One Instruction Is a Benchmark: End-to-End Instruction-Following Evaluation with FlexBench

Anonymous ACL submission

Abstract

Real-world deployments of large language models(LLMs) increasingly depend on long, precise, and carefully constructed user instructions, such as standard operating procedures(SOPs), state machines, and multi-step workflows. In this setting, aggregate leaderboard performance is often a weak proxy for a models ability to faithfully execute any particular instruction that a practitioner cares about. To address this gap, we present **FlexBench**, an on-demand framework that automatically transforms a single seed instruction into an instruction-specialized benchmark. FlexBench (i) decomposes the instruction into a set of verifiable evaluation dimensions by treating it as a collection of checkable clauses, and (ii) generates a conversation corpus using a leakage-resistant user simulator. We further introduce **FlexEval**, which maps per-dimension, tri-valued judgments (`yes/no/unknown`) into instruction-level metrics that reflect workflow progress (Coverage) and conditional correctness (Achievement). Together, FlexBench and FlexEval provide a fully automated pipeline for *instruction-specialized* evaluation, enabling practitioners to build a benchmark tailored to their own instruction and to obtain reproducible, fine-grained diagnostics of instruction following. We validate the framework on 248 complex single-turn instructions and conduct extensive experiments across 10 leading LLMs in three multi-turn conversational scenarios with long, branching instructions. Our source code is publicly available at `https://anonymous.4open.science/r/FlexBench-E0D7`.

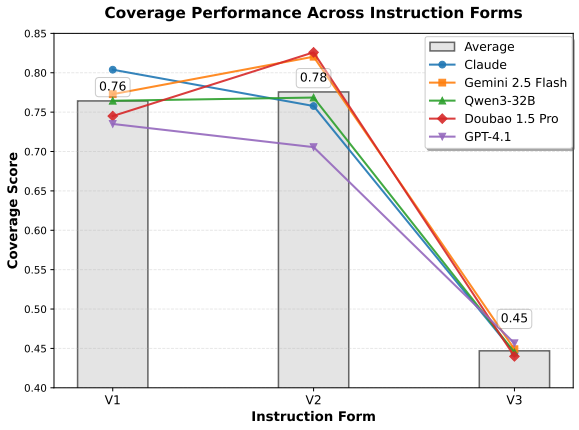
1 Introduction

Large Language Models (LLMs) have made rapid progress across generation, reasoning, and problem solving (Achiam et al., 2023; Chowdhery et al., 2023; Brown et al., 2020). In real deployments, users increasingly provide long, precise, and sometimes idiosyncratic instructions (e.g., strict formats, conditional branches, multi-step workflows). Yet instruction following remains fragile: a model that performs well on general-purpose leaderboards may still fail to satisfy a specific instructions concrete requirements. In practice, the question is often not "which model is best overall?", but "*which model best follows this particular instruction*?"

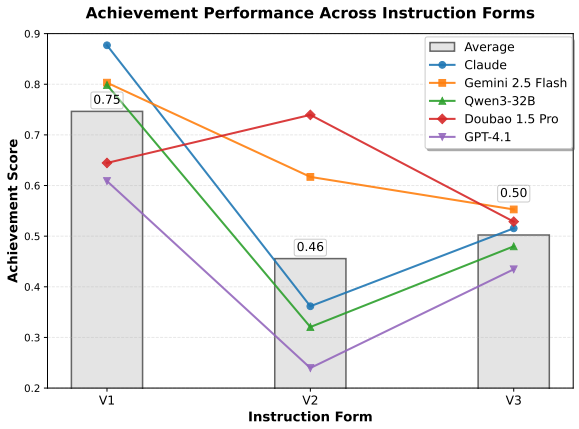
Figure 1 illustrates this issue. We evaluate mainstream LLMs on three representative instruction settings: **V1** (short single-turn), **V2** (long single-turn with multiple explicit constraints), and **V3** (multi-turn procedural/SOP-style instruction with branching conditions). Models show non-uniform patterns on both *Coverage* (whether required steps are attempted) and *Achievement* (whether conditionally applicable requirements are satisfied), indicating that instruction form and structure can strongly affect observed performance. This motivates instruction-specialized evaluation: building a benchmark tailored to the instruction at hand, rather than relying on a predefined fixed dataset.

Existing benchmarks provide useful signals but are typically static and not constructed for a particular user instruction. Qin et al. (2024) decomposes instructions into constraints, but relies on a carefully tailored while fixed dataset and uses flat aggregation without logic-aware gating. Wen et al. (2024) models compositional constraints, but logical structures are manually extracted. Zeng et al. (2023) simulates dialogues, but focuses on single-turn settings. In contrast, we aim to compile one user-provided instruction into an end-to-end, automatically generated test suite that supports multi-turn interaction and conditional structure.

We introduce **FlexBench**, which constructs an instruction-specialized benchmark $\mathcal{B}(I) = (\mathcal{D}, \mathcal{C})$ from a single seed instruction I: a set of verifiable evaluation dimensions $\mathcal{D}$ (instruction clauses) and a synthesized conversation corpus $\mathcal{C}$ designed to elicit the model behaviors expected under its intended deployment context. We also intro-



(a) Coverage (V1–V3).



(b) Achievement (V1–V3).

Figure 1: Instruction settings: V1 (short single-turn), V2 (long single-turn with multiple constraints), V3 (multi-turn procedural/SOP with branching conditions). Models exhibit non-uniform performance across settings, motivating instruction-specialized benchmark construction.

duce **FlexEval**, which aggregates per-dimension, tri-valued outcomes (yes/no/unknown) into two instruction-level metrics: Coverage (workflow progress) and Achievement (conditional correctness), using logic-aware gating to respect preconditions and mutually exclusive branches.

In summary, we propose an instruction-specialized benchmarking pipeline that automatically derives verifiable dimensions and a conversation corpus from a single instruction, and a logic-aware aggregator that reports Coverage and Achievement from tri-valued outcomes for reproducible, instruction-conditioned evaluation.

2 Related Work

Instruction-following evaluation has been studied by many researchers. IFEval (Zhou et al., 2023) introduced verifiable instructions, and InfoBench (Qin et al., 2024) decomposed complex requirements into atomic constraints. ComplexBench (Wen et al., 2024) explored compositional constraints, QA-IF (Adlakha et al., 2024) studied correctness vs. faithfulness in factual QA, LIFBench (Wu et al., 2024) tested stability under long contexts, and StrucText-Eval (Gu et al., 2024) examined reasoning on structured text. Reliability and robustness have been addressed by LLM-BAR (Zeng et al., 2023) and Automatic Dialogue Evaluators (Zhang et al., 2024), while self-evolving benchmarks (Wang et al., 2024) aim to mitigate staleness. Broader benchmarks such as BIG-bench (Srivastava et al., 2023), HELM (Liang et al., 2022), MT-Bench (Bai et al., 2024), and AlpacaEval (Li et al., 2023), along with resources like Super-NaturalInstructions (Wang et al., 2022), Tülu (Wang et al., 2023), Arena-Hard (Li et al., 2024), Chatbot Arena (Chiang et al., 2024), and AlpacaFarm (Dubois et al., 2023), further contribute large-scale or community-driven evaluations.

Domain-specific work extends these directions. CodeIF-Bench (Wang et al., 2025) targets interactive code generation, MathIF (Fu et al., 2025) evaluates reasoning models under compositional mathematical constraints, and InfoSearch (Zhou et al., 2024) and IFIR (Song et al., 2025) benchmark retrieval with nuanced or expert-domain instructions. While these highlight important facets of instruction following, limitations persist: handcrafted taxonomies, dataset dependence, evaluator bias, and narrow domains. FlexBench moves beyond these by inducing benchmarks directly from input instructions, ensuring coverage, long-range dependency, evaluability, and faithful maximality in a fully automated and scalable framework.

3 FlexBench: Instruction-Specialized Benchmark Construction

As shown in Figure 2, FlexBench constructs an instruction-specialized instruction-following benchmark from a single user-provided instruction. Given a seed instruction I, FlexBench outputs a benchmark $\mathcal{B}(I)$ consisting of (i) a set of verifiable evaluation dimensions and (ii) a conversation corpus designed to elicit the behaviors required by I:

$$I \xrightarrow{\text{FlexBench}} \mathcal{B}(I) = (\mathcal{D}, \mathcal{C}),$$
$$\mathcal{D} = \{d_i\}_{i=1}^{D}, \qquad \mathcal{C} = \{c_j\}_{j=1}^{M}. \tag{1}$$

What is a dimension? We treat the instruction I as an executable specification and compile it into

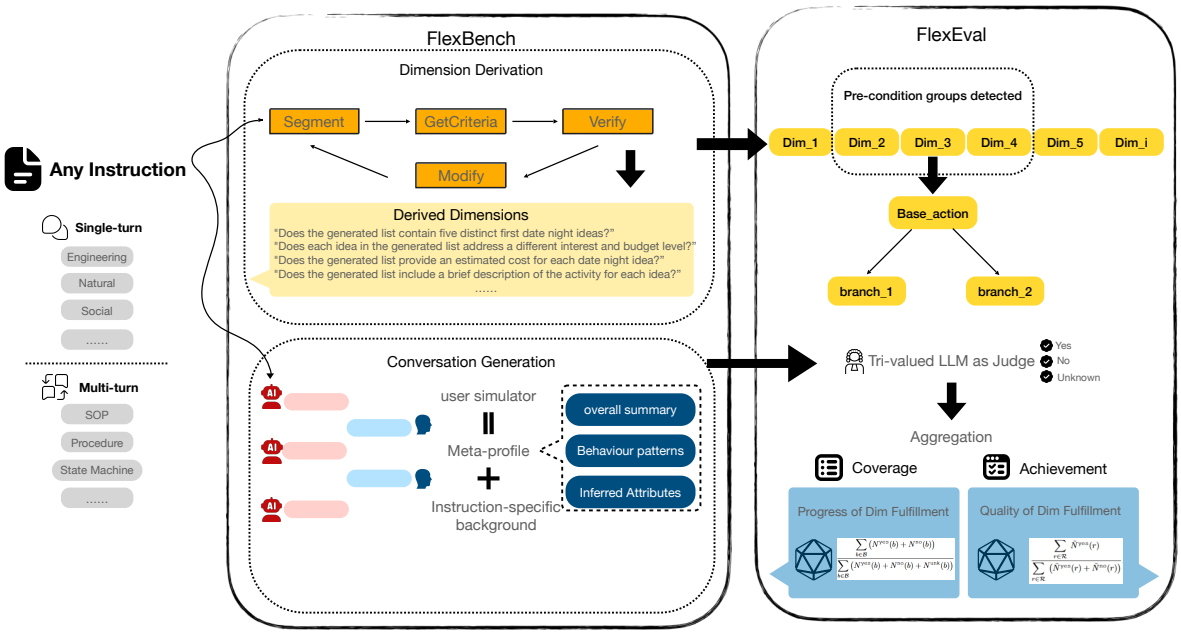


Figure 2: Workflow of FlexBench and FlexEval.

a set of checkable clauses. Each dimension d_i corresponds to one clause (or a small set of tightly related clauses) expressed in an entity-explicit, verifiable form (e.g., "Does the assistant include *X* in the output?"). Importantly, a dimension-level score is not intended as a standalone measure of a general capability; rather, it is a compliance signal for the given instruction.

Two-stage pipeline. FlexBench is fully automated: (1) **Dimension derivation** compiles I into $\mathcal{D}$; (2) **Conversation generation** synthesizes $\mathcal{C}$ to exercise those dimensions in both single-turn and multi-turn interactions. The resulting $\mathcal{B}(I) = (\mathcal{D}, \mathcal{C})$ serves as an on-demand test suite for the specific instruction I.

3.1 Dimension Derivation

Dimension interface: tri-valued, with optional preconditions. FlexBench derives dimensions that can be judged from a conversation trace τ (a multi-turn interaction between a user and an assistant). For each dimension d_i, we produce (i) a verifiable criterion (the "question" to be checked) and (ii) an optional precondition indicating when the criterion is applicable. Judgment is tri-valued: $\mathcal{V} = \{\mathsf{yes}, \mathsf{no}, \mathsf{unknown}\}$, where unknown means that the trace provides insufficient evidence or the dimension is not applicable.

Let $R_i(\tau) \in \mathcal{V}$ denote the per-dimension checker output and $A_i(\tau) \in \{0, 1\}$ an optional applicability predicate. Then the verdict for dimension d_i on trace τ is:

$$s_{d_i}(\tau) := \begin{cases} \mathsf{unknown}, & \text{if } A_i(\tau) = 0 \\ R_i(\tau), & \text{otherwise.} \end{cases} \quad (2)$$

This ensures we do not penalize a model for failing to satisfy a branch that was never triggered (e.g., a conditional requirement whose premise does not hold in τ).

Operational pipeline. We implement dimension derivation with a structured, LLM-assisted pipeline: (1) **Instruction segmentation** (split I into atomic requirement units); (2) **Decontextualization & coreference canonicalization** (make each unit entity-explicit and self-contained); (3) **Verification** (remove redundancy, ensure faithfulness to I, and check verifiability); (4) **Modification** (split overly broad units, merge trivial fragments, and normalize wording). Prompt templates and examples are provided in Appendix B. Implementation details (including the generator model and decoding settings) are reported in the experimental setup.

Design goals. A derived dimension set $\mathcal{D}$ aims to satisfy:

- **Completeness:** dimensions collectively cover the requirements in I so instruction-level eval-

uation can be determined from dimension-level decisions.

- **Non-redundancy & faithfulness:** dimensions do not duplicate each other and introduce no new requirements beyond I.
- **Evaluability:** each dimension is verifiable from traces, and may return unknown when evidence is insufficient.

3.2 Conversation Generation

Given I and derived dimensions $\mathcal{D}$, FlexBench synthesizes a conversation corpus $\mathcal{C}(I) = \{c_j\}_{j=1}^{M}$ using a user simulator. The goal is to elicit instruction-relevant behaviors under diverse interaction styles *without leaking evaluation dimensions* into the simulator. Persona acquisition details and JSON examples are deferred to Appendix D.

User simulator: backbone + constrained profile delta. We define a small taxonomy of user personas (e.g., varying cooperativeness) that share an instruction-invariant base prompt P_0 encoding stable turn-taking and communication conventions. For each instruction, we infer a constrained, instruction-specific profile delta $\Delta(I)$ containing only lightweight stylistic fields (e.g., catchphrases and behavioral patterns). Crucially, $\Delta(I)$ explicitly excludes evaluation dimensions and their templates.

Rollout. For each conversation c_j, we sample a rollout by interacting an evaluated model f with the simulator $U_{P_0 \oplus \Delta(I)}$ under a random seed z_j:

$$c_j \sim \mathrm{Rollout}\big(f,\, U_{P_0 \oplus \Delta(I)};\, z_j\big)\,, \qquad \mathcal{C}(I) = \{c_j\}_{j=1}^{M}. \tag{3}$$

Rollouts terminate upon task completion, simulator-initiated termination (e.g., uncooperative persona), or a maximum turn limit $T_{\max}$. We use seed variation and temperature sampling to encourage diversity while keeping persona constraints fixed.

Design goals. Conversation generation aims to provide:

- **Leakage resistance:** the simulator never observes $\mathcal{D}$ (nor few-shot traces that encode it), reducing the risk of gaming the evaluation.
- **Controlled diversity:** varied but realistic interaction trajectories that surface different branches and edge cases implied by I.
- **Reproducibility:** the corpus is regenerated from I via fixed prompts and decoding settings (reported in experiments).

4 FlexEval: Coverage and Achievement

FlexEval aggregates tri-valued, per-dimension verdicts into instruction-level scores. Our goal is to separate two common failure modes in instruction following: **(i) skipping required steps** vs. **(ii) making mistakes when a step is exercised**. Accordingly, we report two complementary metrics: COVERAGE (workflow progress) and ACHIEVEMENT (conditional correctness).

4.1 Setup

Let $\mathcal{D}$ be the derived dimensions (Section 3.1) and $\mathcal{C}(I)$ the conversation corpus for instruction I (Section 3.2). For each trace $\tau \in \mathcal{C}(I)$ and dimension $d \in \mathcal{D}$, we obtain a tri-valued verdict $s_d(\tau) \in \{\mathsf{yes}, \mathsf{no}, \mathsf{unknown}\}$, where unknown means insufficient evidence or inapplicability (e.g., a conditional branch that is not triggered).

To respect conditional structure, we organize dimensions into *precondition groups* $\mathcal{G} = \{g_k\}_{k=1}^{K}$, each containing one base action b_g and (optionally) a set of dependent branches $\mathcal{R}_g$. Let $\mathcal{B} = \{b_g \mid g \in \mathcal{G}\}$ and $\mathcal{R} = \bigcup_{g \in \mathcal{G}} \mathcal{R}_g$, and denote by $b(r)$ the base action associated with branch $r \in \mathcal{R}$. (How $\mathcal{G}$ is extracted is described in the appendix.)

4.2 Counts

For each dimension d, we summarize outcomes over the corpus:

$$N^{(v)}(d) = \sum_{\tau \in \mathcal{C}(I)} \mathbb{I}[s_d(\tau) = v]\,, \qquad v \in \{\mathsf{yes}, \mathsf{no}, \mathsf{unknown}\}. \tag{4}$$

We also define the attempt count $N^{\mathsf{att}}(d) = N^{\mathsf{yes}}(d) + N^{\mathsf{no}}(d)$, i.e., the dimension is actually exercised rather than unknown.

For a branch $r \in \mathcal{R}$, we use gated counts that only include traces where the associated base action is attempted:

$$\tilde{N}^{(v)}(r) = \sum_{\tau \in \mathcal{C}(I)} \mathbb{I}\Big[s_{b(r)}(\tau) \in \{\mathsf{yes}, \mathsf{no}\} \wedge\ s_r(\tau) = v\Big], \qquad v \in \{\mathsf{yes}, \mathsf{no}\}. \tag{5}$$

(Equivalently: branches contribute to correctness only when their base action is exercised; oth-

erwise they are treated as not applicable. Details are in the appendix.)

4.3 Metrics

Coverage (workflow progress). Coverage measures whether the model *attempts required base actions* (progress), regardless of correctness:

$$\text{Coverage}(I) := \frac{\sum_{b\in\mathcal{B}} N^{\texttt{att}}(b)}{\sum_{b\in\mathcal{B}} \left(N^{\texttt{att}}(b) + N^{\texttt{unknown}}(b)\right)}. \tag{6}$$

Low Coverage indicates omissions: required steps are skipped or never evidenced in the interaction.

Achievement (conditional correctness). Achievement measures *correctness conditioned on acting*: when the workflow is exercised, how often are the conditionally applicable branch requirements satisfied?

$$\text{Achievement}(I) := \frac{\sum_{r\in\mathcal{R}} \tilde{N}^{\texttt{yes}}(r)}{\sum_{r\in\mathcal{R}} \left(\tilde{N}^{\texttt{yes}}(r) + \tilde{N}^{\texttt{no}}(r)\right)}. \tag{7}$$

If the denominator is zero (no branch is ever exercised), we report *n/a*.

Why two metrics? Coverage captures "*did the model do the required steps?*" Achievement captures "*when it did them (and branches became applicable), did it do them correctly?*". Reporting both yields actionable, instruction-specific diagnostics for model selection and debugging. This decoupling offers precise diagnostic signals for model developers. specifically, low Coverage usually implies the model is too passive or fails to recall long-context instructions (requires stronger SFT/instruction following), whereas high Coverage with low Achievement suggests the model attempts the workflow but fails at reasoning or hallucinating details (requires better reasoning capabilities or RLHF). This granularity enables more targeted post-training interventions than a single aggregate score.

Mutual exclusion. For instructions with explicit alternatives (e.g., `if/else`), non-applicable branches naturally yield `unknown` and are excluded by gating; additional mutual-exclusion handling is described in the appendix.

5 Experiments

5.1 Experimental Setup

Models under evaluation. We evaluate 10 LLMs: Claude (Claude Sonnet 4) (Anthropic, 2025); Gemini 2.5 Flash and Gemini 2.5 Flash Lite (Comanici et al., 2025); Qwen3-32B (Yang et al., 2025); Doubao 1.5 Pro (Doubao 1.5 Pro 32k), Doubao 1.5 Role (Doubao 1.5 Pro 32k character 0715), and Doubao 1.5 Lite (Doubao 1.5 Lite 32k) (ByteDance, 2025); and GPT-4.1, GPT-4.1-mini, and GPT-4.1-nano (OpenAI, 2025).

Benchmark construction and evaluation backend (GPT-4.1). Unless otherwise specified, **all LLM-driven components of FlexBench and FlexEval are instantiated with GPT-4.1**: (1) dimension derivation (Segmentation → Criteria → Verification/Modification), (2) logic extraction for precondition groups and mutual-exclusion patterns, (3) instruction-conditioned simulator profile adaptation and LLM-based termination detection, and (4) per-dimension tri-valued judging (`yes`/`no`/`unknown`) used by FlexEval. This keeps benchmark construction and evaluation fixed across all tested models.

Decoding and randomness control. We use a shared decoding temperature of 0.3 for all evaluated models to mitigate excessive randomness while avoiding degenerate overly-deterministic outputs in interactive settings. To preserve reproducibility, all prompts are fixed and we report aggregated results over multiple dialogues with controlled random seeds; we further quantify cross-test stability via two independently generated dialogue corpora (5.3).

5.2 Decomposition

Setup. We compare automatic dimension derivation (OURS) with human-authored decompositions (HUMAN) on **248** instructions using two protocols: (i) *LLM-as-judge* pairwise preferences over six criteria (Granularity, Evaluation Suitability, Completeness, Coverage, Task Alignment, Clarity); and (ii) *semantic alignment* on **1,560** aligned pairs via sentence-level BERT cosine, token-level BERTScore (P/R/F1), and ROUGE-L F1. For (i), we use GPT-4.1 as the judge model (Section 5.1).

Results. Semantic alignment indicates strong content overlap: BERT cosine 0.928, BERTScore F1 0.818, ROUGE-L F1 0.635.

Table 1: **Overall evaluation results.** Table 1a reports semantic alignment, and Table 1b shows LLM-as-judge comparisons.

(a) **Semantic alignment** between OURS and HUMAN across 1,560 aligned pairs.

Metric	Mean	Std
BERT cos. (sent.)	0.928	0.044
BERTScore–P	0.820	0.092
BERTScore–R	0.819	0.085
BERTScore–F1	0.818	0.082
ROUGE-L	0.635	0.171

(b) **LLM-as-judge comparison** between OURS (A) and HUMAN (B) on 248 instructions.

Criterion	A win (%,Δ)	B win (%)
Overall (N=248)	62.1 (+24.6)	37.5
Granularity	61.7 (+33.9)	27.8
Eval. Suitability	62.1 (+25.8)	36.3
Completeness	56.5 (+26.3)	30.2
Coverage	54.8 (+25.0)	29.8
Task Alignment	29.8 (+17.3)	12.5
Clarity	25.4 (−14.1)	39.5

LLM-as-judge prefers OURS in $154/248$ cases (62.1%), with the largest gains on *Granularity* (+33.9 pts) and *Evaluation Suitability* (+25.8 pts). Only *Clarity* favors HUMAN (-14.1 pts), suggesting that our finer-grained dimensions can be improved in wording. Overall, OURS maintains content fidelity while producing more evaluable decompositions. Additional analysis on dissimilar pairs is provided in Appendix C.

5.3 Self and Cross-model Consistency

To examine the stability of benchmark construction, we generate two independent dialogue corpora from the same instruction by varying rollout seeds, and evaluate seven models on both. We compare Coverage and Achievement across the two tests. The average absolute error, reported as $Mean \pm Std.$, is (0.0056 ± 0.0041) for Coverage and (0.0204 ± 0.0139) for Achievement. Figure 3 shows high agreement across tests and preserved cross-model ranking, indicating stable benchmark construction.

Figure 3 summarizes the results of two tests across seven models. Figures 3(a) and (d) show that coverage and achievement in Tests 1 and 2 are similar overall. A more detailed comparison is presented using two complementary methods. Figures 3(b) and (e) present Bland-Altman plots (BA plots), indicating that differences across tests remain small and stable for all models, which demonstrates high self-consistency. Figures 3(c) and (f) show pair line plots where results from each test are sorted by value and connected across models. No crossings are observed, meaning the relative performance ranking of models remains unchanged, which demonstrates high consistency and stability.

5.4 Main Results

Table 2 and Figure 4 summarize model performance. **Claude** achieves the best overall results (Coverage $= 0.804$, Achievement $= 0.877$), followed by **Gemini 2.5 Flash** and **Qwen3-32B**. **GPT-4.1** reaches competitive Coverage (0.735) but lower Achievement (0.609), suggesting that it attempts most required steps yet is less reliable on conditionally gated requirements. Across instruction types, $d4$ consistently depresses Achievement (e.g., Claude 0.467, GPT-4.1 0.150), highlighting the difficulty of logic-gated or formatting-sensitive branches.

Radar plots reveal consistent within-family trends: larger models exhibit fewer sharp drops across dimensions. Within the GPT-4.1 family, GPT-4.1 outperforms its mini and nano variants on both metrics, indicating a scaling-consistent improvement under our instruction-specialized evaluation. Overall, Coverage and Achievement behave differently across models: Coverage often saturates earlier, while Achievement provides finer discrimination on conditional correctness, supporting the need to report both metrics.

5.5 Pearson Correlation

Results and Analysis. Table 3 shows that LLM-as-judge achieves higher dimension-level accuracy than crowd-sourced annotations (micro average 0.852 vs. 0.750, macro average 0.855 vs. 0.755), with consistent improvements on most models. At the system level (Table 4), correlations with expert annotations are nearly perfect ($r = 0.997$ for *Coverage*, $r = 0.986$ for *Achievement*), whereas crowd-sourced labels correlate weakly or negatively. Overall, LLM-as-judge not only aligns more closely with expert judgments but also preserves relative model ranking, making it a more reliable evaluation signal for subsequent experiments.

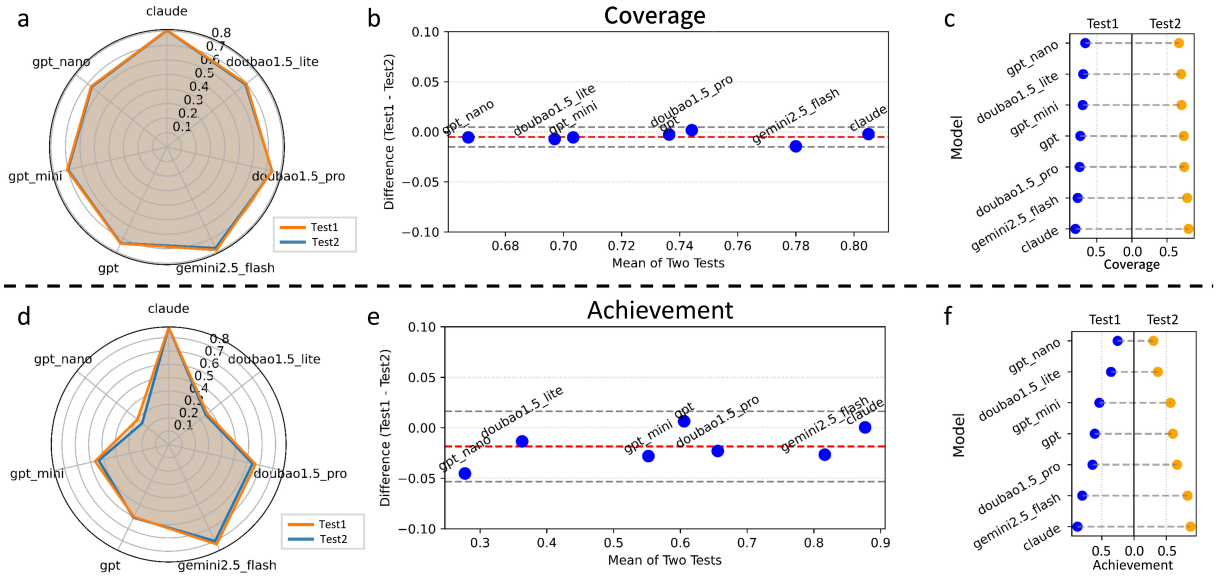


Figure 3: Consistency analysis for **coverage** (a–c) and achievement (d–f) across seven models in two independently generated test sets from the same instruction. (a, d) Radar charts comparing coverage and achievement across test sets. (b, e) Bland–Altman plots showing agreement between Test 1 and Test 2. (c, f) Pair line plots illustrating cross-model changes between tests; no crossovers indicate preserved relative ranking. In pair line plots, scores in each test are sorted by value and dashed lines connect results from the same model.

6 Conclusion and Future Work

We introduced **FlexBench**, a self-evolving benchmark constructed automatically from a single seed instruction. Unlike prior work based on fixed datasets or manual decomposition, FlexBench is fully automated end-to-end, requiring no human-written rubrics or annotations. Every benchmark instance is induced directly from the instruction itself, enabling task- and scenario-specific evaluation rather than relying on static task banks. Built on this foundation, **FlexEval** aggregates tri-valued, per-dimension outcomes into two complementary metrics: *Coverage*, capturing whether models attempt the required steps, and *Achievement*, capturing whether they act correctly when they do. This separation of workflow completeness from conditional correctness provides a robust and interpretable view of instruction-following performance.

FlexBench demonstrates that instruction-specialized, self-evolving evaluation is both feasible and necessary as LLMs are increasingly deployed on long, structured, and consequential tasks. While limitations remainsuch as handling extremely long or underspecified prompts, inducing rare logical dependencies, and extending beyond text-only tracesour framework establishes a foundation for instruction-targeted benchmarking that can scale with real-world demands. Future work will extend FlexBench to richer modalities and strengthen automatic reasoning over dependencies, further advancing reliable and adaptive evaluation for the next generation of LLMs.

Limitations

FlexBench/FlexEval are designed for *instruction-specialized* evaluation, so results should be interpreted as compliance with the given seed instruction rather than a universal measure of general capability. While we minimize randomness via low-temperature decoding, we avoid strictly deterministic settings (i.e., greedy decoding) to prevent degenerate outputs, which may introduce minor variances in the results. In addition, our current aggregation assumes that many complex instructions can be factorized into verifiable clauses with conditional structure; instructions with highly ambiguous, fuzzy, or strongly interdependent requirements may be harder to represent perfectly. Finally, conversations are generated by a simulator to provide controlled diversity and reproducibility, which may not fully match all real user behaviors. We view these as natural directions for future work (e.g., broader backend comparisons and real-user validation).

Table 2: **Main results (Top-10 models) across instruction types (3–5).** Using the *provided* overall *Coverage* and *Achievement* from the dataset (higher is better).

Model	C.(d1)	A.(d1)	C.(d2)	A.(d2)	C.(d3)	A.(d3)	C.(d4)	A.(d4)	C.(d5)	A.(d5)	C. ↑	A. ↑
Claude	1.000	1.000	0.900	1.000	1.000	0.972	1.000	0.467	0.920	1.000	**0.804**	**0.877**
Gemini 2.5 Flash	0.800	0.769	0.900	0.895	0.800	1.000	1.000	0.745	0.880	0.932	0.773	0.803
Qwen3-32B	0.760	0.857	0.960	1.000	0.760	0.968	0.880	0.583	0.880	0.955	0.764	0.799
Doubao 1.5 Pro	0.480	0.700	0.360	0.750	0.480	0.929	1.000	0.086	0.740	0.946	0.745	0.645
GPT-4.1	0.760	0.938	0.640	0.231	0.760	0.500	0.500	0.150	0.760	1.000	0.735	0.609
Gemini 2.5 Flash Lite	0.840	0.957	0.560	0.158	0.840	1.000	0.500	0.467	0.520	1.000	0.745	0.565
Doubao 1.5 Role	0.660	1.000	0.500	0.529	0.660	1.000	0.500	0.067	0.780	1.000	0.731	0.557
GPT-4.1-mini	0.320	0.778	0.340	0.067	0.320	0.429	0.740	0.050	0.280	1.000	0.701	0.538
Doubao 1.5 Lite	0.040	0.000	0.240	0.100	0.040	0.500	0.200	0.000	0.380	0.684	0.693	0.356
GPT-4.1-nano	0.040	0.000	0.100	0.000	0.040	0.000	0.000	0.000	0.000	0.000	0.664	0.255

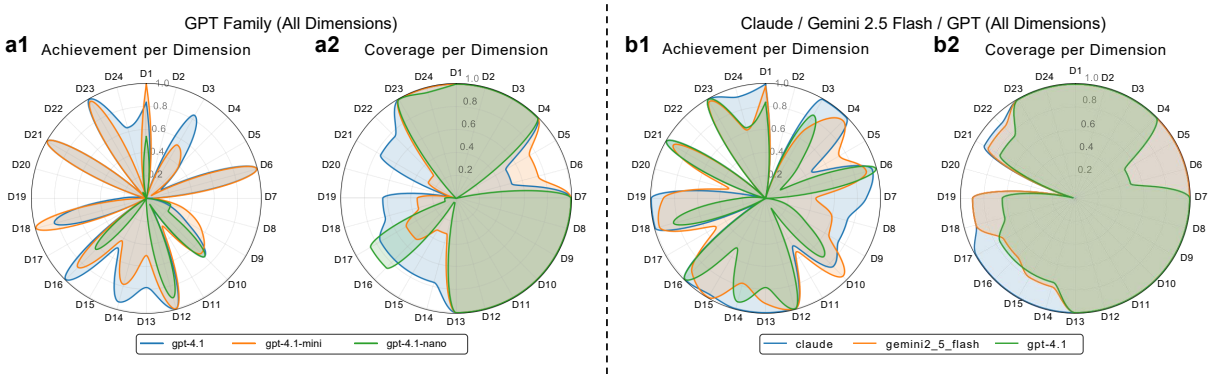


Figure 4: Comparison of different models across 24 dimensions. Five models are evaluated in total. (a) shows comparisons within the GPT family, with (a1) and (a2) presenting achievement and coverage, respectively. (b) compares Claude, Gemini 2.5 Flash, and GPT-4.1, with (b1) and (b2) presenting achievement and coverage, respectively.

Table 3: Dimension-level accuracy (Overall Accuracy; correct if $\arg\max$ prediction matches expert label per dialogue and dimension).

Model	**Crowd-sourced**	**LLM-as-Judge**	**Dialogues**	**Dims**
`claude`	0.611	0.883	21	504
`doubao1_5_pro`	0.852	0.798	20	480
`gemini2_5_flash`	0.772	0.888	13	312
`gpt`	0.782	0.852	18	432
Micro avg (dims)	0.750	0.852	72	1,728
Macro avg (models)	0.755	0.855	–	–

Table 4: Cross-model Pearson correlations r with expert annotations (system-level consistency).

Pair	**Coverage** r	**Achievement** r
Crowd-sourced vs Ground Truth	0.043	-0.227
LLM-as-Judge vs Ground Truth	0.997	0.986

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Vaibhav Adlakha, Parishad BehnamGhader, Xing Han Lu, Nicholas Meade, and Siva Reddy. 2024. Evaluating correctness and faithfulness of instruction-following models for question answering. *Transactions of the Association for Computational Linguistics*, 12:681–699.

Anthropic. 2025. Claude sonnet 4: 1m context window. https://www.anthropic.com/news/1m-context. Accessed: 2025-09-23.

Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and 1 others. 2024. Mt-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. *arXiv preprint arXiv:2402.14762*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

ByteDance. 2025. Doubao 1.5 pro technical blog. https://seed.bytedance.com/en/special/doubao_1_5_pro. Accessed: 2025-09-23.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, and 1 others. 2024. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto. 2023. Alpacafarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36:30039–30069.

Tingchen Fu, Jiawei Gu, Yafu Li, Xiaoye Qu, and Yu Cheng. 2025. Scaling reasoning, losing control: Evaluating instruction following in large reasoning models. *arXiv preprint arXiv:2505.14810*.

Zhouhong Gu, Haoning Ye, Xingzhou Chen, Zeyang Zhou, Hongwei Feng, and Yanghua Xiao. 2024. Structext-eval: Evaluating large language model's reasoning ability in structure-rich text. *arXiv preprint arXiv:2406.10621*.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. 2025. Llms get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. 2024. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *arXiv preprint arXiv:2406.11939*.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, CG Ishaan Gulrajani, P Liang, and TB Hashimoto. 2023. Alpacaeval: an automatic evaluator of instruction-following models (2023). *URL https://github. com/tatsu-lab/alpaca_eval*.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, and 1 others. 2022. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*.

OpenAI. 2025. Gpt-4.1 technical blog. https://openai.com/index/gpt-4-1/. Accessed: 2025-09-23.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. 2024. Infobench: Evaluating instruction following ability in large language models. *arXiv preprint arXiv:2401.03601*.

Tingyu Song, Guo Gan, Mingsheng Shang, and Yilun Zhao. 2025. Ifir: A comprehensive benchmark for evaluating instruction-following in expert-domain information retrieval. *arXiv preprint arXiv:2503.04644*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adri Garriga-Alonso, and 1 others. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on machine learning research*.

Peiding Wang, Li Zhang, Fang Liu, Lin Shi, Minxiao Li, Bo Shen, and An Fu. 2025. Codeif-bench: Evaluating instruction-following capabilities of large language models in interactive code generation. *arXiv preprint arXiv:2503.22688*.

Siyuan Wang, Zhuohan Long, Zhihao Fan, Zhongyu Wei, and Xuanjing Huang. 2024. Benchmark self-evolving: A multi-agent framework for dynamic llm evaluation. *arXiv preprint arXiv:2402.11443*.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, and 1 others. 2023. How far can camels go? exploring the state of instruction tuning on open resources. *Advances in Neural Information Processing Systems*, 36:74764–74786.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, and 1 others. 2022. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705*.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxing Xu, and 1 others. 2024. Benchmarking complex instruction-following with multiple constraints composition. *Advances in Neural Information Processing Systems*, 37:137610–137645.

Xiaodong Wu, Minhao Wang, Yichen Liu, Xiaoming Shi, He Yan, Xiangju Lu, Junmin Zhu, and Wei Zhang. 2024. Lifbench: Evaluating the instruction following performance and stability of large language models in long-context scenarios. *arXiv preprint arXiv:2411.07037*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. 2023. Evaluating large language models at evaluating instruction following. *arXiv preprint arXiv:2310.07641*.

Chen Zhang, Luis Fernando D'Haro, Yiming Chen, Malu Zhang, and Haizhou Li. 2024. A comprehensive analysis of the effectiveness of large language models as automatic dialogue evaluators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19515–19524.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

Jianqun Zhou, Yuanlei Zheng, Wei Chen, Qianqian Zheng, Hui Su, Wei Zhang, Rui Meng, and Xiaoyu Shen. 2024. Beyond content relevance: Evaluating instruction following in retrieval models. *arXiv preprint arXiv:2410.23841*.

A Details in Instruction Selection

A.1 Single-turn

For the single-turn setting, we adopt the `Hard Set` from InfoBench (Qin et al., 2024). The Hard Set is manually curated by experts and covers **72 domains** spanning Natural Sciences, Social Sciences, Engineering, Economics, Arts, and daily-life tasks (see Figure 1 in Qin et al., 2024).

Compared with the Easy Set, the Hard Set features **longer instructions** (average length ∼59 words), **more requirements per instruction** (average ∼6.3), and **richer constraint types** (Content, Linguistic, Style, Format, and Number). We use the Hard Set as a reference, comparing its human-provided decompositions with our automatically extracted dimensions.

Instruction	Words	Sentences
Customer Support Specialist	490	19
Cross-Border 3PL Assistant	140	6
Station Leader (FastRun)	145	11
Average	**258.3**	**12.0**

Table 5: Length and sentence statistics of multi-turn instructions.

A.2 Multi-turn

For the multi-turn setting, we construct three domain-specific long instructions: (i) *Customer Support Specialist for a Course Publishing Platform*, (ii) *Cross-Border 3PL Fulfillment Caller Assistant*, and (iii) *Station Leader for FastRun Riders*. Figure 5 shows the *Customer Support Specialist* instruction, which requires informing institutional clients about new streaming options ("Standard Live" vs. "Low-Latency Direct") while following strict procedures. These instructions embed state-machine style preconditions and branching flows, making multi-turn settings more realistic and challenging.

Table 5 summarizes statistics of our three multi-turn instructions. Compared to the single-turn Hard Set (average length ∼59 words), our multi-turn instructions are much longer on average (∼258 words, $4.4\times$), contain more sentences (∼12), and include branching flows. As a result, manual decomposition becomes substantially more time-consuming and error-prone.

B Dimension Extraction Prompts & Detailed End-to-End Pipeline

This appendix details our fully automated pipeline that converts a fully specified instruction into a structured set of conversation-observable dimen-

sions. The procedure follows a four-stage LLM-driven process: **Segmentation**, **Criteria Generation**, **Coverage Verification**, and **Targeted Modification**, augmented with two engineering enhancements for **Gradual** reduction and **Dependency** consolidation. All core prompts used in the LLM-driven stages are provided in the appendix (Prompts A–D in Figures 6–9).

B.1 Segmentation

We follow the segmentation prompt in Laban et al. (2025) to obtain robust segments. Given a fully specified instruction, we segment it into *non-overlapping*, *conversation-observable* units of information ("segments") that preserve core behavioral requirements while avoiding excessive fragmentation. The prompt enforces non-overlap, encourages minimal yet meaningful granularity, and prioritizes segments that can be verified through multi-turn dialogue.

B.2 Criteria Generation

Using the original instruction together with the segments, we convert each segment into a dimension stated as an *evaluation question* anchored to an observable output artifact. The prompt enforces *zero hallucination and strict grounding* to the instruction/segments, requires explicit *conditional dependencies* ("if.../when..."), and applies a *summary + specific script* dual structure whenever recommended dialogue scripts exist.

B.3 Verification

We assess whether the dimension set adequately covers the core information in the original instruction. The verifier returns `{"coverage":"complete"}` if coverage is sufficient; otherwise it returns `{"coverage":"incomplete","missing_segment":}` and pinpoints the missing unit. This diagnostic signal drives targeted revision.

B.4 Modification

When coverage is incomplete, we perform localized edits based on verification feedback while preserving the original JSON schema. The modification prompt adds missing dimensions, clarifies vague criteria, completes conditional logic, and improves measurability, while avoiding scope creep. The revised set is re-verified until coverage is complete.

B.5 Implementation Details

Our implementation executes the pipeline as: *Segmentation* $\rightarrow$ *Criteria* $\rightarrow$ *Verification* $\rightarrow$ *(Modification)** $\rightarrow$ *Gradual* $\rightarrow$ *Dependency**, where * indicates "applied as needed".

- **Fully automated without human intervention.** Each stage is driven by fixed prompts (Prompts A–D) and programmatic wrappers. We include robust JSON repair, bounded retries, and verification–modification loops to ensure stability and completeness.
- **Retry with temperature annealing.** On malformed outputs or verification failures, we trigger bounded retries with slight temperature increases to promote diversity while maintaining overall determinism.
- **Coverage-first control flow.** Only dimension sets verified as complete proceed to Gradual/Dependency refinements; otherwise we loop through Targeted Modification and re-verify.

Instruction Example: Customer Support Specialist for Course Publishing Platform

Role: Customer Support Specialist for Course Publishing Platform

Task: Inform institution clients that the course publishing page will now display "Standard Live" and "Low-Latency Direct" separately. Encourage selecting Low-Latency Direct when real-time interaction matters.

Opening Line: Hello, are you the person in charge for your training organization/campus?

Conversation Flow:

Step 1: Identity Confirmation

- If the person is in charge → proceed to Step 2

- If not → ask them to relay the message, then proceed to Step 2

Suggested Script: We've upgraded the live streaming product and added a separate "Low-Latency Direct" option. When you publish a class, just choose Low-Latency Direct; everything else remains unchanged.

Step 2: Check Prior Awareness

Ask: Previously you selected Standard Live, but we actually routed you via a low-latency line on the backend to protect quality. Were you aware?

- If not aware → explain that the front-end wasn't open yet and you temporarily enabled low latency to ensure audio/whiteboard sync

- If aware → proceed to Step 3

Step 3: Communicate the Upgrade

Suggested Script: Going forward, both options will be listed separately on the publishing page. Choose according to the course type.

3.1 Differences

- **Standard Live:** lower cost; latency around 5–10 seconds; good for large lectures

- **Low-Latency Direct:** latency around 1–2 seconds; smoother interaction; ideal for small classes/hands-on sessions

3.2 Price

- Standard Live is cheaper

- Low-Latency Direct has stronger bandwidth and node guarantees, so it costs a bit more

3.3 Other Questions

- Answer per knowledge base, then proceed to Step 4

(a) Instruction example (Page 1).

Step 4: Confirm Front-End Visibility

4.1 Ask About Publishing Method

Ask: Do you publish via the Web Console, School Information System A, or SaaS System B?

Web Console:

- If Low-Latency Direct already appears → use directly

- If not visible → you'll configure it on the backend; ask them to check tomorrow

Third-Party System:

- If visible → choose as needed

- If not visible → slow guided enablement (pause 3 seconds per step):

1. Go to [Me]

2. Tap [Service Provider / Streaming Platform Management]

3. Select [Streaming Platform]

4. Under [Service Products], check Low-Latency Direct, then save

Step 5: Check Any Learner-Side Fee / Accelerated Line Fee (if used)

- No fee set → proceed to Step 6

- Fee set → remind them to ensure it also applies to Low-Latency Direct

If they can't configure it → slow guided setup (pause 3 seconds per step):

1. [Me] → [Academic/Finance Settings]

2. → [Charges & Rules]

3. → edit Streaming Line Surcharge and enable it for Low-Latency Direct

4. Save

Step 6: Add via Corporate WeChat

- If current number can be added → say you will add them via Corporate WeChat shortly; please approve

- If it cannot → request a mobile number that can be added; then the same approval message

Step 7: Close

- Address any remaining questions per knowledge base

- If none, wish them smooth classes and full enrollments, then end the call

(b) Instruction example (Page 2).

Figure 5: Multi-turn instruction example: Customer Support Specialist for a Course Publishing Platform.

Prompt A — Segmentation (Instruction → Segments)

```
You are given a fully specified instruction for a conversational AI assistant, and your task is to segmen

**Important Context**: These segments will be used to generate evaluation criteria for assessing whether

You must output a list of segments in the following JSON format:
[
    {"segment": "[exact excerpt from the instruction]"},
    {"segment": "[exact excerpt from the instruction]"},
    ...
]

Rules:
- [Non-overlapping] The segments must be non-overlapping and cover the entire instruction. You can option
- [Minimalistic] Split information into units, but maintain meaningful groupings. If an expression includ
- [No excessive splitting] Avoid breaking down compound expressions into individual units unless it adds
- [Conversation-focused] Focus on segments that describe observable behaviors, communication patterns, or

Example Query:
What are the names and locations of the stadiums that had concerts that occurred in both 2014 and 2015?

Output:
{"segments": [
    {"segment": "names and locations"},
    {"segment": "stadiums"},
    {"segment": "concerts"},
    {"segment": "in both 2014"},
    {"segment": "and 2015"}
]}

Now complete the task for the following fully specified instruction:

[[INSTRUCTION]]
```

Figure 6: Prompt A: Segmentation (Instruction → Segments).

Prompt B — Criteria (Evaluation Question Generation)

```
You are given segments of a conversational AI instruction. Your task is to:
1. Convert each segment into an evaluation question that assesses observable conversation behaviors.
2. For segments with conditional logic, include explicit dependency conditions using "if..." or "when..."
3. **Handle recommended scripts carefully** - preserve key details while maintaining clarity.

**ZERO-HALLUCINATION & GROUNDING RULES (CRITICAL):**
- All criteria MUST be directly grounded in the provided "Instruction" and "Segments".
- DO NOT introduce any new entities, settings, roles, numbers, limits, formats, or constraints that are n
- Only paraphrase; do not expand scope. If the input does not specify a quantity/limit (e.g., "no more th
- If a segment is broad or ambiguous, write the dimension as an observable action without adding extra sp
- Do not carry constraints across segments unless the dependency is explicitly stated. Each dimension mus

**Core Guidelines:**
- Focus on **observable actions** in conversations, not knowledge testing.
- **Replace step references with concrete action descriptions**: Instead of "proceed to the next step" or
- **Emphasize complex dependency conditions**: For multi-conditional logic, make dependencies crystal cle
- Use specific, measurable criteria ONLY IF such metrics are explicitly present in the source text. Other
- Avoid vague conditions like "in relevant situations".
- Do not infer missing details such as locations, counts, personas, or formatting if not provided.

**OUTPUT ANCHORING RULE (CRITICAL):**
- **R1 - Anchor to Output Artifact**: Always reference "the generated [specific artifact]" (e.g., "the ge

**Script Handling - SUMMARY + DETAIL APPROACH:**
- **IMPORTANT DISTINCTION**: Only apply to **conversational scripts/dialogue content**, NOT to format/con
- **What qualifies as a "script"**: Specific dialogue content, suggested wording, recommended phrases for
- **What does NOT qualify**: Format requirements (e.g., "exactly 10 words", "JSON format"), length constr
- **Two-part structure for recommended scripts**: First summarize the core intent, then include the speci
- **Format**: "Does the generated [artifact] [summary of the script's intent], with the specific recommen
- **CRITICAL: Intelligent Summarization Rules**:
  - **DO NOT repeat the script content in the summary.**
  - **Extract the BEHAVIORAL INTENT or COMMUNICATION PURPOSE.**
  - **Use different words to describe the ACTION, not the content.**
  - **Focus on WHAT the agent is trying to achieve, not WHAT they say.**
- **Avoid Redundancy**: The summary should capture the essence/intent, while the script provides the exac
- Only apply this when a recommended script is actually present. Do NOT fabricate scripts.
- **Examples of WHAT QUALIFIES as Scripts (use Script Handling)**:
  - ✓ "The discount can be as low as 60%, but not every order gets a 60% discount." → Summary: "clarify
  - ✓ "I'm very sorry, perhaps my information was incorrect. My apologies for bothering you, please hang
  - ✓ "We apologize, but we have made many improvements now. Would you like to try it again?" → Summary:
  - ✓ "The shipping subsidy is not issued as a coupon; the system will automatically apply a discount to

- **Examples of WHAT DOES NOT QUALIFY as Scripts (direct evaluation, no Script Handling)**:
  - ✗ "exactly 10 words" → Direct: "Does the generated response contain exactly 10 words?"
  - ✗ "no more than 3 sentences" → Direct: "Does the generated response contain no more than 3 sentences
  - ✗ "JSON format" → Direct: "Is the generated response in valid JSON format?"
- **Then attach full script**: Include the complete recommended script for verification purposes.
- **Balance completeness with clarity**: Ensure both summary and detail are present.

**Dependency Condition Requirements:**
- **Multi-step dependencies**: When a dimension depends on complex prior conditions, make ALL conditions
- **Conditional branching**: Clearly specify which condition leads to which specific action.
- **Sequential dependencies**: Make the order and prerequisites crystal clear.
- Do NOT transfer constraints between branches unless explicitly stated.

**Anti-Extrapolation Examples:**
- BAD: Adding "no more than 7 sentences" when no such limit exists in the input.
- BAD: Forcing a "restaurant" setting when the input only says "create a scene".
- GOOD: If the segment says "confirm the user is a merchant, then inform delivery subsidy", write: "When
Your output must be valid JSON only:
```

Figure 7: Prompt B: Criteria generation (evaluation question generation).

Prompt C — Verification (Coverage Judgement)

```
You are given an instruction that fully specifies a problem, and a list of dimensions. Your task is to de

The dimensions should cover the essential elements needed to complete the task. Minor details or optional

If the core requirements are not covered, you should output the critical information unit from the instru

Example 1:

Instruction:
What are the names and locations of the stadiums that had concerts that occurred in both 2014 and 2015?

Dimensions:
{"initial_segment": "stadiums", "initial_dimension": "I'm looking for active stadiums", "dimensions": [{"

Output:
{"coverage": "complete"}


Example 2:
Instruction:
Which Asian countries have a population that is larger than any country in Africa?

Dimensions:
{"initial_dimension": "I'm interested in learning about countries in Asia", "dimensions": [{"dimension":

Output:
{"coverage": "incomplete", "missing_segment": "the dimensions do not specify that the population of the A



You must output in JSON format as shown in the examples above.
Now complete the task for the following fully specified instruction and dimensions:

Instruction:
[[QUERY]]

Dimensions:
[[DIMENSIONS]]
```

Figure 8: Prompt C: Verification (completeness judgment).

C Decomposition Comparison Between Ours and Human

In this section, we provide further analysis comparing the results of dimensions decomposed by humans versus those automatically extracted with our methods.

C.1 Statistics of Decomposition Results

We first investigate the number of derived dimensions. Figure 10 presents a comparison between human-decomposed and LLM auto-extracted dimensions. Overall, both approaches yield similar ranges of dimension counts (typically 4–7), suggesting broad alignment. However, the difference analysis shows that the LLM often extracts slightly more dimensions, while the distribution comparison highlights that the LLM produces a heavier tail with higher counts (occasionally exceeding 15), indicating a tendency towards a finer-grained decomposition relative to human annotators.

C.2 Further Analysis of Dissimilar Pairs

We further investigate the subset of instructions where the sentence-level BERT cosine similarity between human and LLM decompositions falls below 0.85, indicating notable disagreement. For these divergent cases, a closer examination reveals that the LLM-as-judge evaluation consistently favors our LLM-based decomposition. Specifically, our method achieves substantially higher scores across six evaluation dimensions, significantly surpassing the overall average performance. This suggests that when disagreements occur, the LLM-extracted decompositions tend to capture finer-grained and more useful structures than human annotations.

D Conversation Generation Prompts

The conversation generation module leverages the constructed user profiles to synthesize realistic dialogues under different cooperativeness levels. The process can be summarized as follows:

Profile-driven user simulation. We first map each cooperativeness score (15) to a corresponding *Meta User Profile*, which encodes communication style, information disclosure preferences, problemsolving approaches, and typical catchphrases. These profiles are then injected into a simulator prompt template, forming the role description for simulated users.

Dialogue initialization. Each dialogue starts with the assistant issuing an *opening line* defined in the instruction template. The simulator then generates user responses conditioned on the profile, ensuring that behaviors are consistent with the specified cooperativeness level.

Turn-taking and control. Dialogues proceed in alternating turns between the assistant (task-driven agent) and the simulator (profile-driven user). We cap the maximum number of rounds per dialogue to maintain controllability and diversify the interaction lengths.

End-of-conversation detection. The system incorporates a hybrid mechanism to decide when a dialogue should terminate. On the one hand, strict keyword rules capture explicit hangup expressions (e.g., goodbye, stop calling). On the other hand, an LLM-based detector jointly evaluates conversation history and the latest user response to identify either (i) user hangup intent or (ii) the assistant reaching its final closing step. A dialogue is ended once either condition is satisfied, with the reason (user termination vs. assistant completion vs. max rounds reached) logged.

D.1 Meta User Profiles

In this paper, we present only the results of our *Meta User Profiles*, as the categorization of users is not fixed and can be flexibly defined or adapted to the requirements of different application scenarios. Nevertheless, our approach offers a convenient and generalizable framework for constructing conversation datasets with only minor modifications.

We construct user profiles in three stages:

- **(i)** Raw dialogue data are first categorized by cooperativeness levels using an automated grader.
- **(ii)** Dialogues within the same category are aggregated into *Meta User Profiles* (see Figure 11 for details).
- **(iii)** Given downstream instructions, these meta profiles are further adapted into *Adjusted User Profiles*.

D.2 Adjusted User Profiles

As mentioned in Appendix D.1, we first construct *Meta User Profiles* by aggregating dialogues within the same cooperativeness category. To adapt

these category-level profiles to specific downstream tasks, we further generate *Adjusted User Profiles*. The adjustment procedure proceeds as follows:

- **Instruction analysis.** Given a new instruction, we use an LLM to extract its business context, key topics, user concerns, technical terms, typical questions, and intended user goals.
- **Behavioral pattern update.** Based on the analysis, we refine the meta profiles communication style, information disclosure tendencies, and problem-solving approaches, inserting instruction-specific examples (e.g., replacing generic platform references with task-relevant business contexts).
- **Catchphrase enrichment.** We augment the profiles common catchphrases by adding 12 representative questions or terms derived from the instruction (e.g., frequent user questions about new technical terms).

This procedure ensures that the originally general-purpose *Meta User Profiles* are minimally but effectively adapted to the requirements of each instruction, yielding instruction-specific *Adjusted Profiles* that remain compatible with our simulation pipeline.

E Logic Extraction Prompts

See Figure 12 for our automatic logic extraction prompts. Used to extract a *tree-structured logic* over the derived dimension set, resulting in a forest of precondition groups.

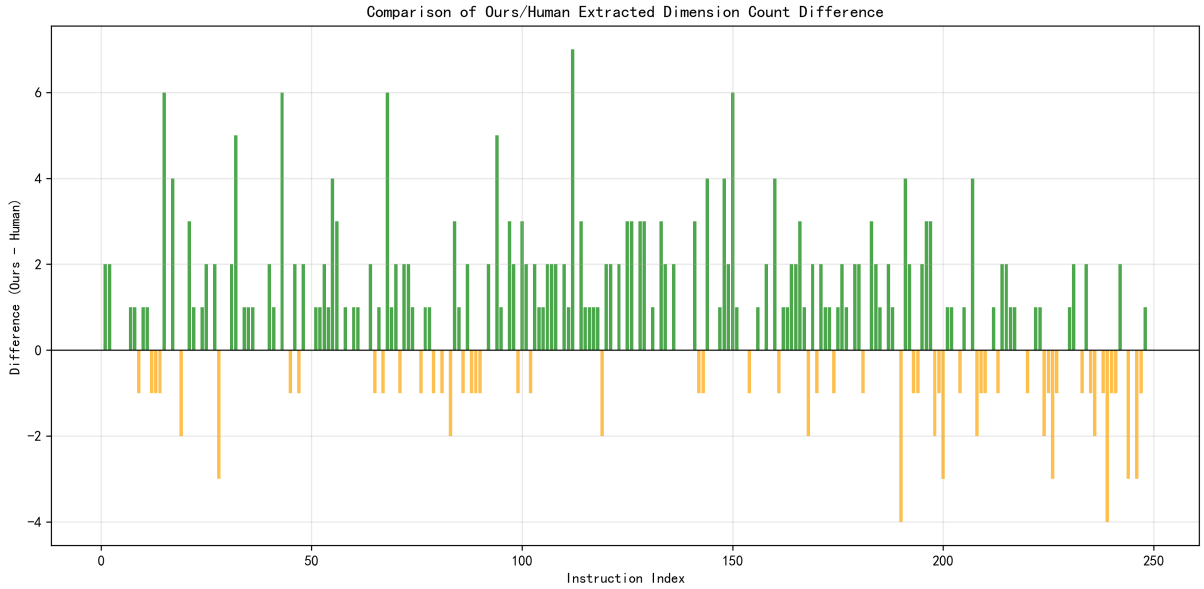
F Detailed Evaluation Results

See Figure 13 for specific derived dimensions and Figures 14 and 15 for detailed coverage and achievement for all models among all dimensions.

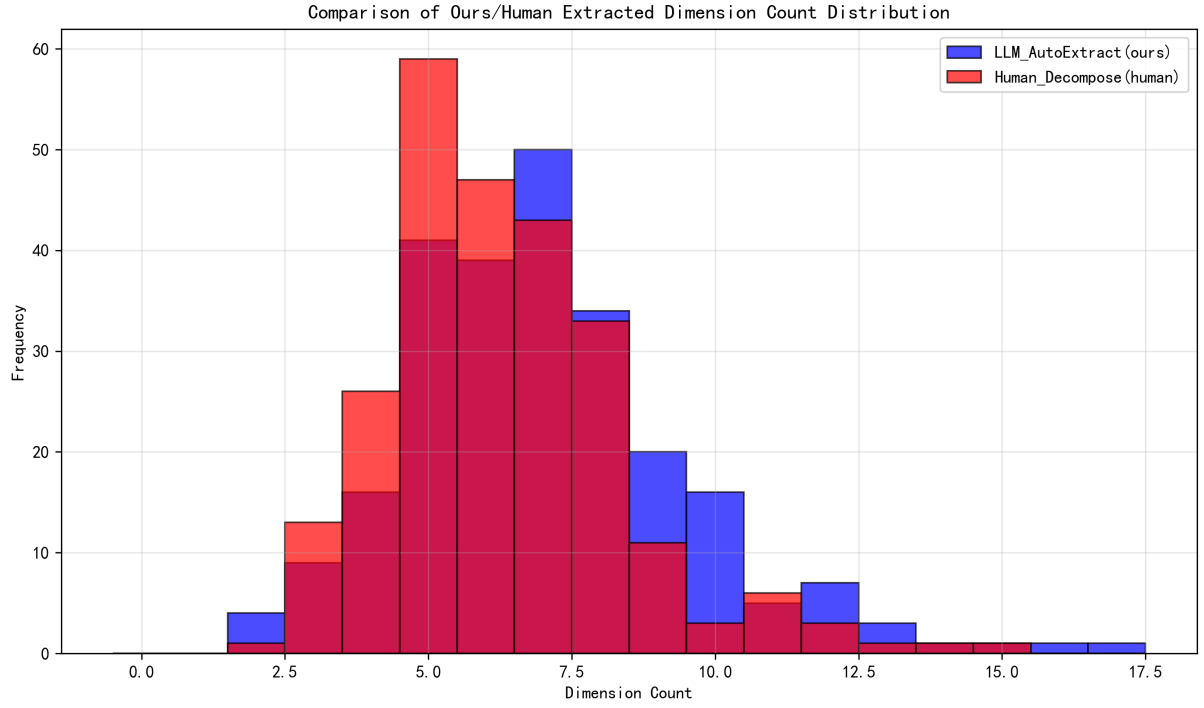
Prompt D — Modification (Refine Dimensions by Feedback)

```
You are an expert specialized in modifying and improving conversational dimensions. Your task is to modif

## Input Information:
1. **Original Instruction**: Complete task instruction
2. **Current Dimensions**: Existing dimension structure
3. **Verification Feedback**: Specific information about what is missing or insufficient

## Modification Principles:

### 1. Maintain Structural Integrity
- Keep the original JSON structure format: `{"dimensions": []}`
- The dimensions array should contain all important evaluation points

### 2. Precise Modifications Based on Feedback
- **If feedback indicates missing information**: Add new dimensions to cover the missing content
- **If feedback indicates unclear expression**: Rephrase relevant dimensions to make them clearer
- **If feedback indicates missing logic**: Add necessary conditional dependency relationships
- **If feedback indicates insufficient detail**: Enhance the specificity and measurability of relevant di

### 3. Follow Conversational Evaluation Best Practices
- **Focus on observable behaviors**: Ensure each dimension can be verified through specific behaviors in
- **Clear conditional dependencies**: Use "when..." or "if..." to clearly express conditional relationshi
- **Concrete step references**: Replace abstract "next step" with specific action descriptions
- **Maintain script integrity**: For recommended scripts, use "intent summary + specific script" dual str

### 4. Quality Assurance Principles
- **Completeness**: Ensure coverage of all key information points in the original instruction
- **Accuracy**: Each modified dimension should accurately reflect corresponding instruction requirements
- **Measurability**: Each evaluation criterion should be verifiable through conversation observation
- **Logical clarity**: Conditional branches and dependency relationships should be clearly expressed

## Output Requirements:
- Output the complete modified dimensions JSON structure
- Ensure JSON format is correct and directly parseable
- Output only JSON, no other explanatory text

## Example Modification Scenarios:

**Scenario 1: Missing Key Information**
- Feedback: Missing handling of specific conditions
- Modification: Add new dimensions or supplement missing conditions in existing dimensions

**Scenario 2: Insufficient Specificity**
- Feedback: A certain dimension is vaguely expressed
- Modification: Rephrase that dimension to make it more specific and measurable

**Scenario 3: Missing Dependencies**
- Feedback: Missing logical connections between steps
- Modification: Add clear conditional dependency expressions in relevant dimensions

Now please modify the dimensions based on the following information:

Original Instruction:
[[INSTRUCTION]]

Current Dimensions:
[[CURRENT_DIMENSIONS]]

Verification Feedback:
[[VERIFICATION_FEEDBACK]]
```

Figure 9: Prompt D: Modification (refine dimensions by feedback).



(a) Difference of number of dimensions from Human decomposed and LLM auto-extracted.



(b) Distribution of number of dimensions from Human decomposed and LLM auto-extracted.

Figure 10: Comparison between human-decomposed and LLM auto-extracted dimensions: (a) difference in number of dimensions, (b) distribution of dimensions.

Aggregated User Profiles (Meta)


```
{
  "Poor cooperation": {
    "profile": {
      "cooperativenessCategory": {
        "category": "Poor cooperation"
      },
      "summary": "Users in this category show obvious defensiveness, perfunctory responses, or impatience in co
      "behavioralPatterns": {
        "communicationStyle": "Mainly uses brief, repetitive, questioning, perfunctory, or irrelevant statement
        "informationDisclosure": "Low willingness to disclose information, rarely actively provides information
        "problemSolvingApproach": "Problem-solving approach is mainly avoidance, evasion, and perfunctory respo
      },
      "inferredAttributes": {
        "primaryGoals": [
          "Maintain existing operational habits, avoid additional learning or changes",
          "Save time, reduce irrelevant or lengthy communication, pursue operational simplicity"
        ],
        "commonCatchphrases": [
          "You say (then)",
          "What's the matter (ah/er)?",
          "What happened?",
          "Who is this?",
          "Who are you?",
          "Then what?",
          "Is there something?",
          "Are you a robot?",
          "No need/Don't need/No time/Goodbye",
          "What's the difference?",
          "Has the price increased now?"
        ],
        "impliedPersonalityTraits": [
          "Strong defensiveness, suspicious or distrustful attitude toward external communication",
          "Lack of patience, easily shows impatience or perfunctory responses",
          "Resistant to change, tends to maintain status quo",
          "Passive coping, low initiative",
          "Focus on practical benefits, low interest in details and process optimization"
        ]
      }
    }
  },
  "General cooperation": {
    "profile": {
      "cooperativenessCategory": {
        "category": "General cooperation"
      },
      "summary": "Users in this category show basic willingness to cooperate but have limited initiative and en
      "behavioralPatterns": {
        "communicationStyle": "Communication style is mainly concise and direct, with mostly neutral or slightl
        "informationDisclosure": "Information disclosure is mainly passive. Users mostly provide required infor
        "problemSolvingApproach": "When encountering problems, users tend to first express questions or confusi
      },
      "inferredAttributes": {
        "primaryGoals": [
          "Ensure simple and clear processes, convenient operations",
          "Focus on speed, costs, and service quality, avoid platform changes affecting their own interests"
        ],
        "commonCatchphrases": [
          "How to operate?",
          "What's the difference from before?",
          "Will the price change?",
          "How to calculate overtime?",
          "Are you a robot?",
          "I've always used direct service",
          "You say it",
          "Can/Can't",
          "Say it again"
```


Figure 11: Aggregated Meta User Profiles constructed from categorized dialogue data.

You are a professional evaluation dimension analysis expert. You will be provided with a set of evaluation dimension descriptions used to assess AI assistant performance in specific tasks.

Below is the complete list of evaluation dimensions:

{dimensions_list}

Task Description:

Please identify dimension combinations that have mutually exclusive relationships. Focus on the following mutual exclusion patterns:

Mutual Exclusion Pattern Examples:

- Dimension A: Whether to actively ask the boss for a phone number to add WeChat

- Dimension B: Whether to explain that Meituan Corporate WeChat will be used to add the boss's WeChat when confirming the current phone number can add WeChat

- Dimension C: Whether to reference knowledge points to request a phone number that can add WeChat when the current phone number cannot add WeChat

The characteristic of this pattern is: there is a basic action, then based on different results of that action (success/failure, yes/no, satisfied/unsatisfied, etc.), it branches into multiple mutually exclusive subsequent behavior paths.

Identification Criteria:

1. **Conditional Branch Mutual Exclusion**: Look for dimension combinations based on opposing or mutually exclusive conditions, including:

- "In the case of..." vs "In the case of not..."

- "In the case of merchants using Platform A" vs "In the case of merchants using Platform B"

- "When... indicates awareness" vs "When... indicates unawareness"

- "In the case of... being able to..." vs "In the case of... not being able to..."

2. **Logical Exclusivity**: In the same dialogue scenario, these dimensions cannot be simultaneously satisfied (Note: exclude sequential actions)

3. **Same Context**: Different processing paths around the same core task or scenario

4. **Strict Mutual Exclusion Requirement**: Each mutual exclusion group must contain at least 2 dimensions, and these dimensions must be based on opposing conditions or states

5. **Strictly Based on Original List**: Can only identify mutual exclusion relationships from the provided dimensions_list above, absolutely no fabrication, creation, or speculation of non-existent dimensions

Key Keyword Patterns to Focus On:

- "In the case of..." vs "In the case of not..."

- "In the case of merchants using..." vs "In the case of merchants using..." (different platforms/tools)

- "When..." vs "When not..."

(a) Page 1

- "When... indicates..." vs "When... indicates not..."

- "In the case of... being able to..." vs "In the case of... not being able to..."

- "If... then..." vs "If not... then..."

Important Exclusion Rules:

- **Exclude Sequential Actions**: If two dimensions are sequentially occurring actions (e.g., first requesting WeChat ID, then confirming WeChat ID), they don't count as mutually exclusive

- **Exclude Single Dimensions**: If there's only one dimension without a corresponding mutually exclusive dimension, it should not be identified as a mutual exclusion group

- **Exclude Time Sequences**: If dimensions describe different time points in the same process, they don't count as mutually exclusive

- **No Fabrication Allowed**: Absolutely no creation, fabrication, or speculation of any dimensions not in the original dimensions_list to form mutual exclusion pairs

Output Format:

Please output the identified mutually exclusive dimension groups in the following JSON format:

```json

{{

"exclusive_groups": [

{{

"group_id": 1,

"description": "The core scenario and branching logic of this mutually exclusive dimension group",

"base_action": "Base action dimension (must be a specific dimension from dimensions_list, if no clear base action dimension exists, fill in 'None')",

"mutual_exclusive_type": "Conditional branching type",

"dimensions": [

{{

"content": "Complete description of the dimension (must be exactly the same as the dimension in dimensions_list)",

"condition_type": "The condition type of this dimension (e.g., base action/success branch/failure branch)",

"key_condition": "Key condition description"

}},

{{

"content": "Complete description of another dimension (must be exactly the same as the dimension in dimensions_list)",

"condition_type": "The condition type of this dimension",

"key_condition": "Key condition description"

}}

(b) Page 2

Figure 12: Exclusive pattern identification template.
```

Appendix: Optimized Dimensions

1. When the generated conversation confirms that the person is in charge, does the generated conversation move on to checking their prior awareness of the live streaming setup?

2. When the generated conversation determines the person is not in charge, does the generated conversation ask them to relay the message and then move on to checking prior awareness of the live streaming setup?

3. Does the generated conversation inform the user about the product upgrade and provide instructions for selecting the new option, with the specific recommended script: 'We've upgraded the live streaming product and added a separate "Low-Latency Direct" option. When you publish a class, just choose Low-Latency Direct; everything else remains unchanged.'

4. Does the generated conversation ask whether the user was aware that their previous Standard Live selection was actually routed via a low-latency line on the backend to protect quality?

5. When the generated conversation determines the user was not aware, does the generated conversation explain that the front-end wasn't open yet and that low latency was temporarily enabled to ensure audio/whiteboard sync?

6. When the generated conversation determines the user was aware, does the generated conversation move on to communicating the upgrade and differences between options?

7. Does the generated conversation inform the user that both options will be listed separately on the publishing page and advise them to choose according to the course type, with the specific recommended script: 'Going forward, both options will be listed separately on the publishing page. Choose according to the course type.'

8. Does the generated conversation describe Standard Live as having lower cost, latency around 5–10 seconds, and being suitable for large lectures?

9. Does the generated conversation describe Low-Latency Direct as having latency around 1–2 seconds, smoother interaction, and being ideal for small classes or hands-on sessions?

10. Does the generated conversation state that Standard Live is cheaper than Low-Latency Direct?

11. Does the generated conversation state that Low-Latency Direct has stronger bandwidth and node guarantees, resulting in a slightly higher cost?

12. Does the generated conversation answer any other questions according to the knowledge base before moving on to confirming front-end visibility?

13. Does the generated conversation ask the user whether they publish via the Web Console, School Information System A, or SaaS System B?

14. When the generated conversation determines that Low-Latency Direct already appears in the Web Console, does the generated conversation instruct the user to use it directly?

15. When the generated conversation determines that Low-Latency Direct is not visible in the Web Console, does the generated conversation inform the user that it will be configured on the backend and ask them to check again tomorrow?

16. When the generated conversation determines that Low-Latency Direct is visible in the third-party system, does the generated conversation instruct the user to choose it as needed?

(a) Page 1

17. When the generated conversation determines that Low-Latency Direct is not visible in the third-party system, does the generated conversation provide a slow, step-by-step guided enablement with a 3-second pause per step: Go to [Me]; Tap [Service Provider / Streaming Platform Management]; Select [Streaming Platform]; Under [Service Products], check Low-Latency Direct, then save?

18. When the generated conversation determines that no learner-side fee or accelerated line fee is set, does the generated conversation move on to adding via Corporate WeChat?

19. When the generated conversation determines that a learner-side fee or accelerated line fee is set, does the generated conversation remind the user to ensure the fee also applies to Low-Latency Direct?

20. When the generated conversation determines that the user cannot configure the fee, does the generated conversation provide a slow, step-by-step guided setup with a 3-second pause per step: [Me] → [Academic/Finance Settings]; → [Charges & Rules]; → edit Streaming Line Surcharge and enable it for Low-Latency Direct; Save?

21. When the generated conversation determines that the current number can be added via Corporate WeChat, does the generated conversation inform the user that they will be added shortly and request approval?

22. When the generated conversation determines that the current number cannot be added via Corporate WeChat, does the generated conversation request a mobile number that can be added and then provide the same approval message?

23. Does the generated conversation address any remaining questions according to the knowledge base?

24. When there are no remaining questions, does the generated conversation wish the user smooth classes and full enrollments, then end the call?

(b) Page 2

Figure 13: Derived dimensions displayed side by side.

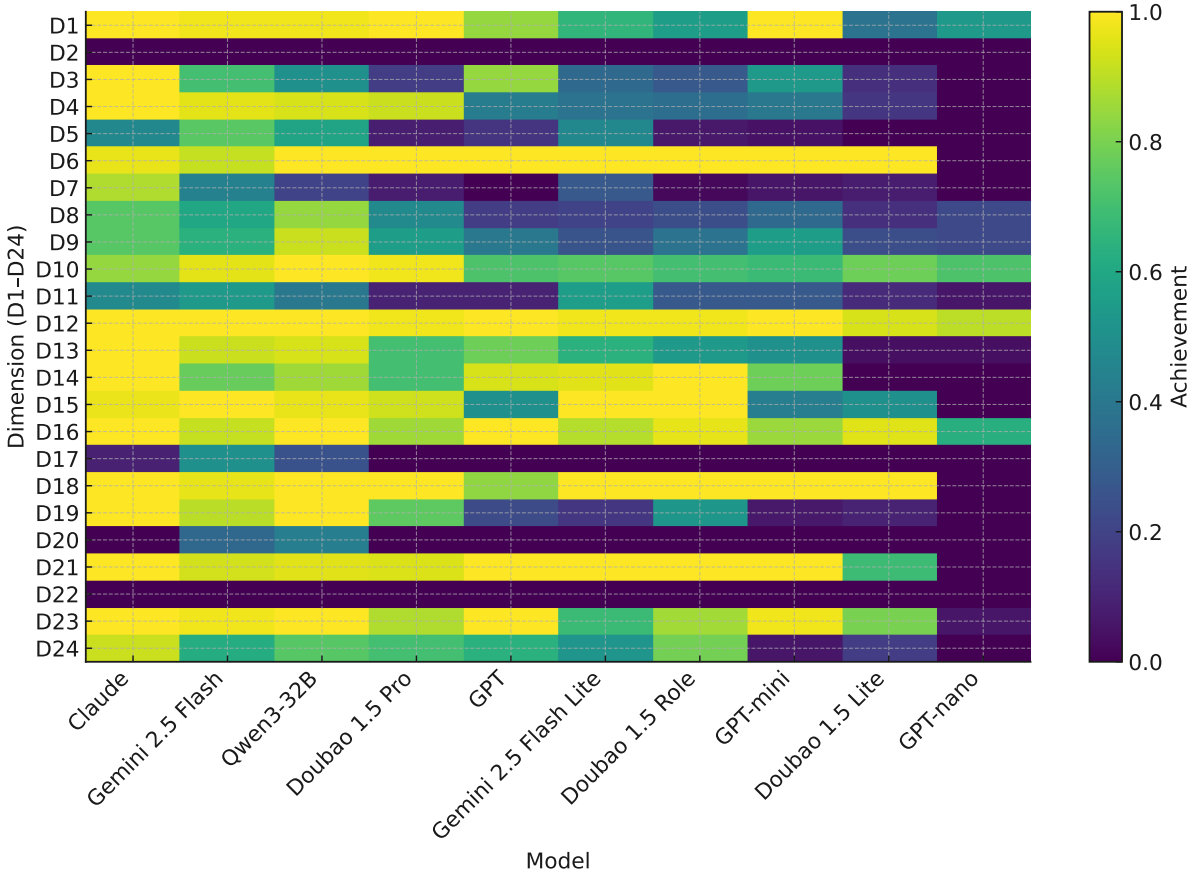


Figure 14: Per-dimension Achievement heatmap.

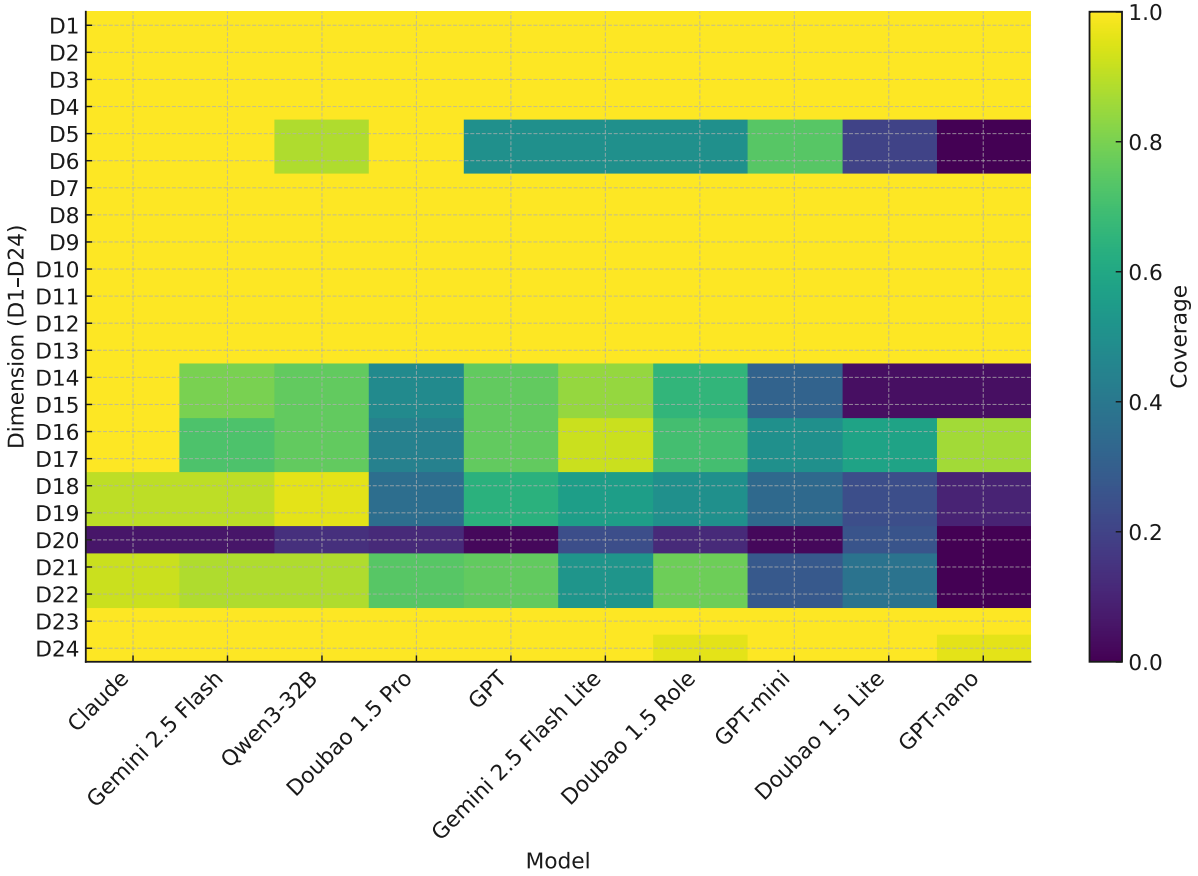


Figure 15: Per-dimension Coverage heatmap.