clem:todd: A Framework for the Systematic Benchmarking of LLM-Based Task-Oriented Dialogue System Realisations

Chalamalasetti Kranti[1], Sherzod Hakimov[1], David Schlangen[1,2]

[1]Computational Linguistics, Department of Linguistics

University of Potsdam, Germany

[2]German Research Center for Artificial Intelligence (DFKI), Berlin, Germany

{kranti.chalamalasetti, sherzod.hakimov, david.schlangen}@uni-potsdam.de

Abstract

The emergence of instruction-tuned large language models (LLMs) has advanced the field of dialogue systems, enabling both realistic user simulations and robust multi-turn conversational agents. However, existing research often evaluates these components in isolation, either focusing on a single user simulator or a specific system design, limiting the generalisability of insights across architectures and configurations. In this work, we propose clem:todd (chat-optimized LLMs for task-oriented dialogue systems development), a flexible framework for systematically evaluating dialogue systems under consistent conditions. clem:todd enables detailed benchmarking across combinations of user simulators and dialogue systems, whether existing models from literature or newly developed ones. To the best of our knowledge, clem:todd is the first evaluation framework for task-oriented dialogue systems that supports plug-and-play integration and ensures uniform datasets, evaluation metrics, and computational constraints. We showcase clem:todd's flexibility by re-evaluating existing task-oriented dialogue systems within this unified setup and integrating three newly proposed dialogue systems into the same evaluation pipeline. Our results provide actionable insights into how architecture, scale, and prompting strategies affect dialogue performance, offering practical guidance for building efficient and effective conversational AI systems.

1 Introduction

Task-oriented dialogue systems (McTear, 2002; Li et al., 2017; Eric et al., 2017; Budzianowski et al., 2018; Balakrishnan et al., 2019; Chen et al., 2019; Elder et al., 2020) help users complete specific goals, such as booking travel or making reservations, through multi-turn natural language interactions. These systems must accurately interpret user intent, manage ambiguity, and respond coherently.

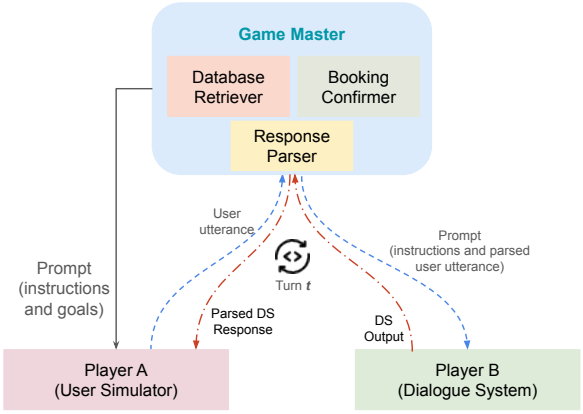

Figure 1: clem:todd framework facilitates turn-based interactions between a user simulator (Player A) and a dialogue system (Player B), coordinated by a Game Master module.

With the rise of LLMs, these systems (Hudecek and Dusek, 2023; Xu et al., 2024) have significantly improved in handling task-oriented, goal-driven conversations. In addition to enhancing dialogue system capabilities, LLMs now play dual roles: as end-to-end dialogue systems (Chung et al., 2023; Dong et al., 2025) and as user simulators (Kojima et al., 2022; Kazi et al., 2024) for training and evaluation. However, evaluation remains a challenge (Casas et al., 2020; Braggaar et al., 2023). Current evaluations often suffer from inconsistent datasets, metrics, and compute settings, making it difficult to compare models or draw conclusions about system design.

A recent paradigm for evaluating LLMs as agents, involves self-play in conversational "games". Frameworks such as clembench (Chalamalasetti et al., 2023), GameEval (Qiao et al., 2023), SPAG (Cheng et al., 2024), and TextArena (Guertler et al., 2025) assess the capabilities of LLMs in instruction following, logical reasoning, and context retention across multi-turn settings. However, they do not address the evaluation of interactive systems built on LLMs, such as task-oriented dialogue (TOD) systems.

To address this gap, we introduce `clem:todd` [1] (see Figure 1), an evaluation framework that builds on the self-play framework setup for TOD. The user simulator and dialogue system act as the two players in the self-play setup, with a central controller (game master) managing turn-taking, task flow, and interactions with external tools such as databases. This setup supports consistent benchmarking, stress testing, and detailed analysis under shared datasets, metrics, and resource constraints.

Using `clem:todd`, we explore three research questions: (1) Can LLM-based self-play be adapted to evaluate interactive TOD systems? (2) On the MultiWOZ2.2 benchmark, a widely used dataset for multi-domain TOD, how do user simulators and dialogue systems perform across different models and architectures? (3) Can `clem:todd` facilitate evaluation beyond MultiWOZ to plug in new instances, either derived from other datasets or generated synthetically, to enhance robustness testing?

We benchmark both existing dialogue systems and three newly developed ones within our framework, providing systematic comparisons and insights across a range of configurations. Our contributions are as follows: (a) we propose `clem:todd`, a self-play-based evaluation framework for system-level benchmarking of TOD systems; (b) we re-evaluate existing systems and our proposed dialogue systems under the same benchmarking setup; and (c) we analyse trade-offs across architectures, model sizes, and user simulators, with a focus on task performance and computational cost.

2 Related Work

Our work builds on research in: LLM-based TOD systems, user simulators, task evaluation, and self-play frameworks for multi-turn interaction.

LLM-Based TOD Systems leverage LLMs in diverse architectural setups to enable successful task completion. Hudecek and Dusek (2023) proposed a modular system using LLMs across functional components, while InstructTOD (Chung et al., 2023) and ProTOD (Dong et al., 2025) extended this design with performance-oriented enhancements. In contrast, AutoTOD (Xu et al., 2024) introduced a zero-shot monolithic system, showcasing the potential of LLMs as end-to-end systems. DARD (Gupta et al., 2024) proposed a multi-agent framework, coordinating domain-specific LLMs via a central LLM-based dialogue manager.

LLM-Based User Simulators use LLMs to generate realistic, goal-driven user interactions for training and evaluating dialogue systems. Kojima et al. (2022); Davidson et al. (2023) leveraged in-context learning to produce diverse and contextually appropriate responses. Sekulic et al. (2024) fine-tuned LLMs on domain-specific data to mitigate hallucinations, while Kazi et al. (2024) proposed an adaptive simulator that dynamically responds to the dialogue system it interacts with.

LLM-Based Task Evaluation explores using LLMs as evaluators to assess dialogue quality and task success. Recent work has prompted LLMs to act as judges in both open-ended (Zheng et al., 2023) and task-specific contexts (Liu et al., 2023b; Thakur et al., 2024; Kim et al., 2024; Chan et al., 2024; Bavaresco et al., 2024). Jia et al. (2024) explored automatic dialogue evaluation using LLMs, and Hashemi et al. (2024) introduced LLM-Rubric, a framework for structured, rubric-based evaluation of language outputs.

Frameworks for Dialogue System Evaluation Ghandeharioun et al. (2019) propose an evaluation framework for dialogue-system self-play in open-domain dialogue systems. Cheng et al. (2022) discuss the importance of using a goal-oriented user simulator for improved evaluation. Xu et al. (2024) showcase the evaluation of different dialogue systems against a single user simulator. Additionally ConvLab (Zhu et al., 2020) and ParlAI[2] offer infrastructure for developing dialogue agents, while DialEvalMetrics (Yeh et al., 2021) focuses on evaluation. However, none provide a unified setup for controlled, plug-and-play experimentation across LLM-based user simulators and dialogue systems, evaluated with consistent metrics and compute constraints. `clem:todd` fills this gap by enabling systematic exploration of architectural choices, simulator–system interactions, and scaling trade-offs.

LLM-Based Self-Play Frameworks Self-play has recently been used to evaluate LLMs on multi-turn tasks in frameworks like clembench (Chalamalasetti et al., 2023), GameEval (Qiao et al., 2023), SPAG (Cheng et al., 2024), and TextArena (Guertler et al., 2025). These systems focus on model-level evaluation, while `clem:todd` adapts self-play for benchmarking full dialogue systems.

[1] https://github.com/clp-research/clem-todd

[2] https://parl.ai/

Building on these foundations, we propose clem:todd an evaluation framework that integrates LLM-based user simulators, dialogue systems, and automatic evaluation. It supports flexible combinations of monolithic, modular, and multi-agent architectures and evaluates them under consistent metrics and compute settings. This design enables comprehensive benchmarking and reveals system interaction dynamics in controlled conditions.

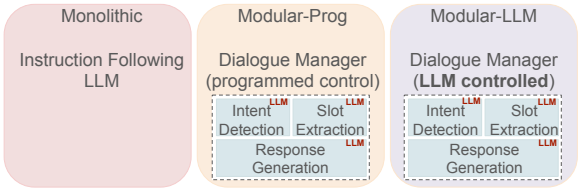

Figure 2: Overview of proposed dialogue system architectures. Monolithic, Modular-Prog, and Modular-LLM configurations vary in how control and components are handled by LLMs.

3 Methodology

We frame task-oriented dialogue as a form of "Assistance Game" (Laidlaw et al., 2024), where one player needs assistance completing a task, and the other must work toward recovering their reward function. While Laidlaw et al. (2024) address the learning problem, we use this framing to develop an evaluation framework for testing different approaches to building dialogue systems.

3.1 "Self-Play" of Task-Oriented Dialogue

We build on *clembench* (Chalamalasetti et al., 2023), a framework that realises conversational games between two (LLM-simulated) players coordinated by a Game Master. clem:todd reuses and modifies the two-player setup as: (a) Player A is an LLM-based user simulator that receives a goal and a prompt (see Appendix 6) describing the task; (b) Player B is a dialogue system that receives user simulator utterances and a prompt (see Appendix 7 and 8) describing the task, with the objective of satisfying Player A's goal; and (c) the Game Master orchestrates the interaction by passing utterances between the two players and managing turn-level coordination.

The Game Master enforces stricter format constraints than the base *clembench* setup by requiring adherence to a predefined Tool Schema (in JSON format) (*e.g.*, `followup`, `querydb`, `validatebooking`) to reflect the demands of real-world deployment. Any violation aborts the conversation, reinforcing that instruction following and format compliance is essential. Valid tool calls are executed programmatically, either to retrieve database results, validate booking details, or forward follow-up messages to the user simulator. The schema is given in Figures 13-18 in Appendix.

A full interaction (see Figure 1) proceeds as follows: the user simulator (Player A) is primed with a goal (*e.g.*, booking a train ticket) and generates a natural language utterance (*e.g.*, "I am looking for a train to Paris from London"). This utterance is passed to the dialogue system (Player B) via the Game Master, which then returns a response (*e.g.*, "Do you have a specific day/time in mind?") to the simulator. The loop continues until the conversation ends when Player A signals by either generating the DONE token or when the maximum turn limit (15) is reached. The Game Master then evaluates the task success by comparing the system's outputs with the ground truth goal specification.

clem:todd draws inspiration from "self-play" and adapts it to goal-driven task-oriented dialogue systems, where the user and system take on distinct roles. Building on prior works (Zhu et al., 2020; Xu et al., 2024; Kazi et al., 2024), clem:todd employs a schema-constrained interaction loop and supports plug-and-play integration of user simulators and dialogue systems across different architectures, enabling more robust and extensible benchmarking.

3.2 User Simulator

We leverage the generative capabilities of LLMs to simulate realistic user interactions. The user simulator is prompted (see Figure 6 in Appendix A.1) with a set of goals, represented as natural language objectives, and is intended to engage in a dialogue with the system under evaluation to fulfill them. The prompt includes relevant task context and dialogue history, enabling the user simulator to produce coherent, goal-oriented responses in the conversation. Such simulators facilitate automated evaluation workflows, complementing real users.

3.3 Dialogue System Variants

As shown in Figure 1, clem:todd enables us to plug in various dialogue systems. We benchmark two existing systems and introduce three variants (Figure 2), each representing a different architectural paradigm and control mechanism.

3.3.1 Existing Dialogue Systems

We integrated two representative systems from prior work: AutoTOD, a zero-shot monolithic system that reported results on MultiWOZ 2.0 dataset (Xu et al., 2024), and a modular pipeline (Hudecek and Dusek, 2023) that reported results on MultiWOZ 2.2 dataset. These systems were selected for their architectural diversity and publicly available implementations. Attempts to incorporate other approaches were not successful [3] (see Appendix A.2 for details).

3.3.2 Proposed Dialogue Systems

Monolithic Dialogue System This variant uses a single instruction-tuned LLM (see Figure 7 for the prompt) to act as a the dialogue system, with backend actions performed via "tool use"; i.e., prediction of API calls. Although similar in style to the zero-shot approach proposed by Xu et al. (2024), our implementation differs in that their method uses a LangChain-powered agent and generates raw SQL queries directly from the LLM, whereas ours avoids third-party agents and instead generates structured tool calls conforming to the Tool Schema as described in Section 3.1.

Modular Dialogue System This design decomposes the dialogue pipeline processing into sub-modules for intent detection, slot extraction, dialogue management, and response generation. (see Figures 10, 11, 12 in the Appendix for prompt templates). We explore two variants that reflect distinct control strategies: (a) *Modular-Prog*, where a programmatic dialogue manager executes modules in a fixed order (a static pipeline); and (b) *Modular-LLM*, where an LLM acts as the dialogue manager, (see Figure 8 for the prompt), dynamically selecting the next sub-module based on intermediate outputs (an adaptive pipeline). All sub-modules use LLMs and produce outputs aligned with the Tool Schema. Outputs are validated before being passed downstream. Both variants follow a simple, traditional modular design, allowing us to assess how classical pipelines perform when augmented with LLMs.

[3] InstructTOD, evaluted on Multiwoz 2.1 (Chung et al., 2023) encountered compatibility issues between the agent implementation and LLM outputs, which led to repeated execution errors. For ProTOD, evaluted on Multiwoz 2.0 (Dong et al., 2025), the publicly released code lacked essential implementation components, rendering integration into our framework infeasible.

4 Experimental Setup

Dataset We conduct experiments using the widely used *MultiWOZ 2.2* dataset (Budzianowski et al., 2018; Hung et al., 2022), which contains annotated dialogues across multiple domains such as restaurant, hotel, and train booking. Importantly for our purposes, each dialogue comes with a verbally specified goal that was given to the 'customer' (e.g., "you want to book a Chinese restaurant in the south of the city, for a party of 4"), and a structured representation of the dialogue outcome. For evaluation, we use only the test split (1000 tasks).

To ensure a focus on end-to-end task completion, we filter the dataset to include only tasks that end in booking actions, resulting in 117 tasks (60 single-domain and 57 multi-domain dialogues) consisting of three domains (restaurant, hotel, and train). These filtered dialogues goals, after pre-processing to remove HTML artifacts, are directly used as input in the user simulator prompt. This filtering is specific to our evaluation goals and does not reflect a limitation of the `clem:todd` framework, which supports arbitrary task types and domains (more details in Section 5.3).

Model Selection We experiment with both open-weight and closed-weight LLMs to compare their effectiveness in task-oriented dialogue settings. The open-weight models include *Llama 3.1–8B*, *Llama 3.2–1B*, *Llama-3.2-3B*, *Llama 3.3–70B* from the Llama family (Grattafiori et al., 2024), as well as Qwen2.5–7B, 32B (Qwen et al., 2025) allowing us to examine performance differences across model sizes and families. For closed-weight models, we evaluate GPT-4o (version *gpt-4o-2024-08-06*). Additionally, we conducted preliminary tests with DeepSeek, but it was excluded from final experiments due to persistent issues with inconsistent JSON output formatting.

This model selection enables us to explore trade-offs between model accessibility, computational cost, and task success in dialogue system deployment. All models were run with a fixed *temperature* of 0 and a *max_new_tokens* limit of 500. GPT-4o was accessed via the OpenAI API, while open-weight models were loaded and executed locally on A100 GPUs.

Evaluation Metrics Our evaluation considers both traditional (Budzianowski et al., 2018) and goal-oriented metrics for measuring the efficiency of the dialogue system. We use *Inform*, which mea-

sures whether the system provides the correct entity matching the user's request; and *Booking Accuracy*, a metric recommended by (Xu et al., 2024), which evaluates whether the final task such as making a reservation was successfully completed. These metrics are computed automatically by comparing system outputs against the ground-truth annotations in the MultiWOZ test set. Since we filtered tasks to include only those requiring booking actions in the restaurant, hotel, and train domains tasks, the resulting set excludes entity attributes such as phone number or address, which appear only in the attraction and taxi domains. Therefore, we do not report the *Success* metric in our evaluation.

In addition to task-specific metrics, we also evaluate the overall quality of dialogue. Inspired by Kazi et al. (2024), we report dialogue-level metrics such as *naturalness*, *coherence*, and *diversity*. Furthermore, to assess the realism of the user simulator's utterances, we conduct a "Turing test" with human evaluators to determine their naturalness (see Appendix A.4 for details).

5 Results

We organize our results around the research questions introduced in Section 1, providing quantitative comparisons and visualizations.

5.1 Extending Self-Play to Task-Oriented Dialogue

Can existing LLM-based self-play frameworks be extended to evaluate interactive systems built using LLMs, such as TOD systems? We explore this question by building on the self-play framework, clembench (Chalamalasetti et al., 2023), to a TOD setting in a framework that we call `clem:todd`. It retains the core structure of the existing framework: a two-agent interaction loop, a central controller for turn coordination, logging and scoring mechanisms for evaluation. In addition, we introduce: (i) interfacing with external APIs or simulated databases, and (ii) computing task-specific metrics such as *slot coverage*, *dialogue efficiency*, and *task success*.

To validate its viability, we instantiated `clem:todd` with different pairings of user simulators (US) and dialogue systems (DS) starting with an examination of US effectiveness and its impact on dialogue evaluation. To address this overarching question, we first attend to the sub-questions.

Model (US)	Naturalness	Coherence	Diversity	TT
`Llama-3.2-1B`	3.84	2.84	1.00	-
`Llama-3.2-3B`	2.42	1.75	1.00	-
`Qwen2.5-7B`	2.75	2.42	1.05	-
`Llama-3.1-8B`	2.82	2.18	1.02	-
`Qwen2.5-32B`	**4.65**	2.95	1.00	**0.38**
`Llama-3.3-70B`	4.42	**2.97**	1.00	0.16

Table 1: Dialogue quality comparison of user simulators for the Monolithic architecture-based dialogue system (using the model: `Qwen2.5-32B`), evaluated on Naturalness (N), Coherence (C), and Dialogue Diversity (D) metrics using `GPT-4o`. The Turing Test (TT) metric measures the percentage of dialogues from a random sample of 50 that are judged as human-like, based on human evaluation. Higher scores indicate better performance.

5.1.1 Analyzing the Role of User Simulators

Which LLMs produce the most effective and realistic user simulation behavior in terms of dialogue naturalness and coherence? This question is central to reliable evaluation (Pietquin and Hastie, 2013; Davidson et al., 2023; Sekulic et al., 2024) in TOD, where systems rely on multi-turn interactions to achieve user-defined goals.

To identify user simulators that show coherent and natural behavior, we evaluate dialogue systems of varying capacities (32B and 70B) paired with simulators from the Qwen and LLaMA model families, (1B $\sim$ 70B) on single-domain test dialogues.

We assess dialogue quality using both automatic and human evaluations. For automatic evaluation, we adopt the LLM-as-a-judge framework (Kazi et al., 2024), `using GPT-4o` in a zero-shot setting to rate generated user utterances. Results in Table 1 indicate that both *naturalness* and *coherence* generally improve with increasing model size, while *dialogue-level diversity* remains relatively constant ($\sim$ 1.0). These trends are corroborated by parallel evaluations using the open-weight `Llama-3.3-70B` model as the judge (see Table 12).

To further validate these findings, we conducted a "Turing Test" (see Appendix A.4) where an annotator chose the more natural dialogue between an LLM output and its corresponding ground-truth. This evaluation assesses the dialogues produced by `Qwen2.5-32B` and `LLaMA-3.3-70B` relative to ground-truth dialogues for the same task, focusing specifically on perceived naturalness.

We developed a simple user interface (see Figure 5 in Appendix) that displays two dialogues side by side, allowing annotators to select which dia-

logue appears more natural. In each comparison, one dialogue is generated by either `Qwen2.5-32B` or `LLaMA-3.3-70B`, and the other is the corresponding ground-truth dialogue from the corpus (Budzianowski et al., 2018).

As shown in Table 1, 19 out of 50 randomly sampled dialogues generated by `Qwen2.5-32B` were preferred (0.38) over the ground truth, compared to 8 (0.16) from `LLaMA-3.3-70B`. This result highlights `Qwen2.5-32B`'s strength in generating realistic user behavior.

5.1.2 Robustness of Dialogue Systems to User Simulator Variability

How robust are dialogue systems when interacting with user simulators of varying capability and coherence? Although dialogue quality assessment (in Section 5.1.1) provides insight into simulated dialogues, it is also important to evaluate the robustness of dialogue systems to user simulators for reliable evaluation. As shown in Table 2, the overall performance of dialogue systems generally is sensitive to the choice (and with this, as established by Table 1, the quality) of the user simulator, but to differing extents: The dialogue systems realised with `Qwen2.5-32B` appear to be more capable of maintaining task success despite differences in simulator behavior. These systems can recover from incoherent user turns and use clarifying strategies to manage imperfect inputs and maintain goal alignment. In contrast, task success declines for `LLaMA-3.3-70B` when the user simulator is too small or behaves inconsistently.

To quantify this effect, we introduce **US-spread**, a robustness metric defined as the range (maximum − minimum) of task success rates achieved by a dialogue system when evaluated across different user simulator models. A lower US-spread indicates greater robustness against varying user behaviour. We observe that a monolithic dialogue system based on `Qwen2.5-32B` achieves a spread of 0.58 (see Table 2), while `LLaMA-3.3-70B` has a slightly higher spread of 0.62. Based on these findings, we recommend testing against a variety of user simulator models and using US-spread to assess robustness, during development.

While `Llama-3.3-70B` achieves the highest task success rate overall as a user simulator, `Qwen2.5-32B` closely matches its performance across both automatic and human evaluations, demonstrating strong task success, high naturalness, and competitive coherence. Given its significantly smaller model size, we select `Qwen2.5-32B` as the user simulator for all subsequent experiments, balancing performance with computational efficiency.

Model (US)	Qwen2.5-32B (DS)			Llama-3.3-70B (DS)		
	M	MP	ML	M	MP	ML
`Llama-3.2-1B`	0.42	0.34	0.32	0.28	0.27	0.18
`Llama-3.2-3B`	0.75	0.55	0.85	0.62	0.40	0.62
`Qwen2.5-7B`	0.47	0.42	0.35	0.45	0.23	0.17
`Llama-3.1-8B`	0.77	0.55	0.80	0.67	0.32	0.62
`Qwen2.5-32B`	0.95	0.58	0.82	0.83	0.53	0.70
`Llama-3.3-70B`	**1.00**	**0.80**	**0.93**	0.90	0.53	0.80
`User Spread`	0.58	**0.46**	0.61	0.62	0.25	0.62

Table 2: Task success (booking) accuracy across different LLM pairings as user simulators (US) and dialogue systems (DS) in the `clem:todd` framework. Rows indicate the models used as US, and columns show the DS models evaluated in three architectural variants: Monolithic (M), Modular-Programmatic (MP), and Modular-LLM (ML). The bottom-most row (User Spread) reports the standard deviation across user simulators for each DS configuration, reflecting its sensitivity to variation in simulator behavior. Lower User Spread values indicate greater robustness, but should be considered alongside overall task success.

5.1.3 Evaluating Dialogue Systems

With the user simulator model fixed, we can now systematically evaluate dialogue system realisation strategies, and the model realising them. *How do different dialogue system configurations, varying in model size, prompting strategy, and architectural design, compare in terms of task success?*

The `clem:todd` framework streamlines experimentation across models and configurations. We assess systems using a wide range of LLMs—spanning 1B to 70B parameters and model families such as Qwen, LLaMA, and GPT. Our objective is to identify how architectural choices impact performance, and to what extent model size influences outcomes.

As mentioned in Section 3, we evaluated three different dialogue systems and the results are reported in Table 3 based on standard metrics for task-oriented dialogue (Budzianowski et al., 2018): *Inform* (I), and *Booking Accuracy* (B) (Xu et al., 2024), with *Booking Accuracy* used as the primary indicator of end-to-end task success.

At smaller scales (1B–3B), all architectures perform poorly (Booking Rate ~ 0.05), highlighting the limitations of small models for complex dia-

Model	Monolithic		Modular-Prog		Modular-LLM		Xu et al. (2024)		Hudecek and Dusek (2023)	
	Inform	Booking	Inform	Booking	Inform	Booking	Inform	Booking	Inform	Booking
Llama-3.2-1B	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.20	-
Llama-3.2-3B	0.05	0.05	0.0	0.0	0.02	0.02	0.0	0.0	0.20	-
Qwen2.5-7B	0.09	0.09	0.28	0.26	0.25	0.24	0.32	0.30	0.25	-
Llama-3.1-8B	0.20	0.18	0.0	0.0	0.08	0.06	0.12	0.09	0.28	-
Qwen2.5-32B	0.76	0.71	0.44	0.41	0.71	0.68	0.63	0.60	0.52	-
Llama-3.3-70B	0.70	0.69	0.43	0.43	0.53	0.52	0.12	0.12	0.32	-
GPT-4o	**0.82**	**0.81**	**0.65**	**0.65**	**0.85**	**0.84**	**0.75**	**0.73**	0.42	-

Table 3: Performance of various dialogue system architectures within the clem:todd framework. The table presents inform(I), and booking accuracy (B) for Monolithic, Modular-Prog, and Modular-LLM architectures across different LLMs. Systems based on Xu et al. (2024) and Hudecek and Dusek (2023) are also evaluated within the clem:todd setup. Larger models consistently yield better performance. Hudecek and Dusek (2023)'s system does not support booking accuracy computation.

logue. We observe that a majority of conversations were aborted prematurely due to models failing to adhere to the expected response format constraints, leading to incomplete task execution and significantly lower task success rates.

At the other end of the scale, GPT-4o achieves higher booking rates in modular configurations, reaching ~ 0.84. In monolithic setups, most failures occur in *train domain* dialogues involving *leave after* or *arrival by* constraints, where the model often produces incorrect output formats (see Appendix A.2.2), leading to database failures and mismatches with ground truth goals. In contrast, the structured decomposition in modular systems helps the model reliably follow schema-specific outputs, improving constraint handling and task success.

In contrast, open-weight models perform better in monolithic configurations, achieving booking success rates around ~ 0.77, but show significantly reduced performance in modular setups. Most failures stem from invalid response formats or prematurely signalling the end of conversation without conversing about all the goals (see Appendix A.2.2). In modular configurations, the overall task is decomposed into subtasks, and individual modules operate without access to the full dialogue history. This limited context appears to hinder the model's ability to consistently satisfy goal constraints, reducing task success. In monolithic setups, a detailed analysis shows that these models can reach booking rates as high as 0.95 (Qwen2.5-32B) for single-domain dialogues (see Table 13 for the breakdown of results across domains). However, their performance drops in multi-domain scenarios due to increased complexity and cross-domain goal tracking.

Model	Monolithic		Modular-Prog		Modular-LLM	
	T($)	F($)	T($)	F($)	T($)	F($)
Llama-3.2-1B	0.009	0.007	0.009	0.007	0.0008	0.007
Llama-3.2-3B	0.007	0.466	0.007	0.362	0.002	0.103
Qwen2.5-7B	0.096	0.091	0.010	0.374	0.004	0.137
Llama-3.1-8B	0.219	0.442	0.011	0.667	0.003	0.298
Qwen2.5-32B	0.007	0.173	0.059	0.487	0.009	0.258
Llama-3.3-70B	0.218	0.502	0.020	1.333	0.007	1.358
GPT-4o	0.019	NA	2.113	NA	0.031	NA

Table 4: Cost comparison of dialogue system architectures within the clem:todd framework. The table reports token-based cost (T) and FLOPS cost (F), both measured in USD ($) per dialogue, for Monolithic, Modular-Prog, and Modular-LLM architectures across different LLMs.

5.2 Evaluating Computational Efficiency of Dialogue Systems

What are the computational cost patterns across dialogue system architectures and model scales? We evaluated both token and FLOPs-based costs across all the three configurations using a standardized estimation approach (Kaplan et al., 2020; Chowdhery et al., 2023). More details on the cost computation are available in Appendix A.6

It is important to note that token costs are not solely determined by the number of tokens used. They also depend on the pricing set by API providers for each specific model. For example, some providers offer significantly lower input/output token prices for certain models, which can reduce the apparent cost even if token usage is high. Therefore, comparisons of token cost across architectures should be interpreted with this pricing variability in mind. While our cost estimates use OpenRouter pricing at the time of writing as

Model	Monolithic	Modular-Prog	Modular-LLM
Llama-3.2-1B	18.85	1.61	18.72
Llama-3.2-3B	5.21	8.32	16.56
Qwen2.5-7B	6.69	11.38	13.70
Llama-3.1-8B	5.97	11.16	42.08
Qwen2.5-32B	8.52	10.16	7.17
Llama-3.3-70B	25.33	34.07	47.98
GPT-4o	4.10	8.61	4.37

Table 5: Latency time (in seconds) comparison across dialogue system architectures in the clem:todd framework. The table reports end-to-end response time for Monolithic, Modular-Prog, and Modular-LLM architectures evaluated across different LLMs.

a consistent baseline,[4] these values may differ in production settings depending on the provider and deployment context.

As shown in Table 4, the Monolithic architecture has the lowest computational cost due to its single-pass execution, avoiding repeated LLM calls. In contrast, Modular Prog and Modular LLM incur higher costs. Modular Prog has the highest FLOP cost, up to $1.33 with Llama-3.3-70B, $\sim$ 2 times more than its Monolithic counterpart, reflecting poorly on efficiency.

Additionally, we computed the latency time for each system, defined as the total time elapsed from the first interaction initiated by the user simulator until the task completion or the maximum number of dialogue turns (15) is reached. As shown in Table 5, Monolithic systems exhibit the lowest latency times across most configurations. This is expected, as Monolithic systems involve a single model call per turn, whereas the Modular-Prog and Modular-LLM systems require multiple calls to sub-modules, increasing overall latency. The highest latency is observed for Llama-3.3-70B, which was executed on two GPUs, suggesting that model size and hardware configuration contribute significantly to response time. Among the modular variants, Modular-LLM systems generally exhibit higher latency than Modular-Prog systems, which can be attributed to the overhead of using an LLM as the dialogue manager in addition to the other modules. In contrast, the lowest latency is observed for Llama-3.2-1B, which is primarily due to early termination of many dialogues resulting from format violations leading to fewer turns and thus shorter overall latency.

These results underscore a key trade-off: modular architectures enhance interpretability and task decomposition but at a significant computational and token cost, especially at scale.

[4] https://openrouter.ai

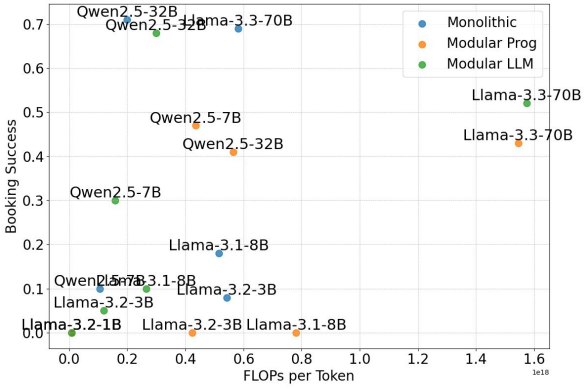

Figure 3: Trade-off between computational cost and booking success across dialogue architectures. The x-axis shows the FLOPs per token (computational cost), and the y-axis shows booking accuracy (task performance). Monolithic and Modular-LLM architectures tend to achieve higher booking accuracy at lower or comparable cost compared to Modular-Prog.

Further more, we want to analyse the trade-offs (see Figure 3) between model scale and computational cost (e.g., API usage, FLOPs). Larger models like Llama-3.3-70B and Qwen2.5-32B achieve markedly higher task performance in both monolithic and modular settings (e.g., Qwen2.5-32B Monolithic: Booking Rate = 0.71). Among variants, Modular-LLM strikes a favorable balance, delivering comparable performance (e.g., Qwen2.5-32B: 0.68 vs. 0.41 in Mod-P) while reducing FLOP cost ($0.26 vs. $0.49). However, modular systems incur higher inference overhead due to multiple LLM calls per turn. This underscores a key trade-off: **monolithic systems offer inference efficiency but limited flexibility, whereas modular systems support extensibility and interpretability at greater computational cost**.

Overall, architecture selection should align with application-specific priorities. Monolithic setups are preferable when minimizing compute or API usage is critical. Modular architectures, particularly Modular-LLM, offer a compelling compromise.

5.3 Evaluating the Adaptability of clem:todd to New Domains

Can clem:todd *be adapted to new domains and data configurations?* clem:todd retains the design philosophy of the underlying "self-play" frame-

Model	Multiwoz-Style			Unrealistic		
	I	S	B	I	S	B
Llama-3.2-1B	0.00	0.00	0.00	0.00	0.00	0.00
Llama-3.2-3B	0.18	0.08	0.18	0.03	0.02	0.02
Qwen2.5-7B	0.13	0.03	0.13	0.08	0.03	0.03
Llama-3.1-8B	0.26	0.08	0.26	0.08	0.03	0.07
Qwen2.5-32B	0.64	0.39	**0.64**	0.41	0.28	0.18
Llama-3.3-70B	0.54	0.24	0.54	0.20	0.11	0.07
GPT-4o	0.68	0.38	**0.64**	0.60	0.35	**0.33**

Table 6: Performance of monolithic dialogue systems evaluated for inform (I), success (S) and booking accuracy (B) with Qwen2.5-32B as the user simulator on the MultiWOZ-style synthetic and unrealistic datasets.

work by separating the evaluation setup from the definition of the test instances. This allows evaluation with new instances (here, new goals), without any changes to the underlying logic. To assess this adaptability, we created two new synthetic datasets (see Appendix A.7).

The first follows the MultiWOZ style but features unseen goal combinations that do not appear in the original dataset, while preserving task structure (see Appendix A.7). The second contains intentionally unrealistic scenarios, such as *booking a dragon for travel to London*, to test how systems handle inputs that are likely to differ from any data encountered during training.

We evaluated our proposed dialogue systems using the new datasets, while keeping the experimental configuration consistent with earlier evaluations. `Qwen2.5-32B` is used as the user simulator and we varied the dialogue system model. Table 6 shows that models exhibit closer task success for the MultiWOZ-style data but struggled with unrealistic tasks.

When the user simulator poses uncommon requests, such as booking a table for -2 people or *reserving a stay in a dungeon in the middle of the ocean*, the dialogue systems often attempt to correct. For instance, it flags the negative party (-2) size as a likely typo, which the user simulator adjusts, resulting in a deviation from the original goal. Similarly, when handling unusual accommodation requests, the dialogue systems steering the user to consider more conventional hotel options. These interactions, while human-like, differ from the ground truth annotations, which leads to lower task success scores. Among all models evaluated, `GPT-4o` and `Qwen2.5-32B` able to accommodate such unconventional requests to some extent.

Together, these results indicate that `clem:todd` enables controlled evaluation on new data distributions and supports robust testing beyond fixed benchmarks. It provides a platform for analyzing how dialogue systems handle unseen goals, detect potential overfitting, and generalize to tasks outside their training exposure.

6 Conclusion

We propose `clem:todd` a framework designed for the systematic evaluation of LLM-based task-oriented dialogue systems. By leveraging a structured self-play paradigm, `clem:todd` enables consistent and modular assessment across diverse dialogue system architectures and user simulators. Our experiments on standard (MultiWOZ) benchmarks, demonstrate that while large-scale monolithic models offer better performance at low computational cost, modular architectures, especially Modular-LLM variants, achieve reasonable trade-offs between performance and efficiency. Furthermore, we show that `clem:todd` can be adapted to evaluate dialogue systems in unseen and unrealistic domains beyond fixed datasets.

Limitations

Although `clem:todd` offers flexibility and adaptability, it has limitations that call for future research. First, the design choice of using a simple, classical pipeline in our proposed modular dialogue systems was intended to assess LLM performance in a controlled setting. However, the relatively lower scores (for modular-program variant) may not reflect model limitations alone, but also limitations of the pipeline design itself, which may be too constrained to capture more complex dialogue behaviors. Second, our proposed setup assumes strict adherence to response formats (as defined by the Tool Schema), which smaller models frequently violate. This leads to premature termination of dialogues and may underestimate the true capabilities of such models. Third, our current evaluation includes only GPT-4o among closed-weight models, limiting broader comparison with other proprietary LLMs (Claude, Gemini etc.). Fourth, while the current self-play setup is designed to be extensible, it is limited to two-player interactions. Extending the framework to support multi-agent dialogue scenarios is an open direction for future exploration.

Acknowledgments

The work reported here has been funded by the Bundesministerium für Bildung und Forschung (BMBF, German Federal Ministry of Research), project "COCOBOTS" (01IS21102A) and Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant 423217434 ("RECOLAGE"). We thank the anonymous reviewers for their helpful feedback.

References

Anusha Balakrishnan, Jinfeng Rao, Kartikeya Upasani, Michael White, and Rajen Subba. 2019. Constrained decoding for neural NLG from compositional representations in task-oriented dialogue. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 831–844. Association for Computational Linguistics.

Anna Bavaresco, Raffaella Bernardi, Leonardo Bertolazzi, Desmond Elliott, Raquel Fernández, Albert Gatt, Esam Ghaleb, Mario Giulianelli, Michael Hanna, Alexander Koller, André F. T. Martins, Philipp Mondorf, Vera Neplenbroek, Sandro Pezzelle, Barbara Plank, David Schlangen, Alessandro Suglia, Aditya K Surikuchi, Ece Takmaz, and Alberto Testoni. 2024. Llms instead of human judges? a large scale empirical study across 20 nlp evaluation tasks. *Preprint*, arXiv:2406.18403.

Anouck Braggaar, Christine Liebrecht, Emiel van Miltenburg, and Emiel J. Krahmer. 2023. Evaluating task-oriented dialogue systems: A systematic review of measures, constructs and their operationalisations. *CoRR*, abs/2312.13871.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Pawel Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gasic. 2018. Multiwoz - A large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 5016–5026. Association for Computational Linguistics.

Jacky Casas, Marc-Olivier Tricot, Omar Abou Khaled, Elena Mugellini, and Philippe Cudré-Mauroux. 2020. Trends & methods in chatbot evaluation. In *Companion Publication of the 2020 International Conference on Multimodal Interaction, ICMI Companion 2020, Virtual Event, The Netherlands, October, 2020*, pages 280–286. ACM.

Kranti Chalamalasetti, Jana Götze, Sherzod Hakimov, Brielen Madureira, Philipp Sadler, and David Schlangen. 2023. clembench: Using game play to evaluate chat-optimized language models as conversational agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 11174–11219. Association for Computational Linguistics.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards better llm-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Xiuyi Chen, Jiaming Xu, and Bo Xu. 2019. A working memory model for task-oriented dialog response generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2687–2693, Florence, Italy. Association for Computational Linguistics.

Pengyu Cheng, Tianhao Hu, Han Xu, Zhisong Zhang, Yong Dai, Lei Han, Nan Du, and Xiaolong Li. 2024. Self-playing adversarial language game enhances LLM reasoning. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Qinyuan Cheng, Linyang Li, Guofeng Quan, Feng Gao, Xiaofeng Mou, and Xipeng Qiu. 2022. Is MultiWOZ a solved task? an interactive TOD evaluation framework with user simulator. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1248–1259, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, and 48 others. 2023. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113.

Willy Chung, Samuel Cahyawijaya, Bryan Wilie, Holy Lovenia, and Pascale Fung. 2023. InstructTODS: Large language models for end-to-end task-oriented dialogue systems. In *Proceedings of the Second Workshop on Natural Language Interfaces*, pages 1–21. Association for Computational Linguistics.

Sam Davidson, Salvatore Romeo, Raphael Shu, James Gung, Arshit Gupta, Saab Mansour, and Yi Zhang. 2023. User simulation with large language models for evaluating task-oriented dialogue. *CoRR*, abs/2309.13233.

Wenjie Dong, Sirong Chen, and Yan Yang. 2025. Protod: Proactive task-oriented dialogue system based on large language model. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 9147–9164. Association for Computational Linguistics.

Henry Elder, Alexander O'Connor, and Jennifer Foster. 2020. How to make neural natural language generation as reliable as templates in task-oriented dialogue. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2877–2888, Online. Association for Computational Linguistics.

Mihail Eric, Lakshmi Krishnan, Francois Charette, and Christopher D. Manning. 2017. Key-value retrieval networks for task-oriented dialogue. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 37–49, Saarbrücken, Germany. Association for Computational Linguistics.

Asma Ghandeharioun, Judy Hanwen Shen, Natasha Jaques, Craig Ferguson, Noah Jones, Agata Lapedriza, and Rosalind Picard. 2019. *Approximating interactive human evaluation with self-play for open-domain dialog systems*. Curran Associates Inc., Red Hook, NY, USA.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, and et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. 2025. Textarena. *arXiv preprint arXiv:2504.11442*.

Aman Gupta, Anirudh Ravichandran, Ziji Zhang, Swair Shah, Anurag Beniwal, and Narayanan Sadagopan. 2024. DARD: A multi-agent approach for task-oriented dialog systems. *CoRR*, abs/2411.00427.

Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. 2024. Llm-rubric: A multidimensional, calibrated approach to automated evaluation of natural language texts. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 13806–13834. Association for Computational Linguistics.

Vojtech Hudecek and Ondrej Dusek. 2023. Are large language models all you need for task-oriented dialogue? In *Proceedings of the 24th Meeting of the Special Interest Group on Discourse and Dialogue, SIGDIAL 2023, Prague, Czechia, September 11 - 15, 2023*, pages 216–228. Association for Computational Linguistics.

Chia-Chien Hung, Anne Lauscher, Ivan Vulic, Simone Paolo Ponzetto, and Goran Glavas. 2022. Multi2woz: A robust multilingual dataset and conversational pretraining for task-oriented dialog. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 3687–3703. Association for Computational Linguistics.

Jinghan Jia, Abi Komma, Timothy Leffel, Xujun Peng, Ajay Nagesh, Tamer Soliman, Aram Galstyan, and Anoop Kumar. 2024. Leveraging llms for dialogue quality measurement. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track, NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 359–367. Association for Computational Linguistics.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *CoRR*, abs/2001.08361.

Taaha Kazi, Ruiliang Lyu, Sizhe Zhou, Dilek Hakkani-Tür, and Gokhan Tur. 2024. Large language models as user-agents for evaluating task-oriented-dialogue systems. In *IEEE Spoken Language Technology Workshop, SLT 2024, Macao, December 2-5, 2024*, pages 913–920. IEEE.

Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. 2024. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 4334–4353. Association for Computational Linguistics.

Takyoung Kim, Jamin Shin, Young-Ho Kim, Sanghwan Bae, and Sungdong Kim. 2023. Revealing user familiarity bias in task-oriented dialogue via interactive evaluation. *CoRR*, abs/2305.13857.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Cassidy Laidlaw, Eli Bronstein, Timothy Guo, Dylan Feng, Lukas Berglund, Justin Svegliato, Stuart Russell, and Anca Dragan. 2024. Scalably solving assistance games. In *ICML 2024 Workshop on Models of Human Feedback for AI Alignment*.

Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. 2017. End-to-end task-completion neural dialogue systems. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing, IJCNLP 2017, Taipei, Taiwan, November 27 - December 1, 2017 - Volume 1: Long Papers*, pages 733–743. Asian Federation of Natural Language Processing.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023a. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9):195:1–195:35.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023b. G-eval: NLG evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 2511–2522. Association for Computational Linguistics.

Michael F McTear. 2002. Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Surveys*, 34(1):90–169. Publisher: School of Information and Software Engineering, University of Ulster.

Olivier Pietquin and Helen F. Hastie. 2013. A survey on metrics for the evaluation of user simulations. *Knowl. Eng. Rev.*, 28(1):59–73.

Dan Qiao, Chenfei Wu, Yaobo Liang, Juntao Li, and Nan Duan. 2023. Gameeval: Evaluating llms on conversational games. *CoRR*, abs/2308.10032.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, and et al. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Ivan Sekulic, Silvia Terragni, Victor Guimarães, Nghia Khau, Bruna Guedes, Modestas Filipavicius, André Ferreira Manso, and Roland Mathis. 2024. Reliable llm-based user simulator for task-oriented dialogue systems. *CoRR*, abs/2402.13374.

Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, Sankaran Vaidyanathan, and Dieuwke Hupkes. 2024. Judging the judges: Evaluating alignment and vulnerabilities in llms-as-judges. *CoRR*, abs/2406.12624.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Heng-Da Xu, Xian-Ling Mao, Puhai Yang, Fanshu Sun, and Heyan Huang. 2024. Rethinking task-oriented dialogue systems: From complex modularity to zero-shot autonomous agent. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 2748–2763. Association for Computational Linguistics.

Yi-Ting Yeh, Maxine Eskénazi, and Shikib Mehri. 2021. A comprehensive assessment of dialog evaluation metrics. *CoRR*, abs/2106.03706.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Qi Zhu, Zheng Zhang, Yan Fang, Xiang Li, Ryuichi Takanobu, Jinchao Li, Baolin Peng, Jianfeng Gao, Xiaoyan Zhu, and Minlie Huang. 2020. Convlab-2: An open-source toolkit for building, evaluating, and diagnosing dialogue systems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, ACL 2020, Online, July 5-10, 2020*, pages 142–149. Association for Computational Linguistics.

A Appendix

A.1 Prompt Templates

In the proposed tasks, LLMs are used in a variety of roles across different dialogue system architectures. Specifically, LLMs serve as user simulator (see Figure 6), as a complete dialogue system (in the monolithic architecture) (see Figure 7), as a dialogue manager (in modular-LLM architecture) (see Figure 8), and as a individual dialogue modules such as intent detector (see Figure 10), slot extractor (see Figure 11), and response generator (see Figure 12) in both modular-LLM and modular-prog architectures.

Following standard prompting approaches (Brown et al., 2020; Wei et al., 2022; Liu et al., 2023a), all prompts were constructed in a zero-shot setting without the use of explicit examples. Prompts were formatted using a system-message (emphasizing the required behavior) followed by a user-message structure.

While the specific content of the prompts varies depending on the role and task (e.g., simulating user intents, extracting slots, managing dialogue state transitions), the overall prompt structure remains consistent across all scenarios. Each prompt is composed of the following components:

1. System Message: Defines the role that the LLM is expected to play (e.g., user simulator, dialogue manager, slot extractor).

2. Task Description: Provides a brief overview of the task context and the objective the LLM should achieve.

3. Instructions and Rules: Specifies detailed guidelines and constraints that the LLM must follow when generating responses.

4. Test Input: The specific input query for which a response is to be generated.

This prompt format ensures that models are contextualized for their assigned tasks while allowing flexibility in content to accommodate the differing requirements of user simulation, dialogue management, and modular dialogue tasks.

A.2 Existing Systems Integration

As part of our benchmarking, we integrated two representative task-oriented dialogue systems from prior work into the `clem:todd` framework: AutoTOD (Xu et al., 2024) and the modular pipeline proposed by Hudecek and Dusek (2023).

A.2.1 AutoTOD Integration

AutoTOD is a zero-shot, monolithic system, using a LangChain-powered agent. In our integration, we maintained AutoTOD's internal logic for database operations and booking reference number generation to maintain consistency with its original design. However, we ensured that it operated over the same MultiWOZ 2.2 database used for all other dialogue systems in our framework, thus preserving fairness in task execution and evaluation.

AutoTOD's original implementation [5] uses an older version of LangChain (v0.0.166), which required adjustments to ensure compatibility with current LangChain APIs. The framework includes two agent types: a ReAct agent and a function agent. We initially attempted to use the ReAct agent, but integration failed due to format mismatches between the LLM-generated outputs and the expected tool call structure—an issue stemming from changes introduced in newer LangChain versions. As a result, we integrated the function agent, which was working under the upgraded environment.

The function agent, however, expects LLMs to generate raw SQL queries. This introduced some issues, as the case of generated column names occasionally did not match the schema of the underlying database. To address this, we modified the database layer to support case-insensitive column matching. Additionally, the booking function uses regular expressions to extract timing information in train booking scenarios. These expressions were failing in some edge cases and we therefore updated the relevant regex patterns.

In Table 7, we report the AutoTOD results from the original paper. To enable fair comparison, we reproduced the system in its original setup and evaluated in two modes inspired by our `clem:todd` experiments: first, by using the same model for both the agent and the dialogue system, and second, by using a single instruction-following model `Qwen2.5-32B` as user model against other models for dialogue systems. These results are presented in Table 11.

Our analysis indicates that performance improves when the same model is used across components, as opposed to mixing models. Furthermore, when using `Qwen2.5-32B` based user simulator in our setup, the dialogue system outperformed its own setup, likely due to the improved user simulator prompts provided by `clem:todd`Ẇe also observe a performance gap between the paper-reported results and our reproduced results, which is likely due to differences in agent configuration during evaluation, the original paper might have used the ReAct agent, whereas we used the function agent due to compatibility issues with newer LangChain versions. This change in agent behavior may have affected how tool calls were handled and how robustly dialogue goals were completed.

A.2.2 Modular Pipeline System Integration

We integrated the modular pipeline dialogue system proposed by Hudecek and Dusek (2023). The system comes with its own implementation [6] for handling database interactions and generating booking reference numbers. We retained these components as is but ensured that the underlying database matched the MultiWOZ 2.2 version used across all systems in `clem:todd` to maintain consistency.

The original implementation used delexicalized responses, as their evaluation setup did not rely on realistic user simulation. Since `clem:todd` requires

[5] https://github.com/DaDaMrX/AutoTOD

[6] https://github.com/vojtsek/to-llm-bot

Model	Domain Level				Dialogue Level			
	Inform	Success	Book	Combine	Inform	Success	Book	Combine
Llama-2-13B	0.37	0.29	0.32	0.34	0.29	0.23	0.27	0.27
Llama-2-70B	0.54	0.43	0.44	0.49	0.33	0.31	0.32	0.32
GPT-3.5-Turbo	0.63	0.53	0.51	0.57	0.43	0.46	0.48	0.46
GPT-4o	**0.85**	**0.59**	**0.87**	**0.79**	**0.80**	**0.47**	**0.82**	**0.72**

Table 7: Domain-level and dialogue-level performance metrics (Inform, Success, Book, and Combine) that were reported in the baseline paper (Xu et al., 2024)

Model	Domain Level			Dialogue Level		
	Inform	Book	Combine	Inform	Book	Combine
Llama-2-13B	0.00	0.00	0.00	0.00	0.00	0.00
Llama-2-70B	0.00	0.00	0.00	0.00	0.00	0.00
Llama-3.2-1B	0.36	0.00	0.13	0.27	0.00	0.08
Llama-3.2-3B	0.24	0.00	0.09	0.09	0.00	0.03
Qwen2.5-7B	0.11	0.00	0.05	0.04	0.00	0.02
Llama-3.1-8B	0.17	0.00	0.07	0.1	0.00	0.05
Qwen2.5-32B	0.11	0.00	0.05	0.05	0.00	0.03
Llama-3.3-70B	0.11	0.00	0.06	0.06	0.00	0.03
GPT-4o	**0.73**	**0.73**	**0.59**	**0.67**	**0.67**	**0.51**

Table 8: Results of the baseline system on the filtered test set. We use the same model for both the user and agent. As the results are based on a filtered subset, performance differs from the baseline but comparable for GPT-4o.

Model	Domain Level			Dialogue Level		
	Inform	Book	Combine	Inform	Book	Combine
Llama-2-13B	0.00	0.00	0.00	0.00	0.00	0.00
Llama-2-70B	0.00	0.00	0.00	0.00	0.00	0.00
Llama-3.2-1B	0.40	0.00	0.13	0.23	0.00	0.07
Llama-3.2-3B	0.24	0.00	0.09	0.09	0.00	0.03
Qwen2.5-7B	0.11	0.00	0.05	0.04	0.00	0.02
Llama-3.1-8B	0.18	0.00	0.07	0.11	0.00	0.04
Qwen2.5-32B	0.11	0.00	0.05	0.05	0.00	0.03
Llama-3.3-70B	0.12	0.00	0.06	0.06	0.00	0.03
GPT-4o	0.16	0.00	0.07	0.10	0.00	0.04

Table 9: Baseline system results using Qwen-2.5-32B as the user simulator and varying the agent model. The performance drops significantly compared to using the same model for both user and agent, showcasing the sensitivity of the dialogue system to the user simulator.

Model	Inform	Success	BLEU
ChatGPT-Zeroshot	0.47	NA	3.76
ChatGPT-Fewshot	0.68	NA	6.84

Table 10: Performance metrics (Inform, Success, and Book) that were reported in the baseline paper (Hudecek and Dusek, 2023)

naturalistic dialogue for interaction with LLM-based user simulators, we modified the code to disable delexicalization, allowing entity names and values to appear directly in the system responses. Additionally, their regular expression logic for extracting domains from user input did not generalize well across all test cases, so we extended this pattern to improve robustness.

The original system also used the Weights & Biases (wandb) framework for logging and tracking experiments. As this was not essential for integration with clem:todd we removed these dependencies during setup. Furthermore, the original implementation did not compute booking success as part of its evaluation, so we could not report these results.

A.3 Response Format Issues

Despite explicitly defining constraints and tool schemas for response generation, models do not

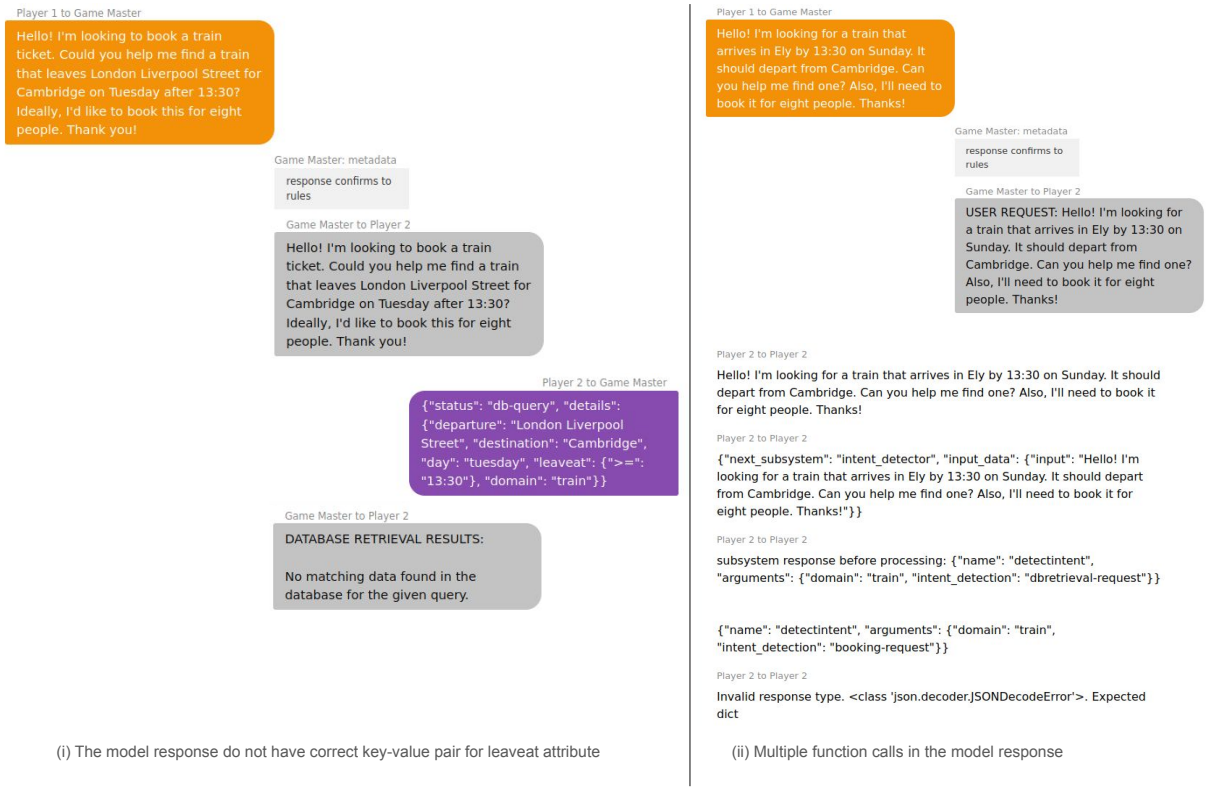

Figure 4: Examples of invalid model responses due to format violations during dialogue system evaluation.

Model	BLEU	Inform	R	H	T
Llama-2-13B	2.43	0.30	0.18	1.00	0.25
Llama-2-70B	0.58	0.30	0.18	1.00	0.25
Llama-3.2-1B	0.32	0.30	0.18	1.00	0.25
Llama-3.2-3B	0.33	0.30	0.18	1.00	0.25
Qwen2.5-7B	0.54	0.30	0.18	1.00	0.25
Llama-3.1-8B	0.49	0.30	0.18	1.00	0.25
Qwen2.5-32B	0.65	0.30	0.18	1.00	0.25
Llama-3.3-70B	0.58	0.30	0.18	1.00	0.25
GPT-4o	0.73	0.30	0.18	1.00	0.25

Table 11: Few-shot results on the Multiwoz filtered test set using the same model for both user and dialogue system. Metrics include overall Inform, and domain-specific scores for restaurant (R), hotel (H), and train (T). Performance differs from the baseline due to filtering.

always adhere strictly to the specified formats. Deviations from the expected response format often lead to invalid outputs, resulting in parsing errors or task execution failures, which ultimately reduce the overall evaluation scores.

common types of invalid responses include incorrect key-value structures, missing required fields, and multiple function calls when only a single function call was expected. Figure 4 illustrates such invalid responses observed during evaluation.

A.4 Dialogue Quality Evaluation

To assess the quality of the generated user simulator utterances, we follow the methodology outlined by Kazi et al. (2024). Specifically, we use two models in a zero-shot setting to score the utterances: a closed-source model (`GPT-4o`) and an open-weight model (`LLaMA-3.3-70B`). The evaluation focuses on three dimensions: naturalness (N), coherence (C), and dialogue-level diversity (D). Naturalness is rated on a scale from 1 to 5, while coherence and diversity are rated on a scale from 1 to 3, with higher scores indicating better performance.

To cross-validate these `GPT-4o` findings(see Table 1), we conducted a parallel evaluation using `LLaMA-3.3-70B`, and results are shown in Table 12. The open-weight model exhibits similar trends, reinforcing the observations made with `GPT-4o`.

Dialogue diversity scores remain mostly stable, typically around $\sim$ 1.0, across different models and architectures. This suggests limited variability in simulator utterances. While task-oriented dialogues are inherently more constrained than open-domain dialogues, real users often introduce a degree of natural variation (Kim et al., 2023) through paraphrasing, hesitations, and reformulations. The relatively low diversity observed here implies that

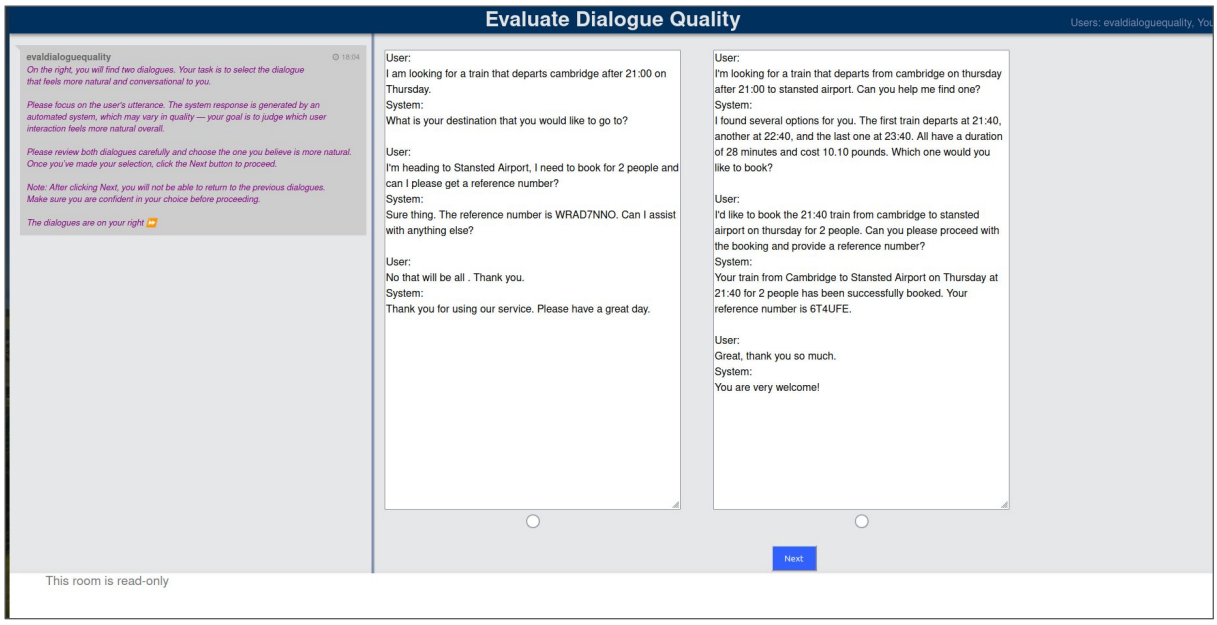

Figure 5: Web interface used for the Human Turing Test. Annotators were shown two dialogues side by side (a generated dialogue and a ground-truth dialogue) and asked to select the one that appeared more natural.

Model	Naturalness	Coherence	Dialogue Diversity
`Llama-3.2-1B`	3.90	2.90	1.00
`Llama-3.2-3B`	3.25	2.18	1.02
`Qwen2.5-7B`	2.55	2.22	1.12
`Llama-3.1-8B`	2.97	2.32	1.03
`Qwen2.5-32B`	**4.87**	**3.00**	1.00
`Llama-3.3-70B`	4.28	2.93	1.00

Table 12: Dialogue quality comparison of user simulators for the Monolithic architecture-based dialogue system (using the model: `Qwen2.5-32B`), evaluated on Naturalness (N), Coherence (C), and Dialogue Diversity (D) metrics using `Llama-3.3-70B`. Higher scores indicate better performance.

current user simulators may be overly deterministic, potentially restricting the robustness of dialogue systems trained with them. To address this, future work should aim to develop user simulators that balance fluency with controlled diversity, thereby better approximating real-world conversational behaviors.

Human Evaluation To further investigate the performance of the user simulator, we conducted a Turing Test to compare generated dialogues against ground-truth dialogues.

We recruited an annotator to perform the annotation. The annotator did not have prior experience in dialogue annotation but had general experience with data labeling tasks. Detailed instructions for the Turing Test were embedded within the web interface (see Figure 5) to guide the evaluation process. The annotation task involved evaluating 100 dialogue pairs and took approximately 150 minutes to complete.

The results are summarized in Table 1 under TT (Turing Test) metric. For each model, 50 generated dialogues were evaluated against ground-truth dialogues. Out of 50 comparisons, 19 dialogues generated by `Qwen2.5-32B` were judged to be more natural, while only 8 dialogues generated by `LLaMA-3.3-70B` were preferred.

It is important to note that the human Turing Test and the LLM-judge evaluation use fundamentally different methodologies. The human evaluation compares generated dialogues directly against ground-truth dialogues for the same task, whereas the LLM-judge evaluation scores each generated dialogue independently based on naturalness, coherence, and diversity, without referencing the ground truth. Although the results from the two evaluations are not directly comparable, both indicate that the dialogues generated by the `Qwen2.5-32B` model tend to sound more natural.

A.5 Domain-specific Results

These domain-wise results provide a finer-grained view of how model and system design choices impact task completion across different dialogue scenarios. Table 13 presents detailed booking success rates for each model across different di-

Domain	Model	Monolithic				Modular-Prog				Modular-LLM			
		R	H	T	Avg	R	H	T	Avg	R	H	T	Avg
Single	Llama-3.2-1B	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Llama-3.2-3B	0.20	0.05	0.00	0.08	0.00	0.00	0.00	0.00	0.10	0.00	0.00	0.03
	Qwen2.5-7B	0.00	0.00	0.30	0.10	0.70	0.50	0.20	0.47	0.45	0.25	0.20	0.30
	Llama-3.1-8B	0.50	0.25	0.10	0.28	0.00	0.00	0.00	0.00	0.20	0.10	0.00	0.10
	Qwen2.5-32B	1.00	0.90	0.95	**0.95**	0.80	0.50	0.45	0.58	0.95	0.75	0.75	0.82
	Llama-3.3-70B	0.85	0.65	1.00	0.83	0.80	0.35	0.45	0.53	0.80	0.50	0.80	0.70
	GPT-4o	0.95	0.70	0.50	0.72	1.00	0.80	0.75	**0.85**	1.00	0.95	0.65	**0.87**
Domain	Model	R-H	H-T	T-R	Avg	R-H	H-T	T-R	Avg	R-H	H-T	T-R	Avg
Multi	Llama-3.2-1B	0.00	0.00	0.00	0.00	0.80	0.50	0.45	0.58	0.00	0.00	0.00	0.00
	Llama-3.2-3B	0.00	0.00	0.05	0.02	0.80	0.50	0.45	0.58	0.00	0.00	0.00	0.00
	Qwen2.5-7B	0.00	0.10	0.10	0.07	0.80	0.50	0.45	0.58	0.19	0.05	0.25	0.16
	Llama-3.1-8B	0.18	0.00	0.15	0.11	0.80	0.50	0.45	0.58	0.05	0.00	0.05	0.02
	Qwen2.5-32B	0.71	0.45	0.60	0.59	0.24	0.15	0.25	0.25	0.59	0.50	0.55	0.55
	Llama-3.3-70B	0.47	0.40	0.60	0.49	0.41	0.80	0.35	0.45	0.47	0.25	0.30	0.34
	GPT-4o	0.76	0.45	0.90	**0.70**	1.00	0.80	0.75	**0.56**	0.94	0.70	0.80	**0.81**

Table 13: Booking rates for each model across different dialogue system architectures (Monolithic, Modular-Prog, Modular-LLM) and domains. "Single" refers to tasks within a single domain (Restaurant(R), Hotel(H), or Train(T)), while "Multi" refers to tasks spanning multiple domains. Higher values indicate better task completion performance.

alogue system architectures (Monolithic, Modular-Prog, and Modular-LLM) and task settings (Single-domain and Multi-domain). For single-domain tasks, the performance is reported separately for Restaurant (R), Hotel (H), and Train (T) domains, along with their average. For multi-domain tasks, performance is reported for each domain pair (e.g., Restaurant–Hotel (R-H), Hotel–Train (H-T), Train–Restaurant (T-R)), along with the corresponding average.

Across architectures, models generally achieve higher booking rates on single-domain tasks compared to multi-domain tasks, reflecting the increased complexity introduced by multi-domain interactions. For single-domain tasks, the majority of failures are due to the model not adhering strictly to the specified response format constraints. For multi-domain tasks, in addition to format violations, several other issues contribute to performance degradation. These include user simulator occasionally terminating the conversation prematurely after completing only one task, and models assuming missing information (such as hotel area or restaurant food type) that differed from the ground-truth data, resulting in different booking choices and thus reduced the overall scores.

A.6 Cost Estimation

We estimate computational costs across dialogue system architectures and model scales by analyzing both token-based and FLOP-based expenses. This evaluation provides insights into the computational trade-offs associated with different dialogue system architectures.

We followed a standardized estimation methodology (Kaplan et al., 2020; Chowdhery et al., 2023) to compute the number of tokens, floating-point operations (FLOPs), and their associated costs.

Token Cost Computation Although all open-weight models were run locally, token costs were estimated using pricing information from OpenRouter APIs [7] to provide an indicative view of real-world deployment expenses. The total token cost (T) is computed as the sum of the prompt(p) and response(r) token costs, using the respective input (c_i) and output (c_o) pricing rates for each model:

$$T = (p \times c_i) + (r \times c_o)$$

Flop Cost Computation The FLOP cost (F) is derived by first estimating the total number of floating-point operations (FP) required to process all tokens, using an estimate (Kaplan et al., 2020; Chowdhery et al., 2023) of $2\times$ the number of model parameters per token. This value is then converted into petaflops and priced based on a standard cost assumption. Specifically, we assume a cost of \$0.05 per petaFLOP (c_{pf}), derived from an A100 GPU rental rate of approximately \$2 per hour. This allows us to estimate the computational overhead of inference independently of specific deployment hardware.

[7] https://openrouter.ai/models

Domain	Model	Restaurant	Hotel	Train	Avg
Single	Llama-3.2-1B	0.00	0.00	0.00	0.00
	Llama-3.2-3B	0.35	0.00	0.65	0.33
	Qwen2.5-7B	0.20	0.10	0.40	0.23
	Llama-3.1-8B	0.82	0.25	0.65	0.57
	Qwen2.5-32B	0.90	0.80	0.90	**0.87**
	Llama-3.3-70B	0.70	0.50	0.85	0.68
	GPT-4o	0.80	0.60	0.90	0.77
Domain	Model	R-H	H-T	T-R	Avg
Multi	Llama-3.2-1B	0.00	0.00	0.00	0.00
	Llama-3.2-3B	0.00	0.00	0.05	0.02
	Qwen2.5-7B	0.00	0.00	0.10	0.03
	Llama-3.1-8B	0.00	0.00	0.00	0.00
	Qwen2.5-32B	0.35	0.35	0.35	0.42
	Llama-3.3-70B	0.30	0.20	0.70	0.40
	GPT-4o	0.35	0.45	0.75	**0.52**

Table 14: Performance of monolithic dialogue systems evaluated with Qwen2.5-32B as the user simulator on the MultiWOZ-style synthetic dataset. Results are reported for single-domain (Restaurant, Hotel, Train) and multi-domain (Restaurant-Hotel, Hotel-Train, Train-Restaurant) tasks. "Avg" denotes the average score across the domains.

Domain	Model	Restaurant	Hotel	Train	Avg
Single	Llama-3.2-1B	0.00	0.00	0.00	0.00
	Llama-3.2-3B	0.00	0.00	0.10	0.03
	Qwen2.5-7B	0.05	0.05	0.05	0.05
	Llama-3.1-8B	0.20	0.05	0.15	0.13
	Qwen2.5-32B	0.10	0.50	0.25	0.28
	Llama-3.3-70B	0.05	0.05	0.20	0.10
	GPT-4o	0.15	0.80	0.40	**0.45**
Domain	Model	R-H	H-T	T-R	Avg
Multi	Llama-3.2-1B	0.00	0.00	0.00	0.00
	Llama-3.2-3B	0.00	0.00	0.00	0.00
	Qwen2.5-7B	0.00	0.00	0.00	0.00
	Llama-3.1-8B	0.00	0.00	0.00	0.00
	Qwen2.5-32B	0.10	0.10	0.00	0.07
	Llama-3.3-70B	0.00	0.05	0.05	0.03
	GPT-4o	0.15	0.45	0.05	**0.22**

Table 15: Performance of monolithic dialogue systems evaluated with Qwen2.5-32B as the user simulator on the unrealistic synthetic dataset. Results are reported for single-domain (Restaurant, Hotel, Train) and multi-domain (Restaurant-Hotel, Hotel-Train, Train-Restaurant) tasks. "Avg" denotes the average score across the domains.

$$FP = 2 \times \text{model parameters}$$

$$\text{Total PetaFLOPs} = \frac{(p \times FP) + (r \times FP)}{10^{15}}$$

$$F = \text{Total PetaFLOPs} \times c_{pf}$$

A.7 Synthetic Dataset Generation

To assess the adaptability and generalization capabilities of the `clem:todd` framework, we created two types of synthetic test datasets: one that mimics the structure of the original MultiWOZ 2.2 benchmark but introduces unseen goal configurations, and another that contains intentionally unrealistic or adversarial dialogue tasks.

A.7.1 MultiWOZ-Style Synthetic Dataset

This dataset preserves the general structure and domain constraints of MultiWOZ 2.2 (restaurant, hotel, and train), but combines slot values and goals in configurations not found in the original corpus. The goal was to simulate unseen but similar tasks to assess how well systems generalize.

To construct the MultiWOZ-style synthetic dataset, we first collected the available slots for each domain (restaurant, hotel, and train) and then randomly generated novel combinations of these slots. Each generated goal was checked to ensure that the exact same combination of slot types and values did not exist in the original MultiWOZ 2.2 corpus, thereby guaranteeing the inclusion of unseen goal configurations. These synthetic goals were then rendered into natural language using templates. We generated a total of 120 task goals to match the size of the baseline evaluation set, comprising 60 single-domain and 60 multi-domain dialogue tasks.

A.7.2 Unrealistic Synthetic Dataset

For the unrealistic dataset, we constructed a custom ontology that retained the original slot names from the MultiWOZ dataset, such as area, food, hotel type, and travel day etc., but replaced their values with unusual values. For example, area values included "middle of ocean" and "top of volcano", hotel type included "dungeon" and "wormhole", food included "rotten" or "leftover", and travel days were set as "someday" or "yesterday". Using this ontology, we generated random combinations of slot-value pairs and then rendered them into natural language goals using the same template-based generation approach as the synthetic MultiWOZ-style dataset. We created a total of 120 tasks, 60 single-domain and 60 multi-domain, to mirror the scale of the baseline evaluation set. These tasks, while structurally valid, were semantically unrealistic, allowing us to probe the dialogue systems' behavior in the face of adversarial user requests.

TEMPLATE A.7.1

System Info

ROLE: You are a user simulator tasked with interacting naturally with a dialogue system.

TASK:

$goal

INSTRUCTIONS:

1. Communicate naturally by expressing preferences, asking clarifying questions, and making decisions as needed.
2. Maintain a polite and conversational tone.
3. Respond strictly based on the dialogue system's response. Do not add logic or interpretation beyond what is explicitly stated in the TASK.
4. Ensure that names and terms remain exactly as provided in the input, without any added or altered punctuation (e.g., do not add apostrophes, hyphens, or other symbols). Maintain strict adherence to the original formatting.
5. When booking a train, exact time matches may not always be available. If the dialogue system provides alternative options close to the desired time, the you should accept a suitable nearby option that reasonably aligns with the goal.
6. Once the dialogue system completes the task and provides the reference number, reply with "DONE". No additional text should follow/preceed.
7. Do not simulate or act as the dialogue system; only interact with it.
8. Keep responses concise and focused, avoiding unnecessary elaboration or overly conversational tone.

OUTPUT FORMAT

1. Interaction: Respond appropriately using only the dialogue system's response and the information under TASK.
2. Task Completion: Reply with "DONE". Do not add any customary comments (thank you, great etc.)
3. Use 'DONE' only (without any prefix/suffix) to confirm task completion.
Lets begin

Figure 6: Prompt template for the User Simulator, specifying the task description, interaction instructions, and response format guidelines.

TEMPLATE A.7.2
System Info

ROLE: You are a specialized booking assistant interacting with a human user through JSON function calls using the provided tool schema. Your role is to process user requests and ensure successful task completion while maintaining a professional, helpful tone. You are NOT allowed to return free-form messages outside tool calls.

TASK:

Assist the user conversationally by:
1. Extracting key details needed for the task (e.g., domain, date, time, location).
2. Cross-referencing user-provided information with the database to find relevant matches.
3. If too many records are available, the database system returns only the first five. If the required information is not available in the returned records, apply additional filters to narrow down the results.
4. Generating responses to gather missing or unclear information or to provide the booking status.
5. For train bookings, if the database does not have trains available at the exact requested time, clarify with the user whether they are interested in seeing the closest available options that best match their query.
6. Consolidating all extracted and clarified details for booking finalization.
7. Keeping responses concise and focused, avoiding unnecessary elaboration or overly conversational tone.
8. Do not assume any details; always ask the user for clarification when necessary.

INSTRUCTIONS:

1. Communicate naturally by expressing preferences, asking clarifying questions, and making decisions as needed.
2. Maintain a polite and conversational tone.
3. Respond strictly based on the dialogue system's response. Do not add logic or interpretation beyond what is explicitly stated in the TASK.
4. Ensure that names and terms remain exactly as provided in the input, without any added or altered punctuation (e.g., do not add apostrophes, hyphens, or other symbols). Maintain strict adherence to the original formatting.
5. When booking a train, exact time matches may not always be available. If the dialogue system provides alternative options close to the desired time, the you should accept a suitable nearby option that reasonably aligns with the goal.
6. Once the dialogue system completes the task and provides the reference number, reply with "DONE". No additional text should follow/preceed.
7. Do not simulate or act as the dialogue system; only interact with it.
8. Keep responses concise and focused, avoiding unnecessary elaboration or overly conversational tone.

OUTPUT FORMAT

1. Interaction: Respond appropriately using only the dialogue system's response and the information under TASK.
2. Task Completion: Reply with "DONE". Do not add any customary comments (thank you, great etc.)
3. Use 'DONE' only (without any prefix/suffix) to confirm task completion.
Lets begin

$USER_SIMULATOR_UTTERANCE

Figure 7: Prompt template for the monolithic dialogue system, detailing the task procedures, interaction instructions, and output format for user request processing via JSON function calls.

TEMPLATE A.7.3
System Info

ROLE: You are the dialogue manager for a specialized booking assistant bot and interact through JSON function calls using the provided tool schema. Your role is to process user requests, coordinate interactions with subsystems, and ensure successful task completion. You are NOT allowed to return free-form messages outside tool calls.

TASK:

1. For each user request:
a. Determine appropriate flow based on user input and available information
b. Identify next required subsystem. Always use the exact subsystem names as specified in the tool schema.
c. Prepare the necessary input data for that subsystem
d. For database queries and validating booking information, use the exact function names as specified in the tool schema.
e. Do not generate any booking confirmation (reference number) on your own. Use the appropriate function to validate the booking and to generate the reference number.
2. All responses must strictly adhere to the format. Include all required fields and the response must be a valid JSON.

RESPONSE RULESS:

1. Use the most appropriate function call based on the user's request and available data.
2. To interact between the sub-systems (intent detection, slot extraction, or response generation), call the 'processnextsubsystem' function.
3. To respond to the user, as a final message after coordinating with the dialogue sub-systems call the 'followup' function.
4. Similarly for booking action or database lookup, use the appropriate function from the tool schema.
5. Every response MUST be a valid tool call (tool_call). Never respond with plain text.
6. Only one function call is allowed per turn. Never return multiple function calls in a single response. If multiple actions are needed, handle them sequentially across turns.

USER REQUEST:

$USER_SIMULATOR_UTTERANCE

Figure 8: Prompt Template for the Modular-LLM Dialogue Manager, specifying task responsibilities, response rules, and interaction guidelines for subsystem coordination.

TEMPLATE A.7.4

System Info

You are required to evaluate a task oriented dialogue on several metrics, including task completion, naturalness, coherence and dialogue-level diversity. Alongside the dialogue, you are also provided with a user goal which states the specific requirement of the user.

TASK:

Here is some detailed explanations for the metrics:
1. Task completion You should check whether each intention in the user goal is fulfilled in the conversation. The task is completed ONLY if all the intentions are fulfilled. This would be a binary metric and you should only response with Yes or No.
This would be a binary metric and you should only response with Yes or No.

2. Naturalness
This metric measures the resemblance to human.
In the dialogue, the user or the system could either be AI or human.
You should report a numeric rating from 1 to 5, where 5 represents most likely to be human.
You are required to evaluate the naturalness of both the user and the system.
Here are some more detailed guidelines for naturalness for your reference:
1: The speaker continuously repeat itself, typical robotic behavior. Or the speech is hard to understand.
2: The speaker repeat itself occasionally, the vocabulary is limited, like a robot.
3: The speaker does not have repeated behaviors (unless for verifying information). Vocabulary is enough to communicate effectively, speech is easy to understand. But I am confident that human rarely speak like this.
4: The speaker is likely to be a human. There is rarely logical inconsistency. But from some details I feel like the utterance is a bit weird and somewhat resembles AI.
5: Can not really tell if this is AI or human. Human could probably say the same thing in real life.

3. Coherence
This metric measures the logical consistency within a dialogue.
You should report a numeric rating from 1 to 3, where 3 represents the best coherence.
Here is some detailed guidelines for coherence.
a. Locally, the utterances are coherent/logical based on previous turns of conversations.
b. Globally, the utterances reasonably and logically adhere to achieving the initial user goal step by step.
If both conditions a and b are satisfied, you should give a score of 3. If only one condition is satisfied, you should give a score of 2. Report 1 if none of the conditions are satisfied.

4. Dialogue-level diversity
In addition to trying to achieve the initial goal, does the user introduce some reasonable deviations from the normal conversation flow?
Give a score from:
3 (highest score): > 20% of the time (frequently deviate from normal flow of the conversation)
2: 0% < deviation frequency < 20% (Normal)
1 (lowest score): 0% (too artificial, maximizing information exchange)

Note that for naturalness and coherence, you need to evaluate both the user and the system. For dialogue-level diversity, you only need to evaluate the user.
You should return 6 results in total, with the order of task completion, naturalness for the user, natualness for the system, coherence for the user, coherence for the system, diversity for the user. Each evaluation results should be separated by commas. For example, 'Yes,5,3,3,1,2' will be a valid response.
Please be strict on the format of your response. Do not include any other words like 'Sure!', 'Here is the result:'. Simply response with only the results.

The user goal is as following:
$user_goal

The dialogue to be evaluated is as following:
$dialogue

Figure 9: Prompt template for Dialogue Evaluation Task, describing detailed guidelines for assessing task completion, naturalness, coherence, and dialogue-level diversity.

TEMPLATE A.7.5

System Info

```
ROLE: You are an Intent Detection system designed to classify user requests into predefined domains and intents using the provided tool schema. You are NOT allowed to return free-form messages outside tool calls.

AVAILABLE INTENTS:

For all intent detections, use these exact names:
1. booking-request: User wants to proceed with the booking.
2. booking-success: The booking was successful and has some booking number.
3. booking-failure: There is a failure in the booking.
4. dbretrieval-request: User is looking for some information.
5. dbretrieval-success: The data is fetched from the DB and the retrieval was successful.
6. dbretrieval-failure: There is a failure in fetching the data from the DB.
7. detection-unknown: If the input doesn't fall into any of the above

AVAILABLE DOMAINS:

1.  Classify the request into only one of the following domains (choose the closest match):
* restaurant, hotel, train
2. Not all utterances can be categorized into a domain. In such cases, use "donotcare".

TASK:

1.   Analyze the provided input.   Dialogue history is provided to understand the context better.
2. Classify the request into only one of the above predefined intents (the closest match) and domain
3. Return the detected intent and domain by using those exact names.
4. Do not add any other information or explanation or comments.
5. Every response MUST be a valid tool call (tool_call). Never respond with plain text.
6. Only one function call is allowed per turn.

INPUT:

$USER_SIMULATOR_UTTERANCE
```

Figure 10: Prompt template for the Intent Detection module, specifying task guidelines for classifying user requests into predefined intents and domains.

TEMPLATE A.7.6

System Info

```
ROLE: You are an Slot Extraction system designed to identify and extract key entities from
user requests to support downstream tasks using the provided tool schema. You are NOT allowed to
return free-form messages outside tool calls.
```

TASK:

```
1. Analyze the provided user request.
2. Identify and extract relevant slots (e.g., name, area, time, date, type of cuisine, number of
people, type of hotel) based on the task context.
3. Focus on extracting the most concise and precise values for each slot, avoiding unnecessary
descriptive phrases or additional words.
4. Return the extracted slots in a structured format.
5. Return only the formatted data—do not add explanations, comments, or additional information.
6. Only extract a slot if it is **explicitly mentioned** in the user input. Do not infer, assume, or
hallucinate values based on common patterns or prior examples.
7. If a relevant slot is not present in the input, **omit it from the output entirely**—do not
fabricate or guess.
8. When handling follow-up user requests, compare the new input to the dialogue history:
- If a slot was **previously extracted** but the new input **replaces or contradicts** it (e.g.,
rephrasing or simplifying),
then **explicitly reset** that slot by setting its value to an empty string (e.g., ‘"area": ""‘).
- If the user is merely **adding new information** (e.g., number of people, dates), do **not** reset
previous values—**preserve them**.
```

RESPONSE RULES:

```
1. Use the most appropriate function call based on the user’s request and available data.
2. Every response MUST be a valid tool call (tool_call). Never respond with plain text.
3. Only one function call is allowed per turn.
```

USER REQUEST:

```
$USER_SIMULATOR_UTTERANCE
```

Figure 11: Prompt template for the Slot Extraction module, outlining task instructions for identifying and extracting structured key entities from user requests.

TEMPLATE A.7.7

System Info

ROLE: You are a Response Generation system responsible for crafting contextually appropriate and concise replies based strictly on the provided input using the provided tool schema. You are NOT allowed to return free-form messages outside tool calls.

TASK:

Given the input data (domain, intent, extracted slots, database (DB) information, and dialogue history):
1. Generate a meaningful response:
a. If additional information is required to proceed, respond conversationally using direct and focused phrasing.
b. If recommendations are provided in the DB:
* Ask the user to choose from the list of options.
* Clearly present all options to the user for selection. Do not decide on any recommendation yourself.
c. For train bookings, if the database does not have trains available at the exact requested time, clarify with the user whether they are interested in seeing the closest available options that best match their query.

2. Guidelines for Response:

a. Responses must be concise and to the point.
b. Avoid unnecessary elaboration or an overly conversational tone.
c. Do not generate or fabricate any information that is not explicitly present in the DB or provided input.
d. If too many records are available, the database system returns only the first five. If the required information is not available in the returned records, request the user for additional information to narrow down the results.
e. Do not generate any booking confirmation (reference number) on your own.
f. If the input contains booking confirmation (reference number), share the same to user without any changes.
g. Every response MUST be a valid tool call (tool_call). Never respond with plain text.
h. Only one function call is allowed per turn. Never return multiple function calls in a single response. If multiple actions are needed, handle them sequentially across turns.

INPUT:

$USER_SIMULATOR_UTTERANCE

Figure 12: Prompt template for the Response Generation module, specifying task and response guidelines for producing contextually appropriate and structured replies based on user input and dialogue history.

```
[
       {
           "type": "function",
           "function": {
               "name": "followup",
               "description": "Use this function to respond to the user with follow-up messages. This includes asking for missing
                     or unclear information, confirming details, sharing booking reference numbers, or continuing the dialogue
                     based on the current conversation state.",
               "parameters": {
                   "type": "object",
                   "properties": {
                       "message": {
                           "type": "string",
                           "description": "The response from the dialogue system to the user"
                       }
                   },
                   "required": ["message"],
                   "additionalProperties": false
               }
           }
       },
       {
           "type": "function",
           "function": {
               "name": "retrievefromrestaurantdb",
               "description": "Use this function to query the restaurant database and retrieve restaurants that match optional
                     filters such as area, pricerange, food (cuisine), or restaurant name. This function is typically used to find
                      available restaurant options before validating or making a reservation. Returns up to 5 matching restaurants,
                      or fewer if less than 5 matches are found.",
               "parameters": {
                   "type": "object",
                   "properties": {
                       "area": {
                           "type": "string",
                           "enum": ["centre", "north", "east", "west", "south"],
                           "description": "The area/location/place of the restaurant. Optional."
                       },
                       "pricerange": {
                           "type": "string",
                           "enum": ["cheap", "moderate", "expensive"],
                           "description": "The price budget for the restaurant. Optional."
                       },
                       "food": {
                           "type": "string",
                           "description": "The cuisine of the restaurant you are looking for. Optional."
                       },
                       "name": {
                           "type": "string",
                           "description": "The name of the restaurant. Optional."
                       }
                   },
                   "required": []
               }
           }
       }
]
```

Figure 13: Schema definition detailing parameters and constraints for querying the database and booking confirmation (Part 1/6).

```
[
       {
           "type": "function",
           "function": {
               "name": "retrievefromhoteldb",
               "description": "Use this function to query the hotel database and retrieve hotels/guesthouses that match
                     optional filters such as area, pricerange, type, hotel name, internet, parking, or stars. This function
                      is typically used to find available hotel options before validating or making a reservation. Returns
                     up to 5 matching hotels, or fewer if less than 5 matches are found.",
               "parameters": {
                   "type": "object",
                   "properties": {
                       "area": {
                           "type": "string",
                           "enum": ["centre", "north", "east", "west", "south"],
                           "description": "The area/location/place of the hotel. Optional."
                       },
                       "pricerange": {
                           "type": "string",
                           "enum": ["cheap", "moderate", "expensive"],
                           "description": "The price budget for the hotel. Optional."
                       },
                       "type": {
                           "type": "string",
                           "enum": ["hotel", "guesthouse"],
                           "description": "What is the type of the hotel. Optional."
                       },
                       "name": {
                           "type": "string",
                           "description": "The name of the hotel. Optional."
                       },
                       "internet": {
                           "type": "string",
                           "enum": ["yes", "no"],
                           "description": "Indicates, whether the hotel has internet/wifi or not. Optional."
                       },
                       "parking": {
                           "type": "string",
                           "enum": ["yes", "no"],
                           "description": "Indicates, whether the hotel has parking or not. Optional."
                       },
                       "stars": {
                           "type": "object",
                           "description": "The star rating of the hotel. Optional.",
                           "properties": {
                               "operator": { "type": "string", "enum": ["=", ">=", "<=", ">", "<"] },
                               "value": { "type": "string", "enum": ["1", "2", "3", "4", "5"] }
                           },
                           "required": ["operator", "value"],
                           "additionalProperties": false
                       }
                   },
                   "required": []
               }
           }
       }
]
```

Figure 14: Schema definition detailing parameters and constraints for querying the database and booking confirmation (Part 2/6).

```
[
       {
           "type": "function",
           "function": {
               "name": "retrievefromtraindb",
               "description": "Use this function to query the train database and retrieve trains that match optional filters
                      such as destination, departure, day, arriveby, or leaveat. This function is typically used to find
                     available options before validating or making a reservation. Returns up to 5 matching trains, or fewer
                     if less than 5 matches are found.",
               "parameters": {
                   "type": "object",
                   "properties": {
                       "destination": {
                           "type": "string",
                           "description": "Destination of the train. Optional."
                       },
                       "departure": {
                           "type": "string",
                           "description": "Departure location of the train. Optional."
                       },
                       "day": {
                           "type": "string",
                           "enum": ["monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday"],
                           "description": "Journey day of the train. Optional."
                       },
                       "arriveby": {
                           "type": "object",
                           "description": "Arrival time of the train. Optional.",
                           "properties": {
                               "operator": { "type": "string", "enum": ["=", ">=", "<=", ">", "<"] },
                               "value": { "type": "string", "pattern": "^(0[0-9]|1[0-9]|2[0-3]):[0-5][0-9]$",
                                         "description": "A time string formatted as HH:MM (24-hour format)."
                                       }
                           },
                           "required": ["operator", "value"],
                           "additionalProperties": false
                       },
                       "leaveat": {
                           "type": "object",
                           "description": "Leaving time for the train. Optional.",
                           "properties": {
                               "operator": { "type": "string", "enum": ["=", ">=", "<=", ">", "<"] },
                               "value": { "type": "string", "pattern": "^(0[0-9]|1[0-9]|2[0-3]):[0-5][0-9]$",
                                         "description": "A time string formatted as HH:MM (24-hour format)."
                                       }
                           },
                           "required": ["operator", "value"],
                           "additionalProperties": false
                       }
                   },
                   "required": []
               }
           }
       }
]
```

Figure 15: Schema definition detailing parameters and constraints for querying the database and booking confirmation (Part 3/6).

```
[
    {
        "type": "function",
        "function": {
            "name": "validaterestaurantbooking",
            "description": "Use this function to check the availability of a restaurant based on user preferences such as
                  area, food (cuisine), pricerange, name, people, day, and time before proceeding with a reservation. This
                  function should be called to validate whether a booking can be made with the provided details. If the
                  details are accurate, it returns a booking reference number.",
            "parameters": {
                "type": "object",
                "properties": {
                    "area": {
                        "type": "string",
                        "enum": ["centre", "north", "east", "west", "south"],
                        "description": "The area/location/place of the restaurant."
                    },
                    "pricerange": {
                        "type": "string",
                        "enum": ["cheap", "moderate", "expensive"],
                        "description": "The price budget for the restaurant."
                    },
                    "food": {
                        "type": "string",
                        "description": "The cuisine of the restaurant you are looking for."
                    },
                    "name": {
                        "type": "string",
                        "description": "The name of the restaurant."
                    },
                    "phone": {
                        "type": "string",
                        "description": "Phone number of the restaurant. Optional."
                    },
                    "postcode": {
                        "type": "string",
                        "description": "Postal code of the restaurant. Optional."
                    },
                    "address": {
                        "type": "string",
                        "description": "Address of the restaurant. Optional."
                    },
                    "people": {
                        "type": "string",
                        "enum": ["1", "2", "3", "4", "5", "6", "7", "8"],
                        "description": "Number of people for the restaurant reservation."
                    },
                    "day": {
                        "type": "string",
                        "enum": ["monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday"],
                        "description": "Day of the restaurant reservation."
                    },
                    "time": {
                        "type": "string",
                        "pattern": "^(0[0-9]|1[0-9]|2[0-3]):[0-5][0-9]$",
                        "description": "Time of the restaurant reservation, formatted as HH:MM (24-hour format)."
                    },
                },
                "required": ["food", "area", "pricerange", "name", "people", "day", "time"],
                "additionalProperties": false
            }
        }
    }
]
```

Figure 16: Schema definition detailing parameters and constraints for querying the database and booking confirmation (Part 4/6).

```
[
    {
        "type": "function",
        "function": {
            "name": "validatehotelbooking",
            "description": "Use this function to check the availability of a hotel based on user preferences such as area,
                  type (hotel/guesthouse), pricerange, name, internet, parking, stars, people, day and stay before proceeding
                   with a reservation. This function should be called to validate whether a booking can be made with the
                  provided details. If the details are accurate, it returns a booking reference number.",
            "parameters": {
                "type": "object",
                "properties": {
                    "area": {
                        "type": "string",
                        "enum": ["centre", "north", "east", "west", "south"],
                        "description": "The area/location/place of the hotel."
                    },
                    "pricerange": {
                        "type": "string",
                        "enum": ["cheap", "moderate", "expensive"],
                        "description": "The price budget for the hotel."
                    },
                    "type": {
                        "type": "string",
                        "enum": ["hotel", "guesthouse"],
                        "description": "What is the type of the hotel."
                    },
                    "name": {
                        "type": "string",
                        "description": "The name of the hotel."
                    },
                    "internet": {
                        "type": "string",
                        "enum": ["yes", "no"],
                        "description": "Indicates, whether the hotel has internet/wifi or not."
                    },
                    "parking": {
                        "type": "string",
                        "enum": ["yes", "no"],
                        "description": "Indicates, whether the hotel has parking or not."
                    },
                    "stars": {
                        "type": "string",
                        "enum": ["1", "2", "3", "4", "5"],
                        "description": "The star rating of the hotel."
                    },
                    "people": {
                        "type": "string",
                        "enum": ["1", "2", "3", "4", "5", "6", "7", "8"],
                        "description": "Number of people for the hotel booking."
                    },
                    "day": {
                        "type": "string",
                        "enum": ["monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday"],
                        "description": "Day of the hotel booking."
                    },
                    "stay": {
                        "type": "string",
                        "enum": ["1", "2", "3", "4", "5", "6", "7", "8"],
                        "description": "Length of stay at the hotel."
                    },
                    "phone": {
                        "type": "string",
                        "description": "Phone number of the hotel. Optional."
                    },
                    "postcode": {
                        "type": "string",
                        "description": "Postal code of the hotel. Optional."
                    },
                    "address": {
                        "type": "string",
                        "description": "Address of the hotel. Optional."
                    }
                },
                "required": ["area", "pricerange", "type", "internet",
                             "parking", "name", "stars", "people", "day", "stay"],
                "additionalProperties": false
            }
        }
    }
]
```

Figure 17: Schema definition detailing parameters and constraints for querying the database and booking confirmation (Part 5/6).

```
[
       {
           "type": "function",
           "function": {
               "name": "validatetrainbooking",
               "description": "Use this function to check the availability of a train based on user preferences such as
                     destination, departure, arriveby, leaveat, day, people, and trainid before proceeding with a
                     reservation. This function should be called to validate whether a booking can be made with the provided
                      details. If the details are accurate, it returns a booking reference number.",
               "parameters": {
                   "type": "object",
                   "properties": {
                       "destination": {
                           "type": "string",
                           "description": "Destination of the train."
                       },
                       "departure": {
                           "type": "string",
                           "description": "Departure location of the train."
                       },
                       "day": {
                           "type": "string",
                           "enum": ["monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday"],
                           "description": "Journey day of the train."
                       },
                       "arriveby": {
                           "type": "string",
                           "pattern": "^(0[0-9]|1[0-9]|2[0-3]):[0-5][0-9]$",
                           "description": "Arrival time of the train."
                       },
                       "leaveat": {
                           "type": "string",
                           "pattern": "^(0[0-9]|1[0-9]|2[0-3]):[0-5][0-9]$",
                           "description": "Leaving time for the train."
                       },
                       "people": {
                           "type": "string",
                           "enum": ["1", "2", "3", "4", "5", "6", "7", "8"],
                           "description": "Number of train tickets for the booking."
                       },
                       "trainid": {
                           "type": "string",
                           "description": "ID of the train."
                       },
                       "price": {
                           "type": "string",
                           "description": "Price of the train journey. Optional."
                       },
                       "duration": {
                           "type": "string",
                           "description": "Duration of the travel. Optional."
                       }
                   },
                   "required": ["destination", "departure", "day", "arriveby", "leaveat", "people", "trainid"],
                   "additionalProperties": false
               }
           }
       }
]
```

Figure 18: Schema definition detailing parameters and constraints for querying the database and booking confirmation (Part 6/6).