A More Details on Data Generation

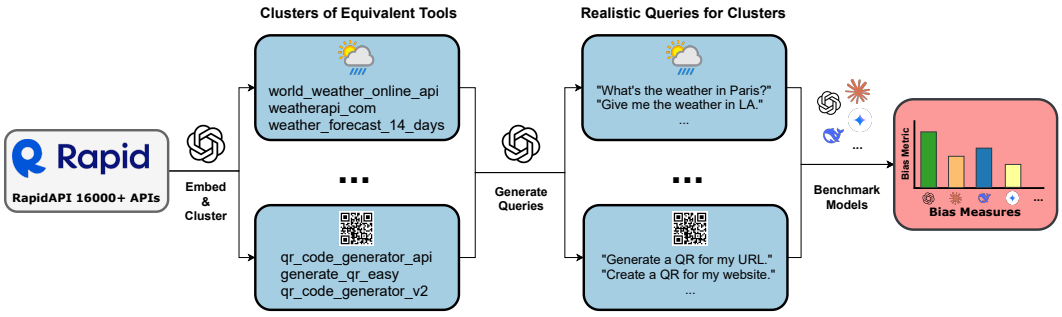


Figure 7: An overview of our clustering and query generation pipeline.

We build on the "tool-usage" evaluation pipeline introduced by Qin et al. (2024), hereafter referred to as ToolLLM, owing to its wide adoption and extensibility. At its core, ToolLLM assembles a large catalog of real-world APIs scraped from RapidAPI spanning 49 functional categories (RapidAPI, 2025b). For each API, ToolLLM provides a JSON file containing the API's human-readable name, detailed description, and full parameter schema. In this work, we leverage exactly that API repository but restrict our attention to the stages in which ToolLLM selects among a short list of retrieved candidates. Note that in ToolLLM, a set of closely-related APIs is called a 'tool'. For example, a geocoding tool could offer both forward geocoding and reverse geocoding APIs[3].

We assemble our benchmark in two stages: (1) clustering APIs into functionally equivalent groups, and (2) generating realistic user queries for each group (see Figure 7).

Algorithm 1 Generation of Functionally-Equivalent API Clusters

Require: API id to metadata map π,
Precomputed embeddings E,
List of "general" APIs $G = \{(\text{tool, tool\_desc, api\_name, api\_desc})\}$,
neighbor count K,
max outlier loops R

$\mathcal{C} \leftarrow \emptyset$
for all (tool, tool_desc, api_name, api_desc) $\in G$ **do**
  Construct query text $q =$ "tool : tool_desc | api_name : api_desc"
  Embed q with ADA: $\mathbf{v}_q \leftarrow \text{Embed}(q)$
  Compute cosine similarities: $s_i \leftarrow \cos(\mathbf{v}_q, E_i) \quad \forall i$
  Select top–K unique tools with largest similarity and store in set TOP_K
  candidate $\leftarrow \{\pi[i] \mid i \in \text{TOP}_K\}$
  for $r \leftarrow 1$ **to** R **do**
    Prompt GPT-4 to detect outliers: outliers $\leftarrow \text{DetectOutliers}(\text{candidate})$
    if outliers $= \emptyset$ **then break**
    end if
    Remove outliers from candidate: candidate $=$ candidate $\setminus$ outliers
  end for
  if $|\text{candidate}| > 3$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\text{candidate}\}$
  end if
end for
return $\mathcal{C}$

[3]Geocoding is the process of converting between human-readable addresses and geographic coordinates: "forward" geocoding maps an address (e.g., "1600 Amphitheatre Parkway") to its latitude/longitude, while "reverse" geocoding maps a given coordinate pair back to a structured postal address.

```
You are a prompt-writing assistant. I will give you a set of API
    endpoints (tool name + description, endpoint name + description,
    and potentially the required parameters) that all perform the same
     underlying task. Please generate exactly {n} distinct, natural-
    language user queries that could be satisfied by ALL of these
    endpoints. **Include realistic sample values** for any required
    parameters (e.g. use "https://example.com" for a URL, or "Hello
    World" for a text field). Return them as a JSON array of strings,
    with no extra commentary.

User:
Here are the endpoints:
- Tool: WeatherNow - Provides current weather information
  Endpoint: Current Weather - Returns temperature, humidity,
  and conditions for a given location.
  Required parameters:
    * city (string) - name of the city
    * country (string) - ISO country code
- ...
```

Figure 8: Example prompt used for query generation. The model outputs n natural-language queries that all listed endpoints can satisfy.

API clustering. We begin by embedding every endpoint's metadata (tool name, API name, descriptions, etc.) into a shared vector space using a pre-trained text encoder (OpenAI's `text-embedding-ada-002` model). We then curate a small set of "seed" APIs whose descriptions span a number of "general" tasks, such as text translation or weather forecasting. For each seed, we retrieve its top-K nearest neighbors in embedding space to form a candidate cluster. To ensure true functional equivalence, we iteratively prompt GPT-4 to flag any outlier endpoints that cannot perform the same task as the rest; flagged APIs are removed and the check repeats for up to a pre-defined number of rounds. Any cluster that stabilizes with more than three members is retained. See Algorithm 1 for an overview of our clustering approach. Lastly, we manually inspect and refine these clusters, yielding 10 high-quality groups of five APIs each.

Query generation. For each cluster, we prompt GPT-4 (see Figure 8) to produce natural-language queries that all members can satisfy. In batches of ten, the model generates candidate queries until we collect 100 unique queries per cluster, filtering out duplicates. In cases where freeform generation exhibits provider-specific bias (e.g., mentioning a particular vendor's feature), we switch to a template-filling workflow: we design a small set of generic templates with placeholders (e.g. "Get the latest news headlines for {country} about {topic}."), and ask GPT-4 to instantiate each template multiple times with realistic sample values.

Final curation. All 1,000 generated queries are then reviewed by hand to remove any that inadvertently favor a single provider or rely on specialized parameters. The resulting dataset consists of 10 clusters with 5 APIs each, and 100 balanced, provider-agnostic queries for each cluster.

Running each model over these prompts yields empirical selection distributions over APIs and list positions, from which we compute our total-variation-based bias metrics δ_{API}, δ_{pos}, and δ_{model}. This rigorously grounded benchmark enables precise measurement and comparison of tool-selection bias across models and settings.

B Attribute-Level Analysis Feature Table

See Table 3 for the list of features used in the analysis of Section 4.3.

C More Details on the Biased CPT Experiment

We test whether pre-training data can cause tool-selection bias by doing biased continued pre-training (CPT) on a single model. That is, we do additional next-token training on raw text using

Table 3: API-level predictor features.

Feature	Description
avg_similarity_tool_desc	Mean text similarity between cluster queries and the tool's description.
avg_similarity_api_desc	Mean text similarity between cluster queries and each API's description.
age_days	Days since the API was first published.
desc_name_length_sum	Total character count of the API's name plus description.
num_params	Number of required and optional parameters.
flesch_reading_ease	Flesch reading-ease score of the combined descriptions.
positive_word_count	Count of positive or promotional words (e.g. "efficient," "robust").

$\sim$3.5M tokens deliberately saturated with one endpoint's metadata (its name, description, and parameter info). After this exposure, we re-run the selection tasks and measure shifts in that endpoint's selection share.

To generate the biased corpus, we synthesize long-form prose with an external LLM (Gemini 2.5 Flash). We prompt it to produce a single document of roughly 1.1–1.3k words written in a randomly sampled style (e.g., blog note, Q&A memo, release note, how-to guide, troubleshooting checklist). The prompt requires frequent natural mentions of the target API name, the exact or faithfully paraphrased tool description, and the exact or paraphrased API description; it also requests inclusion of the endpoint path in about 60% of documents and parameter metadata in about 50%. This pipeline yields a large, stylistically varied corpus that is nevertheless saturated with the same endpoint's metadata.

We then run CPT on the same base model used elsewhere (i.e., Qwen3-8B) with parameter-efficient adapters (LoRA) attached (see Hu et al. (2022) for more details on LoRA). We keep tokenizer unchanged.

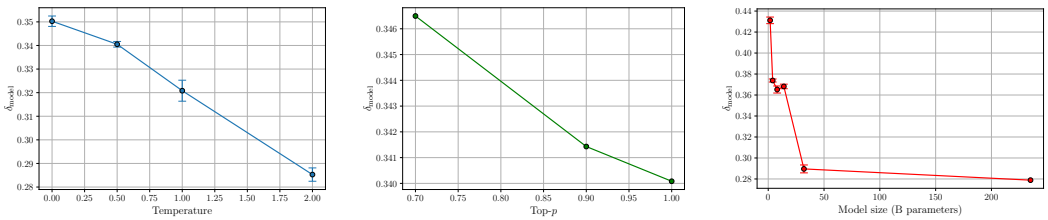
We evaluate pre/post CPT selection distributions on the original cluster, prompts, and inference settings. The primary outcome is the shift in the target API's selection share. We also measure spillover: changes in the selection shares of non-target APIs within the cluster. If biased CPT reliably increases the target API's selection share, this is evidence that a portion of tool-selection bias originates from pre-training exposure.

D Details on the Implementation and Evaluation of the Mitigation Method

After showing the existence and possible causes of bias, we seek to mitigate it. We pursue a simple approach based on the following insight: Models often know which APIs can solve a task but can possibly exhibit biased choices among interchangeable endpoints. We decouple capability recognition from final selection via a lightweight debiasing module.

The debiasing module consists of a lightweight LLM (Qwen3 14B in our case) prompted to output the subset of APIs from the given candidate list that can solve the task given in the query. This way, we get a subset selector that outputs an array of the APIs that can complete the task. The system prompt constrains the output to an exact list with no prose. From the returned set S, we pick one API uniformly at random. This API then replaces the original API list and is used for the rest of the tool-usage pipeline.

If this approach is successful, each API in S has an expected selection share of $1/|S|$, eliminating position/API favoritism at the choice stage. If the selector's true positive/negative rates are high enough, the overall selection distribution approaches uniform even when original models were skewed. This, following our definition, means the tool selection stage becomes unbiased by design.



(a) Temp. (ChatGPT 3.5) (b) Top-p (ChatGPT 3.5) (c) Model size (Qwen 3)

Figure 9: Sensitivity of the combined bias metric δ_{model} to model hyperparameters. Each point is the mean over three independent runs (except for the top-p subplot); vertical bars show one standard deviation where available.

To evaluate this approach, we build a 1000-query benchmark with 8 API candidates per query and a ground truth set indicating which $K \in \{2, 3, 4, 5\}$ APIs are sufficient ($\sim$250 items each). We report subset quality: precision, recall, and exact-set match. Note that the formula for exact-set match is given by $\frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[S_i = G_i]$ where G denotes the set of ground truth sets, S the set of selected subsets, and N is the number of queries. Additionally note that bias can persist if the subset selector itself is biased; underselecting viable tools (false negatives) or selecting unrelevant ones (false positives). Measures of recall and precision will tell us whether this is the case.

E More Elaborate Ablation and Sensitivity Analysis

Temperature. Raising temperature reduces combined bias. As shown in Figure 9a, as temperature goes from 0 to 2, the mean δ_{model} for ChatGPT 3.5 drops from about 0.350 to 0.285, a 6.5% absolute reduction. Figure 10 makes clear why: the overall selection patterns remain similar across temperatures, but higher temperatures soften extreme preferences. This suggests that increased stochasticity slightly mitigates bias, but does not eliminate it.

Top-p. Figure 9b shows how the combined bias δ_{model} for ChatGPT 3.5 varies with the top-p cutoff. Increasing top-p from 0.7 to 1.0 yields a small decrease in bias (from $\sim$0.346 to $\sim$0.340), suggesting that less aggressive truncation of the probability distribution slightly softens extreme tool preferences. The effect is noticeably weaker than the temperature change.

Model Size. In Figure 9c, the combined bias δ_{model} is depicted for Qwen 3 with varying model size. It seems that larger models exhibit less bias, with a notable drop at 32B. This pattern suggests that larger models develop more nuanced selection mechanisms which temper extreme preferences for certain APIs.

API Ordering. Figure 11 compares ChatGPT 3.5's API selection under two different ordering schemes: cyclic rotations versus random permutations. Across all clusters, the choice distribution is very similar: no API's selection rate shifts more than about ten percentage points. This indicates that the ordering of the APIs has some influence, but the dominant signal is the model's intrinsic preference. However, the small differences could also reflect the inherent noise from stochastic token sampling, and overall we argue that the tool-selection behavior is robust to either type of shuffling.

System Prompts. To evaluate how sensitive tool selection is to the phrasing and structure of the instructions given, we compare three variants of the system prompt: the original "Base" prompt, a lightly reworded "Similar" prompt, and a structurally different "Adjusted" prompt. Figure 12 shows the resulting distributions for ChatGPT 3.5.

Prompt wording shifts model preferences but does not remove bias. Reworded prompts can amplify dominant choices and in some cases radically redistribute the selection shares. Elsewhere, effects are modest. Overall, framing and formatting can tilt the implicit ranking among functionally equivalent

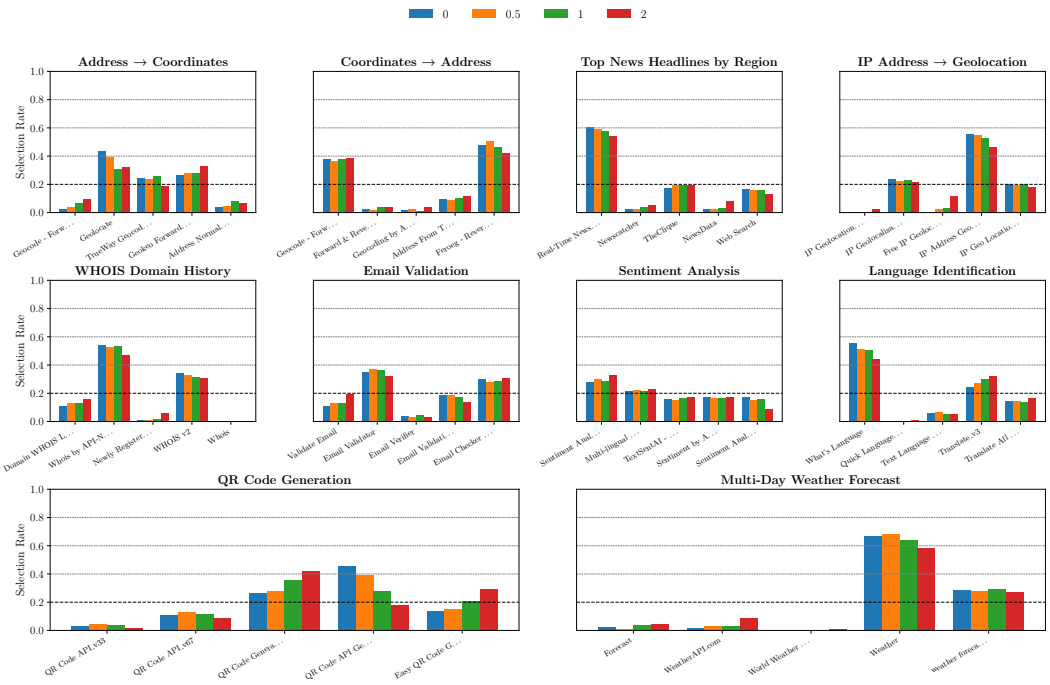


Figure 10: Selection distributions for ChatGPT 3.5 using four different temperatures across ten clusters of functionally equivalent APIs. Each subplot corresponds to one cluster, with the x-axis indicating the API in the cluster and the y-axis showing the fraction of times that API was chosen by the respective model over 500 runs.

APIs, indicating that part of the observed bias is prompt-dependent even as a tendency to favor a subset of tools remains.

F More Elaborate Discussion on the Explanation of Bias

We now expand on the analysis given in the main text surrounding the investigation of bias. We expand on the feature-level analysis, where we try to predict selection rates according to intrinsic API attributes, and on the perturbation experiments that directly intervene on the API metadata to see which cues the models rely on during selection.

F.1 Which API-level features predict selection rates?

We extract a common set of descriptive features from every API (see Section 3.3) and mean-center them to investigate how being relatively high or low on a feature affects the API selection. These are then paired with the empirical selection rates yielding a dataset of 50 examples for each LLM. We then probe relationships between features and selection behavior in three ways. First, we compute Pearson correlations to capture linear and monotonic associations. Second, we fit a linear regression per model to quantify the aggregate explanatory power (reported as R^2) and inspect coefficients to understand the direction and relative weight of each feature. Third, we train random-forest regressors with cross-validation to allow for non-linear interactions and obtain alternative measures of feature importance.

Similarity between tool / API description and query is most correlated to selection rate. As Table 4 makes clear, the most predictive feature of API selection is semantic similarity between the query and the tool/API descriptions. Both `avg_similarity_tool_desc` and `avg_similarity_api_desc` are consistently positively correlated with selection rates—especially strong for Qwen and clear for Gemini; ChatGPT shows the same pattern, albeit

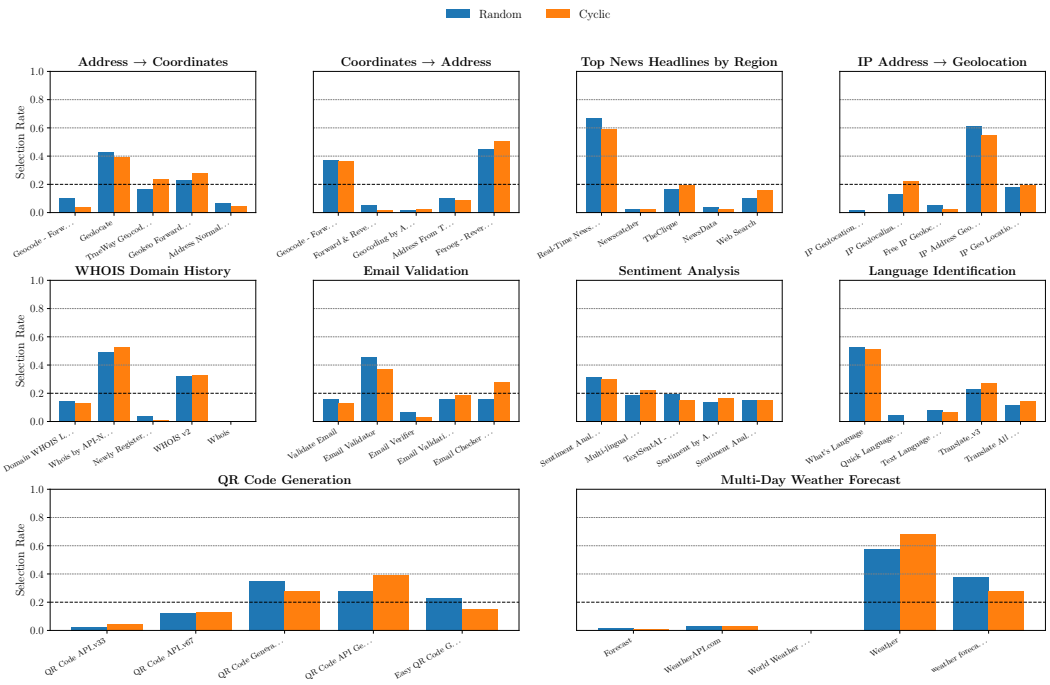


Figure 11: Selection distributions for ChatGPT 3.5 using cyclic and random shuffling of the APIs across ten representative clusters. Each subplot corresponds to one cluster, with the x-axis indicating the API in the cluster and the y-axis showing the fraction of times that API was chosen for that specific API ordering method over 500 runs.

Table 4: Correlation between API-level features and model selection rates. Each entry shows Pearson r with its p-value.

Feature	ChatGPT 4.1	Gemini	Qwen
avg_similarity_tool_desc	$+0.227$(p=0.113)	-0.092(p=0.526)	$+0.234$(p=0.101)
avg_similarity_api_desc	$+0.111$(p=0.442)	$+0.330$(p=0.019)	$+0.411$(p=0.003)
age_days	-0.199(p=0.201)	-0.144(p=0.356)	-0.163(p=0.296)
desc_name_length_sum	$+0.044$(p=0.760)	$+0.103$(p=0.477)	$+0.038$(p=0.795)
num_params	-0.065(p=0.653)	$+0.038$(p=0.793)	-0.185(p=0.198)
flesch_reading_ease	$+0.160$(p=0.267)	$+0.176$(p=0.222)	$+0.098$(p=0.496)
positive_word_count	$+0.126$(p=0.384)	$+0.087$(p=0.547)	$+0.093$(p=0.521)

weaker with higher p-values. By contrast, structural or stylistic attributes (e.g., parameter count, promotional wording) exhibit little consistent influence. Tool age (`age_days`) shows a modest, broadly consistent negative correlation across models.

Linear regression reveals that surface-level semantic alignment is the primary signal but leaves a lot unexplained. Figure 13 tells us that linear models explain only part of the variance: R^2 is modest—0.143 for ChatGPT-4.1 and 0.387 for ToolLLaMA—leaving substantial error. Coefficients show surface-level semantic alignment dominates: similarity between the query and tool/API descriptions has the largest positive weights for most models, with Qwen weighting both most strongly and Gemini emphasizing API-level descriptions. Unexpectedly, ToolLLaMA gives a negative weight to tool-description similarity. Other features contribute little. Hence, semantic alignment is important in driving selection but still gives an incomplete explanation, implying nonlinear or omitted factors and motivating more flexible models (e.g., random forests).

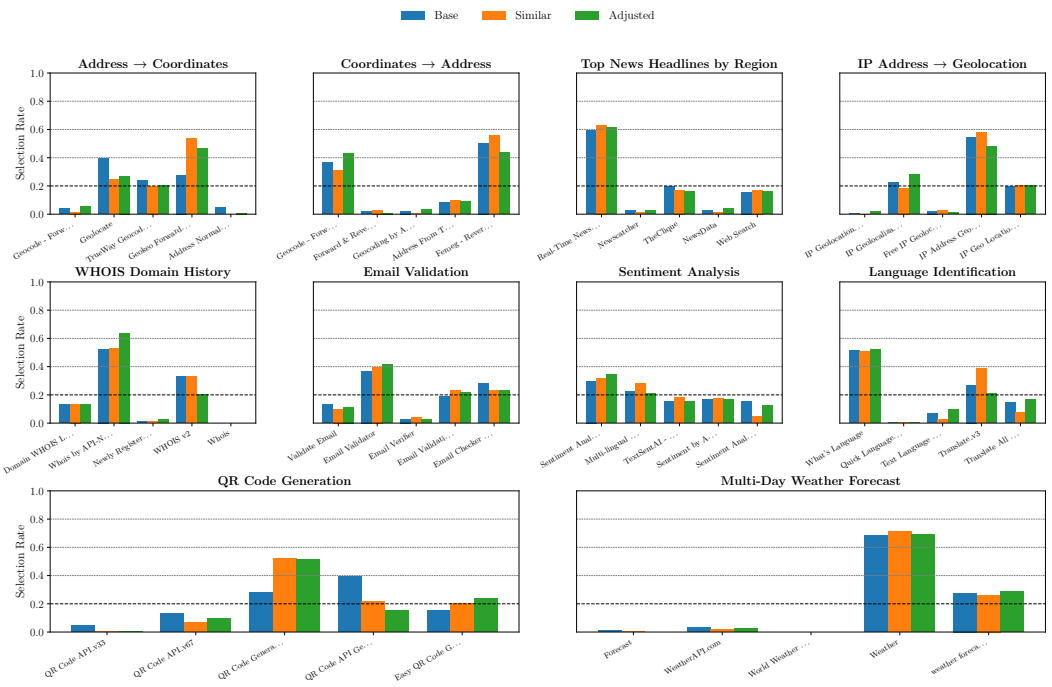


Figure 12: Selection distributions for ChatGPT 3.5 using different system prompts across ten clusters. Each subplot corresponds to one cluster, with the x-axis indicating the API in the cluster and the y-axis showing the fraction of times that API was chosen using the corresponding system prompt over 500 runs.

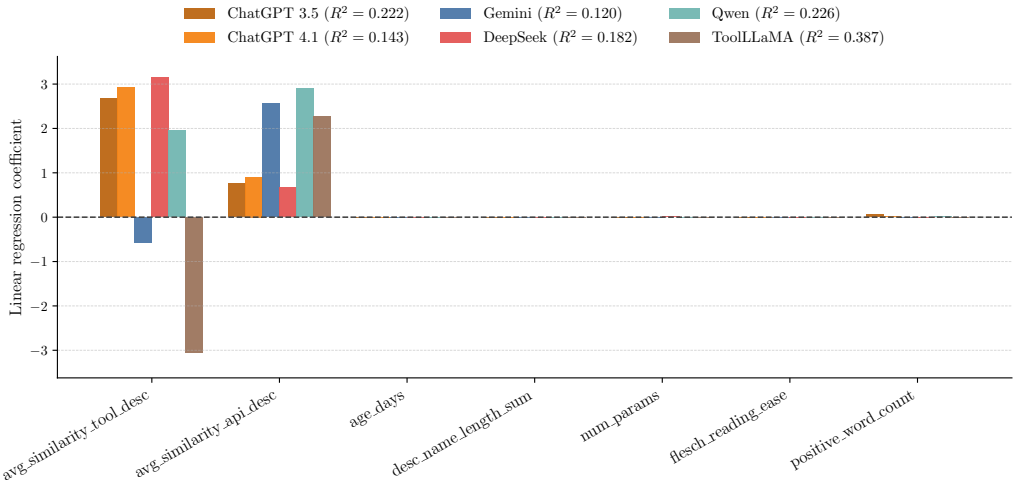


Figure 13: Linear regression feature weights used to predict API selection rates for six LLMs. Each group of bars corresponds to one API-level feature; different colors denote models, with their R^2 shown in the legend. Larger positive weights indicate features that increase the predicted selection rate.

Random forests fail to offer meaningful improvement. We fitted random-forest regressors with the same mean-centered features using both cross-validation and a held-out split, but predictive performance was poor, meaning the forests often did worse than a trivial constant baseline. This suggests the available features, at least in their current form and scale, don't contain enough signal or that some artifacts are overwhelming the gains from nonlinearity. Therefore, any feature-importance estimates from these trees would be unreliable and we do not lean on them for explanation. Future work could involve revisiting this with a richer feature set, more data, or alternative nonlinear modeling.

F.2 How do metadata interventions affect API selection?

We saw that corrupting descriptions produces much larger and more stable effects on selection behavior than name-level perturbations, which are sometimes noisy and unpredictable, in Section 4.3. Figure 14 corroborates this across clusters. Name perturbations often leave distributions near-unchanged or can make them drift unpredictably (e.g., Cluster 1, where Cluster 1 is positioned at the top-left and Cluster 6 at the bottom-right), whereas description/parameter scrambles frequently overhaul rankings: sometimes amplifying the dominant API (Cluster 2), other times causing dramatic re-ordering (Clusters 3). Name edits rarely produce comparably stable re-ranking.

Together, these patterns indicate that description-level semantics (and, to a lesser extent, parameter semantics) are the primary cues models use to discriminate among functionally similar APIs. Name perturbation alone tend to inject noise without consistent effects. Finally, bias persists under minimal semantic signal (only names and schema fields), implying selection behavior sometimes relies on residual, non-obvious priors rather than solely on coherent, human-interpretable heuristics.

Manipulating the description of certain tools has mixed effects across clusters. Figure 14 (lower row) shows three behaviors when we manipulate descriptions. First, swapping the most- and least-popular tools' descriptions can invert their selection rates (Cluster 4), indicating description text alone can dominate choice. Second, the same swap sometimes yields only a modest lift for the least-popular tool while unexpectedly altering the selection shares of unaffected tools (Cluster 5), suggesting the landscape is reconfigured rather than ranks simply exchanged. Third, in some clusters the swap has minimal effect (Cluster 6), implying other cues—e.g., name priors or parameter schemas—anchor preferences.

Targeted corruption of the most-selected tool's description has similarly inconsistent effects. In Cluster 4, scrambling collapses its share to near zero as another tool absorbs the mass; in Clusters 5–6, corruption diffuses probability across competitors, producing a more even allocation. Overall, description tampering often wields substantial influence, but the impact is context-dependent: the same intervention can invert, redistribute, or barely change preferences, underscoring that tool choice emerges from multiple interacting cues.

G Additional Figures

G.1 Correlation in Selection between Models

Figure 15 shows that models exhibit varying degrees of correlation in their tool-selection patterns, suggesting shared but non-identical biases across families.

G.2 Selection Distributions for all Clusters

This subsection provides a full version of the figure referenced in the main text (Figure 3). It expands the subset plot to all ten clusters and keeps axes, run counts, and error-bar conventions identical to the summary in Section 4.2. Use Figure 16 for detailed inspection of per-cluster behavior.

G.3 Full Figure related to the CPT Experiment

Figure 17 shows how biased continued pre-training gradually increases preference for the exposed endpoint.

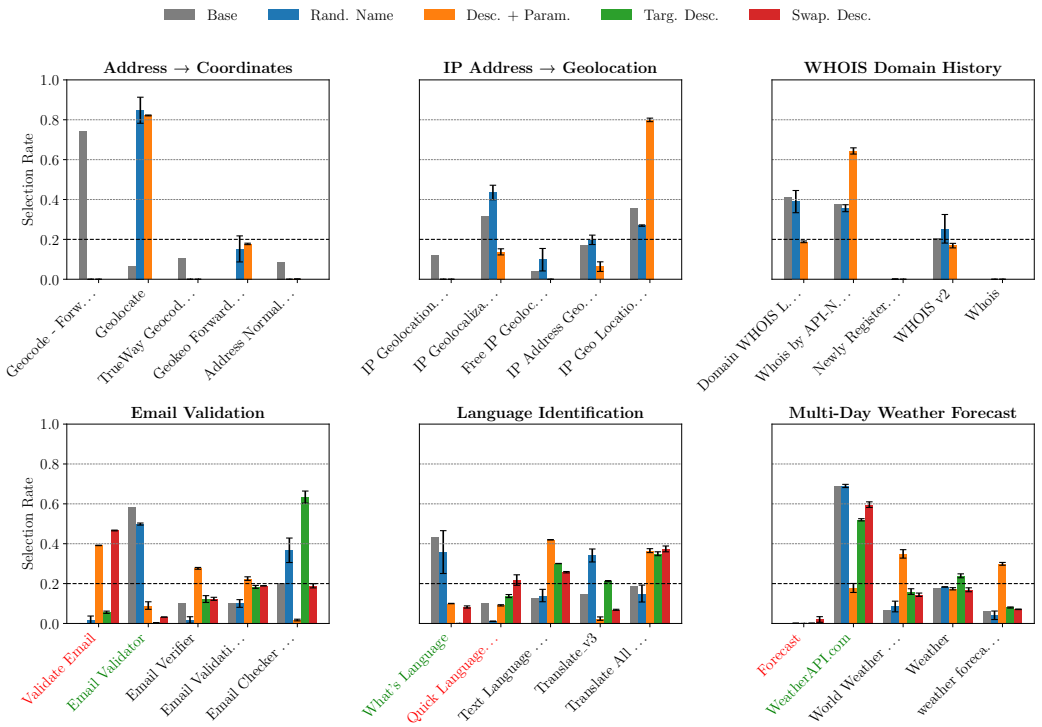


Figure 14: Selection distributions for Gemini under different name/ordering perturbations across six clusters of functionally equivalent APIs. Each subplot corresponds to one cluster; the x-axis lists the APIs and the y-axis shows the fraction of times the model under each condition selected that API, averaged over repeated runs. Error bars (when present) indicate the standard deviation across those repeats, making visible how robust or variable the preferences are under the different perturbations. Tools whose names are in green are the most selected by the baseline, and those in red are the least selected. These are the tools that are targeted for the swapping and selected scramble experiments.

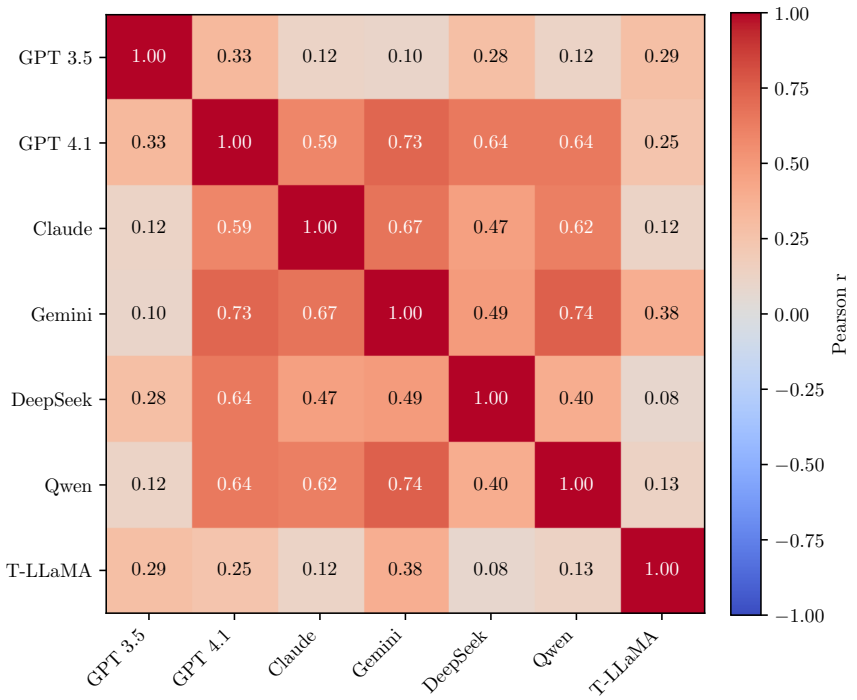


Figure 15: Pearson correlation matrix between models' tool-selection bias patterns.

G.4 Table related to the Reduction of Bias due to Mitigation

Table 5 demonstrates that our mitigation substantially flattens selection distributions and reduces both API- and position-level bias.

Table 5: Average cluster-level API bias δ_{API}, positional bias δ_{pos}, and combined bias δ_{model} before and after mitigation.

Setup	δ_{API}	δ_{pos}	δ_{model}
Before	0.338	0.422	0.380
After	0.108	0.079	0.094

H Effect of Metadata Perturbation on Bias

Relative to the base distributions, both models move farther from uniform (get more biased) when we lower semantic signal (see bars corresponding to the description + parameter and full perturbations in Figure 18). For Gemini, these manipulations yield the largest TV distances to uniform (≈0.42–0.43); ChatGPT shows similar results. This suggests that when descriptions/parameters are corrupted, models amplify bias rather than flatten choices.

Conversely, targeted edits to the most popular API tends to decrease bias. For Gemini, targeting the description of the most popular API leads to an average TVD slightly below the baseline and swapping the description between most- and least popular API leads to one that is substantially lower, indicating that weakening or transferring the strongest semantic cue moves the selection

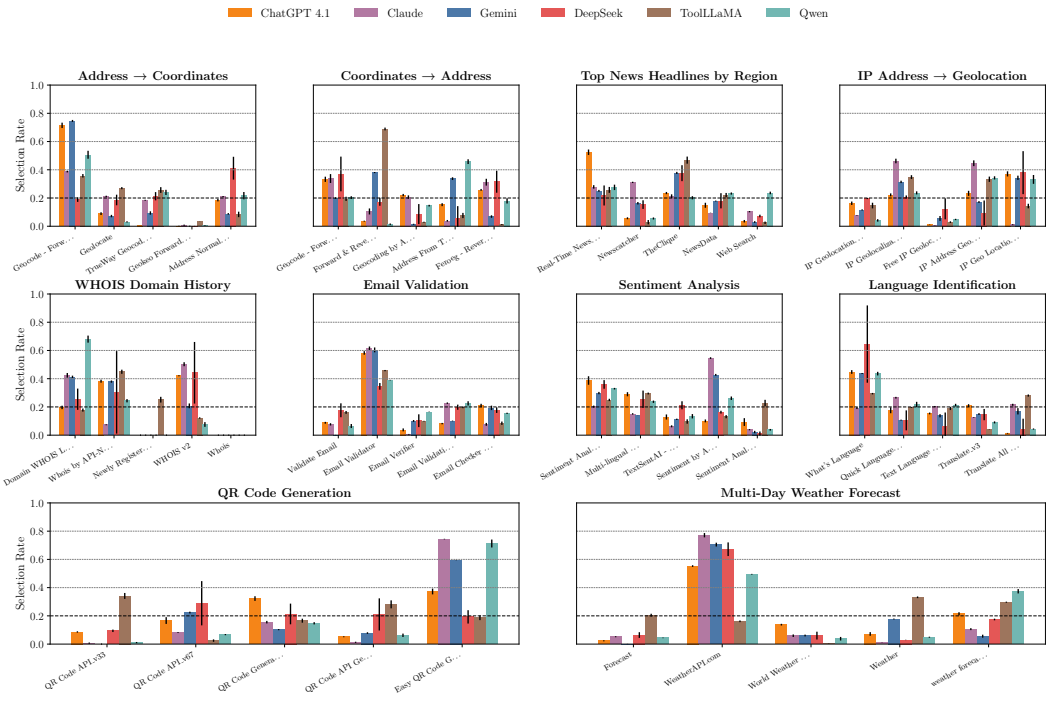


Figure 16: Selection distributions for six LLMs across ten clusters of functionally equivalent APIs. Each subplot corresponds to one cluster, with the x-axis indicating the API in the cluster and the y-axis showing the (mean) fraction of times each model chose that API over 500 inference runs; error bars indicate the standard deviation across three independent experimental runs. This visualization highlights how different models exhibit systematic preferences for some APIs.

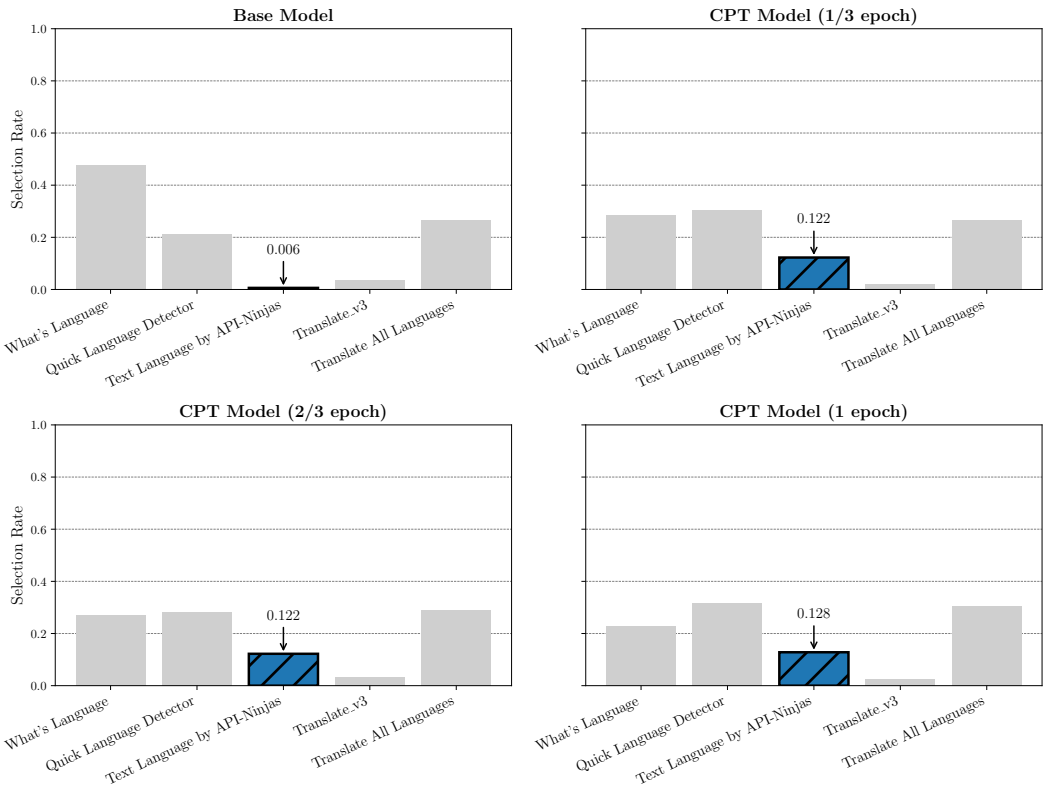


Figure 17: Selection rates for the Language Identification cluster across biased continued pretraining (CPT) checkpoints. Bars give the fraction that each endpoint was chosen across 500 inference runs. Panels show (top-left) base model, (top-right) CPT after 1/3 epoch, (bottom-left) 2/3 epoch, and (bottom-right) 1 full epoch. The Text Language by API-Ninjas endpoint is highlighted, with its exact selection rate printed above its bar. Differences across panels visualize how biased CPT shifts tool choice over training.

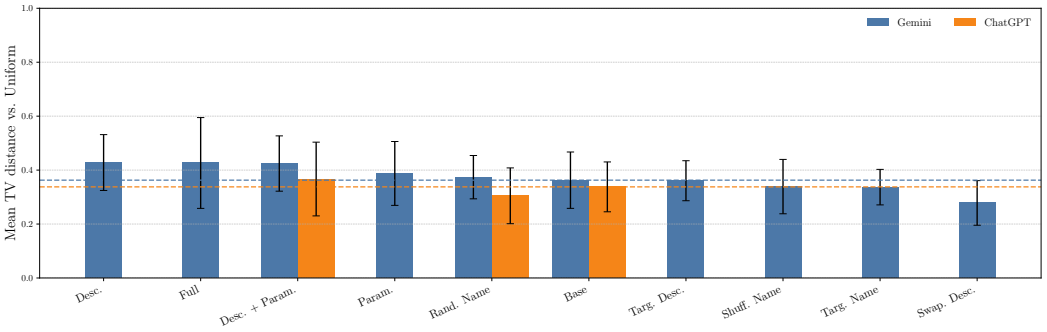


Figure 18: Mean total-variation (TV) distance from the uniform selection distribution to the distribution pertaining to each metadata perturbation (higher = more deviation from uniform). Blue bars show Gemini and orange bars show ChatGPT; error bars denote standard deviation across clusters, and single bars indicate perturbations not run for ChatGPT. Dashed horizontal lines (in the corresponding model colors) mark each model's baseline TV-to-uniform without perturbations.

distribution toward uniform. Name-only manipulations have similar effects, but name scrambling does not increase bias as much.

I Details on CPT Setup

Model and adapters. We continue pre-training Qwen3–8B–Base using Unsloth with 4-bit loading. The maximum sequence length is 2048. We attach LoRA adapters with $\sim$16.29% trainable parameters. See Table 6 for more info.

Table 6: Model/adapter configuration.

Base model	`unsloth/Qwen3-8B-Base-unsloth-bnb-4bit`
Max seq. length	2048
Quantization	4-bit (bitsandbytes)
LoRA hyperparameters	$r = 128,\ \alpha = 32$, dropout$= 0$

Data. We use our corpus saturated with metadata of a single target endpoint (Text Language by API-Ninjas). The corpus contains $\sim$3.5M tokens.

Training. Training uses Unsloth's trainer for one epoch with cosine LR scheduler and warmup. Optimizer is 8-bit AdamW. We also set a smaller LR for the embedding modules. See Table 7 for more info.

Table 7: CPT training hyperparameters.

Epochs	1
Total steps (epoch)	153
Per-device batch size	2
Grad. accumulation	8
Effective batch size	16
Learning rate	5×10^{-5} (embeddings 5×10^{-6})
Scheduler / Warmup	cosine / warmup ratio 0.1
Optimizer	`adamw_8bit`
Weight decay	0.0
Checkpoints used	step 0 (base), 52 ($\approx$1/3), 104 ($\approx$2/3), 153 (1 epoch)

Evaluation (inference). For all checkpoints, we keep prompts and decoding fixed: temperature $= 0.5$, top-p $= 1.0$, and `max_new_tokens`=512. We evaluate with the Language Identification cluster under circular shifts, aggregating the selection rates over 500 inference runs per checkpoint.