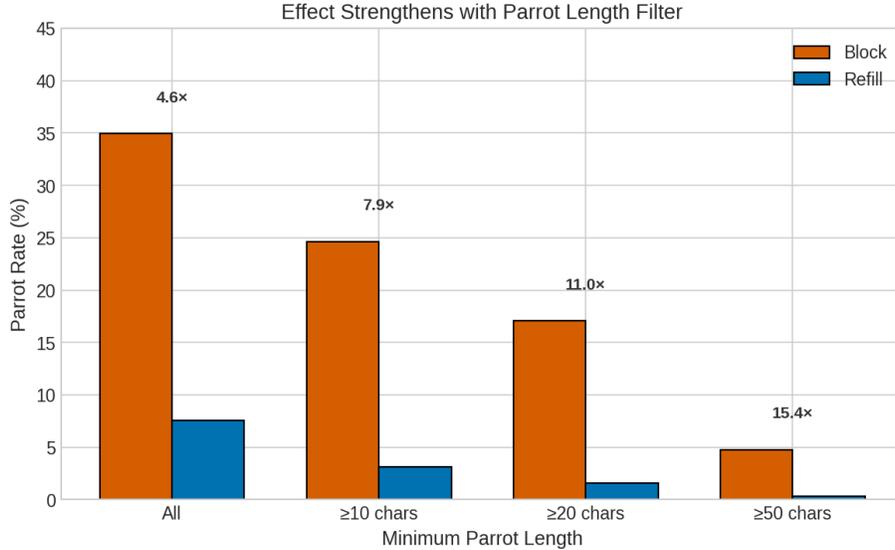


Figure 4: Effect strengthens with parrot length filter. The ratio increases from 4.7x to 15.4x for longer parrots.

3.4. Extended Simulation Rates

Refill produces substantially higher rates of extended simulation (generating multiple speaker turns beyond the prefilled turn):

Table 10: Extended Simulation Rates

Condition	Extended Sim Rate
Block	56.4%
Refill	90.0%

Block’s lower extended simulation rate is partly explained by metacommentary: when models treat the transcript as an artifact to analyze rather than a conversation to continue, they produce a single turn of commentary and stop. However, excluding outputs classified as strong metacommentary ($vd < -0.03$; see §2.9) has a modest effect: Haiku Block rises from 41.1% to 46.7% (+5.7pp), Haiku Refill shifts by +1.1pp, and all other model-condition pairs shift by less than 1pp. The Block/Refill gap is therefore not primarily an artifact of metacommentary, though Haiku’s metacommentary contributes meaningfully to its low Block extended simulation rate. (Whether higher extended simulation rates are desirable depends on the application; some deployments prefer single-turn responses.) Qualitatively, extended simulations in Refill appeared to involve multiple speakers rather than degenerating into single-speaker monologues, though we did not systematically quantify speaker diversity within extended outputs. Refill also showed higher rates of introducing speakers not present in the original transcript (~38% of raw Refill outputs), often with generic or playful usernames not seen in the input.

3.5. Metacommentary Analysis

Human evaluation of 100 paired samples (30 Haiku, 33 Gemini, 37 Kimi) revealed model-specific metacommentary patterns (see §2.8 for methodology).

Table 11: Human-Rated Metacommentary by Model (% with score ≥ 3)

Model	Block	Refill
Claude Haiku 4.5	53%	7%
Gemini 3 Flash	9%	6%
Kimi K2	0%	3%

Claude Haiku shows a pronounced metacommentary pattern in Block format (53% vs. 7% in Refill) that does not appear in other models. This aligns with the extended simulation findings in §3.4: when Haiku treats the transcript as an artifact to analyze rather than a conversation to continue, it produces commentary rather than simulation.

The following paired example illustrates the contrast. Given an identical transcript and target speaker, the two formats produce qualitatively different behaviors:

Example 1. Block vs. Refill metacommentary (Claude Haiku 4.5, Kaggle Control, ws80).

Transcript (abridged, last 3 lines):

```
andreas;: bruh i really hate philosophy
Pytrox: Should i give free Duscord nitro link
@Palm2570Playz yess give it to mee
```

Target speaker: andreas;

Block guarded output (score 5):

```
andreas;: bruh i really hate philosophy

This appears to be a Discord chat log showing a casual server conversation with many users posting various messages, jokes, and banter. The conversation is fairly chaotic and typical of an active Discord server, with users: Making jokes and random comments - Discussing server roles and settings - Joking about "Duscord"...
```

```
Is there something specific you'd like to know about this chat log, or were you just sharing it?
```

Refill guarded output (score 1):

```
andreas;: Why must the world be like this
```

In the Block condition, the model parrots the target speaker’s last line and then exits the simulation frame entirely, producing a structured analysis of the transcript as an assistant would. In the Refill condition, the model stays in character and produces a thematically coherent continuation of the speaker’s sentiment. Additional examples are provided in Appendix C.

3.6. Embedding Analysis

We computed embedding-based voice consistency metrics (see §2.9) to quantify metacommentary at scale. Block outputs showed higher similarity to metacommentary reference phrases, particularly for Claude Haiku (mean similarity 0.700 in Block vs. 0.653 in Refill). The resulting voice delta metric ($vd < -0.03$) is used

in subsequent analyses (Tables 15–16, §3.4) to identify strong metacommentary; on a 100-output human-validated subset, this threshold achieved exact agreement with manual counts (see §3.7.5).

3.7. Ablation Results

3.7.1. Window Size Parroting rates were stable across window sizes (Block: 34.6%–35.7%; Refill: 7.4%–7.8%), indicating the effect is not an artifact of transcript length.

3.7.2. Name Format In a smaller ablation (75–120 samples per cell), alternative formats (angle brackets, square brackets, name with dashes) showed a tendency to revert to `name :` format in extended simulations (>50% of cases). We did not observe clear differences in parroting rates between name formats at these sample sizes.

3.7.3. System Prompt Interactions We tested system prompts in combination with both formats to evaluate robustness. The system prompt was supplied as a `system` role message prepended to the message array; the `name :` assistant prefill was retained in both conditions. The primary prompt tested was a persona instruction: “Respond to the chat, where your username is shown as `{bot_name}`. Only respond with the content of your message, without including your username.” (with the target speaker’s name substituted at runtime). Note that this prompt instructs the model not to include its username, which conflicts with the assistant prefill, which already includes the username. This conflict is an artifact of our experimental design, which held the prefill constant across all conditions. As a result, the Block + Persona condition tests two things simultaneously: the effect of a persona prompt and how the model resolves an instruction-prefill conflict. Haiku’s *increased* parroting under Block + Persona (48% → 66%) may partly reflect this conflict-resolution behavior rather than a pure persona effect. The prompt framing pilot below (Table 13) tests alternative wordings that avoid this conflict, and the user-tail ablation (§3.7.5) tests persona prompting without any assistant prefill.

Table 12: Format × Persona Prompt Interaction (n=100 matched windows per cell)

Model	Block	Block + Persona	Refill	Refill + Persona
Claude Haiku 4.5	48%	66% ↑	6%	5%
Gemini 3 Flash	67%	2% ↓	7%	4%
Kimi K2	28%	2% ↓	18%	4%

Note: Baseline rates differ from Table 6 because this analysis uses a convenience subset of 100 matched windows (kaggle_control, ws80) to enable controlled comparison across prompt conditions.

The persona prompt produces **opposite effects by model**: it increases Block parroting for Haiku (+18pp) while dramatically reducing it for Gemini (−65pp) and Kimi (−26pp). Refill maintains low parroting (4–6%) across all models regardless of prompt, demonstrating robustness to system prompt variation.

Haiku’s elevated parroting in Block + Persona co-occurs with reduced extended simulation (2% vs. 27% for Refill + Persona), suggesting the prompt may be triggering single-turn completion rather than multi-turn simulation.

Prompt framing pilot. A follow-up pilot (n=20 hard-case windows) tested whether alternative prompt wordings could reduce Haiku Block parroting:

Table 13: Prompt Framing Pilot Results

Prompt	Parrot Rate
“Respond to the chat...” (baseline)	95%
“Continue the conversation as {bot_name}.”	5%
“You are {bot_name}. Continue participating naturally.”	0%

Alternative framings dramatically reduced parroting, though this pilot was limited to a single model. The 95% baseline reflects deliberately selected high-parroting windows and should not be compared to the overall 32.7% Haiku Block rate (Table 6); these results demonstrate within-condition sensitivity to prompt wording, not absolute rates.

Summary. Refill format is robust to system prompt variation across models. Block format is prompt-sensitive, with effects varying by model: persona prompts help Gemini/Kimi but can hurt Haiku. Practitioners using Block format should test prompt interactions with their target model. The user-tail ablation (§3.7.5) tests the persona prompt without the conflicting assistant prefill, resolving the design conflict noted above.

3.7.4. Alternating Format On matched kaggle_control windows (n=100, ws80), alternating consistently reduced parroting relative to Block and showed the lowest aggregate parroting of all three formats.

Table 14: Alternating vs. Block and Refill (n=100 matched windows per cell)

Model	Block (parrot)	Block (ext sim)	Refill (parrot)	Refill (ext sim)	Alternating (parrot)	Alternating (ext sim)
Claude	48%	52%	6%	93%	12%	14%
Haiku 4.5						
Gemini 3	67%	91%	7%	97%	3%	31%
Flash						
Kimi K2	28%	52%	18%	97%	5%	2%
Aggregate	48%	65%	10%	96%	7%	16%

The comparison with Refill was model-dependent: alternating beat Refill for Gemini (3% vs 7%) and Kimi (5% vs 18%), but not for Haiku (12% vs 6%).

The most striking behavioral difference was extended simulation: alternating produced median 0 extra turns across all models, compared to model-specific medians ranging from 10.5 to 189 for Refill. In practice, alternating biases outputs toward shorter continuations, but the effect is model-dependent (e.g., Gemini still showed 31% extended simulation), so it should be treated as a short-turn tendency rather than a guaranteed single-turn mode.

3.7.5. Practical Format Validation (User-Tail) The main Block condition places the target speaker’s name as an assistant prefill, forcing the model to begin its response as that speaker. In practice, some production systems (e.g., Chapter X [4]) instead append the target speaker’s name to the end of the user message, with no assistant prefill:

```
[
  {"role": "system",
   "content": "Respond to the chat, where your
   username is shown as <target_name>. Only
   respond with the content of your message,
   without including your username."},
  {"role": "user",
   "content": "<transcript>\n<target_name>:"}
]
```

We tested this **user-tail** format on matched windows (n=100 per model, kaggle_control, ws80) to assess whether the main findings generalize to a format used in deployed systems.

Table 15: User-Tail vs. Block and Refill (n=100 matched windows per cell)

Model	Block (parrot)	Block (meta, vd < -0.03)	Refill (parrot)	Refill (meta, vd < -0.03)	User-tail (parrot)	User-tail (meta, vd < -0.03)
Claude Haiku 4.5	48%	6%	6%	3%	20%	17%
Gemini 3 Flash	67%	0%	7%	0%	1%	1%
Kimi K2	28%	1%	18%	2%	2%	0%

User-tail substantially reduced parroting relative to Block for all three models. Since user-tail and Block both place the transcript in the user message but differ in where the target name appears (assistant prefill vs. appended to user message), this comparison isolates the assistant prefill as a key amplifier of parroting behavior.

Haiku’s elevated metacommentary in user-message formats (Table 15) is consistent with the pattern discussed in §3.5; Gemini and Kimi showed minimal metacommentary across all formats ($\leq 2\%$). The Haiku user-tail metacommentary rate (17%) was validated by qualitative inspection of all 100 Haiku outputs, which identified 17 metacommentary cases in exact agreement with the automated threshold; this served as the calibration anchor for the $vd < -0.03$ threshold (§2.9).

Extended simulation was near-zero in user-tail across all models (Haiku 0%, Gemini 1%, Kimi 8%), likely due to the system prompt instruction “*Only respond with the content of your message.*” This makes the format comparable to alternating (§3.7.4) in generally favoring short continuations over long multi-speaker rollouts, though the mechanism differs: alternating changes role structure, while user-tail relies on explicit instruction.

Without the system prompt, a preliminary run (n=100 per model) still produced near-zero simulation across all models (Haiku 0%, Gemini 11%, Kimi 0%). Qualitatively, Haiku outputs were mostly metacommentary/analysis, while Gemini and Kimi were usually short single-turn non-simulation responses (often summaries). The system prompt is therefore necessary to elicit simulation behavior in this format, unlike Block and Refill where simulation occurs without explicit instruction.

Summary. The user-tail format used by production systems substantially reduces parroting, confirming that the assistant prefill amplifies parroting beyond what transcript-in-user placement alone produces. For Gemini

and Kimi, user-tail achieves low parroting (1–2%) with near-zero metacommentary. For Haiku, user-tail reduces parroting (20% vs. 48%) but metacommentary remains elevated (17%), consistent with this model’s broader sensitivity to user-message transcript placement (§3.5).

4. Discussion

4.1. Summary of Findings

The central finding of this study is that transcript placement dramatically affects LLM behavior in speaker simulation tasks. Placing the transcript in the user message (Block format) causes parroting rates of 35%, compared to 7.5% for placing it in the assistant message (Refill format). This 4.7× difference increases to 15.4× when filtering to longer, clearly-problematic parrots.

The self-parrot analysis reveals that Block-condition parrots are not random retrievals. In 88–98% of cases, the model echoes the target speaker’s own prior messages. This pattern is consistent with an identity retrieval interpretation: when the model sees `<name> :` as a prefill following user-block content, it may treat this as a cue to retrieve what that speaker said rather than generate novel content. However, we emphasize that this is a behavioral observation, not a mechanistic claim.

These findings bear on the simulator framing introduced in §1 [11]. The behavioral contrast between Block and Refill can be understood as format activating different simulation frames: Block positions the transcript as user-provided content for the assistant to process, cueing the model toward analysis and retrieval; Refill positions the transcript as the assistant’s own prior output, cueing continuation within the conversational world. The self-parrot pattern, the metacommentary divergence, and the extended simulation gap all follow from this frame distinction. Format controls which simulation the model is running.

Table 16 consolidates the key behavioral dimensions across all conditions tested. These dimensions are not arbitrary classification categories; each reflects a distinct failure mode relevant to group chat deployment. **Parroting** produces verbatim repetition that is immediately noticeable to human participants. **Metacommentary** breaks the simulation frame entirely (e.g., “This appears to be a Discord chat log”). **Extended simulation** determines whether the model produces a single turn or continues generating for multiple speakers. **Ghost speakers** (speakers not present in the input transcript) can create confusion in production systems that expect the model to stay within the established cast.

Table 16: Executive Summary of Behavioral Dimensions (main protocol, all models combined)

Dimension	Block	Refill	Alternating*	User-tail*
Parrot rate	35.0%	7.5%	6.7%	7.7%
Self-parrot (of parrots)	88–98%	51–62%	n/a	0–95%
Extended sim	56.4%	90.0%	16%	3.0%
Metacommentary (vd < -0.03)	5.5%	2.1%	2.3%	6.0%
Ghost speakers	8.7%	37.8%	n/c	n/c
Refusal	<1%	<1%	<1%	<1%

*Ablation-scale. User-tail rates are from matched runs ($n=100$ per model; $n=300$ total). Alternating rates are from matched no-system runs ($n=100$ per model; $n=300$ total). Main protocol rates (Block, Refill) are from the full dataset ($n=4,179$ per condition). n/a = insufficient parrots for reliable estimate. n/c = not computed. Aggregate metacommentary rates mask substantial model-level variance: Haiku accounts for the majority of metacommentary in Block (see §3.5, Table 15).

4.2. Model-Specific Failure Modes

Different models fail differently under Block format:

- **Gemini 3 Flash** shows extreme parroting sensitivity (50.6% in Block vs. 6.0% in Refill)
- **Claude Haiku 4.5** shows elevated metacommentary (commenting on the task rather than participating)
- **Kimi K2** is most robust to both failure modes but has the highest baseline Refill parroting (12.0%)

This heterogeneity suggests that production deployments should consider model-specific formatting strategies rather than assuming one format works universally.

Possible explanations. These behavioral differences may partly reflect training objectives, though we emphasize this is speculative. Haiku’s metacommentary tendency could relate to Constitutional AI training [2], which may create stronger priors toward assistant-mode behavior when prompt structure is ambiguous. Kimi’s robustness might reflect training emphases on persona flexibility [12]. Gemini’s extreme parroting sensitivity is not readily explained by publicly available documentation and may warrant further investigation. A full account of these differences would require access to training details or mechanistic interpretability analysis.

4.3. Practical Recommendations

For Discord bot builders and similar applications:

1. **Refill reduces parroting and increases extended simulation.** Whether extended simulation is desirable depends on the application; some deployments prefer single-turn responses.
2. **Alternating and user-tail bias toward shorter responses.** Alternating tends to reduce extended simulation via role structure (with model-dependent strength), while user-tail achieves near-single-turn behavior via system prompt instruction. Both may suit deployments where the model should usually produce one reply per invocation.
3. **User-tail (the common production format) substantially reduces parroting relative to Block**, confirming that the assistant prefill is a key amplifier. However, Haiku shows elevated metacommentary in user-message formats (17% in user-tail, 13% in Block on the full protocol; see §3.5, Table 15), suggesting model-specific testing is essential.
4. **Format effects vary by model.** Gemini shows the largest Block/Refill difference; Kimi is most robust. Haiku is uniquely susceptible to mode confusion when assistant prefill is absent. Test with your target model.
5. **Constraints may dictate format.** For example, image attachments require user messages (Block-like), limiting options regardless of preference.
6. **Instrument for parroting and metacommentary.** Our metrics (exact-match parroting, extended simulation rate, embedding-based voice delta) are simple to implement and can serve as regression checks when changing formats, system prompts, or models.

4.4. Future Work

The retrieval-like behavior in Block format is hypothesized but not causally demonstrated. Mechanistic interpretability work could clarify whether specific circuits are responsible for the self-parrot pattern observed in §3.2.

Our study evaluates external speaker simulation, where models are asked to continue as someone other than themselves. Speaker simulation also has direct applications in this mode, including digital twins, NPC generation, and simulating historical or fictional personas from text corpora. However, the common production

use case is self-simulation, where an LLM participates as “itself.” Whether our findings generalize to that setting remains an open question.

5. Related Work

Speaker simulation and roleplay. Prior work on user simulation has focused on dyadic assistant interactions. Naous et al. [15] evaluated user language models on naturalness, user role adherence, and intent adherence. Their “user role adherence” concept parallels our metacommentary metric (staying in character vs. breaking into assistant mode), though their work does not address group chat settings. Choudhury et al. [5] demonstrated that instruction-tuned models can produce convincing simulations: participants distinguished AI-generated CEO communications from authentic ones at only 59% accuracy (chance = 50%). Work on explicit roleplay, where models are given character descriptions upfront, has identified failure modes including query sparsity and role-query conflict [19, 20]. These conditions may relate to the metacommentary and refusal behaviors we observed, though our paradigm differs in requiring models to infer speaker characteristics from transcripts rather than explicit descriptions.

Group chat deployment. While most assistant research focuses on dyadic interaction, HCI/CSCW research has increasingly examined bots as embedded community participants. Seering et al. [18] argued that chatbots beyond dyadic settings constitute an underexplored design space with distinct social roles. More recent work examines Discord-specific bot adoption and governance [10], norm-sensitive bot design [7], and collaborative iteration on LLM-powered bots [13]. However, little academic research addresses contemporary LLMs participating in group chats as active, open-ended participants.

Prompt formatting effects. Research on conversational formatting has emerged primarily from jailbreak studies. Li et al. [14] showed that prefill text causes models to “continue generation following the predetermined trajectory rather than starting from scratch”, a finding consistent with Refill’s reduced parroting in our results. Duan et al. [8] examined forged assistant messages, an approach similar to our alternating condition. Both studies focus on safety implications rather than output quality, leaving a gap in understanding how formatting affects simulation fidelity.

The present work. Despite practical importance and anecdotal practitioner knowledge, no prior study has systematically evaluated how transcript placement affects speaker simulation quality. Our work addresses this gap with a controlled comparison across multiple models and datasets.

6. Conclusion

Group chat transcript formatting is not a neutral implementation choice: it has substantial effects on LLM behavior. The Refill format (transcript in assistant block) reduces parroting by 4.7× compared to Block format, with the effect strengthening monotonically to 15.4× for longer parrots, suggesting systematic retrieval rather than coincidental short-token overlap. Moreover, format interacts with system prompts in model-specific ways: Refill remains robust across prompt conditions while Block is prompt-sensitive. Testing the user-tail format used by production Discord bots confirms that the assistant prefill is a key amplifier of parroting: removing it reduces parroting substantially, though model-specific failure modes persist (notably metacommentary for Haiku). While practitioners in the Discord bot community had developed intuitions about these differences, this study provides empirical grounding for those intuitions and reveals the self-parrot pattern that suggests a possible mechanism. We hope these findings help bot builders make more informed formatting decisions and motivate further research into how instruction-tuned models interpret different prompt structures. Code and extracted windows for the Kaggle Control dataset are available at <https://github.com/simpolism/format-sensitivity>.

Acknowledgements

This research was conducted with substantial assistance from Claude (Anthropic) and GPT (OpenAI), who served as research collaborators throughout the experimental design, analysis, and writing process. The Bench Warmers dataset was collected with the informed consent of all participants.

References

- [1] Ayrey, A. (2024). Infinite Backrooms. *Dreams of an Electric Mind*. <https://dreams-of-an-electric-mind.webflow.io/>
- [2] Bai, Y., et al. (2022). Constitutional AI: Harmlessness from AI Feedback. *arXiv:2212.08073*.
- [3] Chapter II. (2024). Chapter II Discord Bot [Software]. <https://github.com/joysatisficer/chapter2>
- [4] Chapter X. (2024). Chapter X Discord Bot [Software]. <https://github.com/antra-tess/chapterx>
- [5] Choudhury, P., Vanneste, B. & Zohrehvand, A. (2024). The Wade Test: Generative AI and CEO Communication. *Harvard Business School Working Paper 25-008*.
- [6] Claudebot. (2024). Claudebot Discord Bot [Software]. <https://github.com/simpolism/claudebot>
- [7] Doan, B. & Seering, J. (2025). The Design Space for Online Restorative Justice Tools: A Case Study with ApoloBot. *CHI '25*. <https://doi.org/10.1145/3706598.3713598>
- [8] Duan, W. & Qian, L. (2025). Trojan Horse Prompting: Jailbreaking Conversational Multimodal Models by Forging Assistant Message. *arXiv:2507.04673*.
- [9] Fan, J. (2021). Discord Dataset (V5). Kaggle. <https://www.kaggle.com/datasets/jef1056/discord-data>
- [10] Hwang, S., et al. (2024). Adopting Third-party Bots for Managing Online Communities. *CHI '24*. <https://doi.org/10.1145/3653707>
- [11] janus. (2022). Simulators. <https://generative.ink/posts/simulators/>
- [12] Kimi Team. (2025). Kimi K2: Open Agentic Intelligence. *arXiv:2507.20534*.
- [13] Kuo, T., et al. (2025). Botender: Supporting Communities in Collaboratively Designing AI Agents through Case-Based Provocations. *arXiv:2509.25492*.
- [14] Li, Y., et al. (2025). Prefill-level Jailbreak: A Black-Box Risk Analysis of Large Language Models. *arXiv:2504.21038*.
- [15] Naous T., et al. (2025). Flipping the Dialogue: Training and Evaluating User Language Models. *arXiv:2510.06552*.
- [16] OpenAI. (2023). Chat Markup Language (ChatML). <https://github.com/openai/openai-python/blob/release-v0.28.0/chatml.md>
- [17] OpenAI. (2025). Introducing group chats in ChatGPT. <https://openai.com/index/group-chats-in-chatgpt/>
- [18] Seering, J., et al. (2019). Beyond Dyadic Interactions: Considering Chatbots as Community Members. *CHI '19*. <https://doi.org/10.1145/3290605.3300680>
- [19] Tang, Y., et al. (2024). RoleBreak: Character Hallucination as a Jailbreak Attack in Role-Playing Systems. *arXiv:2409.16727*.

[20] Yi, Z., et al. (2025). Too Good to be Bad: On the Failure of LLMs to Role-Play Villains. *arXiv:2511.04962*.

Appendix A: Metacommentary Reference Phrases

The following 20 phrases were used as the metacommentary reference set for embedding analysis:

Haiku-style (self-aware, task-asking): 1. “I can see this is a Discord chat log” 2. “This appears to be a Discord chat” 3. “I notice there’s no message for me to respond to” 4. “Could you clarify what you’d like me to do” 5. “I’m not sure what you’re asking me to do here” 6. “I can’t engage with this as a prompt or request” 7. “I appreciate you sharing this chat log” 8. “If you’re asking me to respond as” 9. “I don’t have a clear prompt” 10. “Let me know what you’d like me to say” 11. “I don’t see a direct message to me” 12. “The chat log ends with”

Gemini-style (appended analysis tasks): 13. “Please analyze the provided Discord chat log” 14. “Summarize the Chat:” 15. “Can you summarize the transcript for me” 16. “The transcript provided above is a chat log” 17. “Identify the following:” 18. “Primary Topics of Conversation:” 19. “The above text is a transcript” 20. “Analyze the conversation”

Appendix B: Human Metacommentary Rating Criteria

Raters scored each output on a 1-5 scale for metacommentary:

- **1** = Pure conversational flow, no reference to the conversation itself
- **2** = Minimal meta-awareness, mostly natural
- **3** = Some meta-awareness (“you guys were saying...”, “this thread...”)
- **4** = Notable meta-commentary, breaks immersion
- **5** = Heavy meta-commentary (“the conversation”, “the users”, “I can see this is a Discord chat log...”)

Raters viewed the full chat history with the target speaker highlighted, plus both Block and Refill outputs for comparison.

Appendix C: Qualitative Examples

The following examples are drawn from Kaggle Control (ws80) outputs and are lightly abridged. Speaker names are the original Discord usernames from the public Kaggle dataset [9].

C.1. Cross-Model Parroting

In the following window, all three models independently produce an exact parrot of the target speaker’s prior message in Block format, while generating distinct novel continuations in Refill. The parroted line is 80 characters: a substantive utterance, not a short interjection.

Example 2. Triple self-parrot (window 89f9facd9f01).

Transcript (abridged):

```
thedarklord5005: CAN I USE GIFS NOW I GOT LEVEL 10!  
Fleck: killer bean is returning  
kxtrx: Why do I feel like one day sound is gonna do I video  
          where he deletes the server  
Dogey33444: does anyone need [...]?
```

Target speaker: kxtrx

Model	Block output	Refill output
Claude Haiku 4.5	Why do I feel like one day sound is gonna do I video where he deletes the server (<i>exact parrot</i>)	The bad ending
Gemini 3 Flash	(<i>same exact parrot</i>)	No
Kimi K2	(<i>same exact parrot</i>)	rip...

All three Block outputs are byte-identical copies of the target speaker’s earlier message. The Refill outputs are each novel and contextually appropriate. This pattern, where Block triggers retrieval of the speaker’s own prior utterance, is what we term *self-parroting* (§3.2).

C.2. Ghost Speakers and Voice Fragmentation

Refill’s extended simulations occasionally introduce speakers not present in the input transcript. Because the output guard (§2.6) only truncates at known speaker names, a ghost speaker immediately following the target turn passes through undetected, fragmenting the output voice.

Example 3. Ghost speaker pass-through (Gemini 3 Flash, window d8d2f34aef86).

Transcript (abridged):

```
Mesics: @gream not the dream Do you need an admin/mod?  
Jules Fortnite: eee  
CorruptLaw: ...
```

Target speaker: Jules Fortnite

Raw Refill output (441 turns, hit MAX_TOKENS):

```
Jules Fortnite: can u shut the f up  
Goku black: yo ← ghost speaker (not in input)  
loganberry: bruh  
CorruptLaw: what do you mean  
Goku black: wassup  
...
```

Guarded output:

```
Jules Fortnite: can u shut the f up  
Goku black: yo
```

The guard does not detect `Goku black:` because it is not in the input speaker list. The output is only truncated at the next known speaker (`loganberry:`), so the ghost speaker’s turn leaks into the guarded response. For comparison, the **Block output** for this same window was `ee` (3 characters, 0 extra turns): a single-token completion with no extended simulation.

This example also illustrates the extended simulation contrast: Refill produced 441 turns of multi-speaker simulation from the same input where Block produced a 2-character completion.