pxpx💬 SDialog: 一个用于合成对话生成和分析的 Python 工具包 *

Sergio Burdisso[1], Esaú Villatoro-Tello[1], and Petr Motlicek[1,2]

[1] Idiap Research Institute, Martigny, Switzerland
[2] Brno University of Technology, Brno, Czech Republic
{ sergio.burdisso,esau.villatoro,petr.motlicek } @idiap.ch

Abstract. 对话式人工智能系统的发展依赖于高质量、灵活且可重现的合成对话的可用性，以用于训练、评估和基准测试。SDialog 是一个模块化、可扩展的 Python 工具包，旨在解决合成对话生成和分析的挑战。通过利用经过指令微调的大型语言模型（LLMs），SDialog 提供了用于角色、编排和场景管理的抽象，能够创建用于研究和开发的真实、多样且可控的对话数据。SDialog 支持多代理模拟和场景驱动生成等工作流程，并在合成数据生成工具和框架标准化方面迈出了重要一步——这对于确保在当今快速变化的研究环境中实现可重现性至关重要。

Keywords: synthetic dialogue, conversational AI, large language models, persona, orchestration, scenario management, dialogue simulation, dataset, benchmarking, reproducibility

1 介绍

对话人工智能（AI）领域已经取得了显著的进展，特别是随着大型语言模型（LLMs）[1,9,8,7,6] 的兴起。这些模型使得开发能够进行细致和信息丰富对话的复杂代理成为可能。然而，对话 AI 研究的进展和强健对话系统的部署在根本上依赖于高质量、灵活且可再现的合成对话数据的可用性。

合成数据生成已经成为训练、评估和基准测试会话代理的基石，特别是因为真实世界的带注释对话数据集往往有限、获取成本高昂或受隐私问题限制。生成多样化的、上下文感知的、知识为基础的合成对话的能力对于推动该领域的发展至关重要。例如，上下文感知型对话 AI 旨在理解和利用对话的动态历史和具体细节，包括用户意图和先前的话轮，以生成相关且连贯的响应 [11,10] 。类似地，整合知识库使得会话代理能够将其响应基于事实的、特定领域的数据和语义关系，从而生成更准确和富有洞察力的答案 [12,4,5] 。

随着合成对话越来越多地应用于敏感领域，如医疗、金融和法律援助，数据生成的可解释性和透明性变得至关重要。例如，最近的研究强调了一项知名心理健康数据集中存在的意外偏见，该数据集中的临床访谈由一个扮演治疗师的人类控制虚拟代理进行。由于该虚拟代理的多样性有限、行为确定性强及词汇量小，在该数据上训练的模型很容易学会利用表面的模式作为预测捷径，而不是学习对抑郁人群语言建模 [3] 。

*https://github.com/idiap/sdialog

尽管有这些需求，生成现实的、可再现的并且能够适应各种实验需求的合成对话仍然是一个挑战。在对话、角色和事件结构方面缺乏标准化，并且需要支持精细控制、场景管理和无缝集成 LLM 的模块化、可扩展框架。

在本文中，我们介绍了一款名为 SDialog 的 Python 工具包，专门用于合成对话的生成和分析。该工具包为人物、编排和场景管理提供了抽象，使研究人员和从业者能够创建真实、多样且可控的对话数据，以便进行研究和开发。通过应对合成数据生成的挑战，SDialog 旨在加速对话 AI 的进步，并支持可重复、透明的实验。

2 需求说明

会话式人工智能的研究和应用越来越需要那些不仅真实而且可以复现和适应各种实验需求的合成对话。然而，生成这样的对话存在几个挑战：SDialog 通过提供一个全面的框架来满足这些需求，支持合成对话生成和分析、基于个性角色扮演、多代理和协调对话、场景和数据集集成，以及灵活的序列化和可视化。

表格 **??** 提供了 SDialog 的主要资源和元数据的概览，包括文档、教程、许可证和联系信息。

3 实现

SDialog 围绕几个核心抽象和模块构建，每个模块都旨在提供灵活性、可扩展性以及在研究和开发环境中的易用性。下面详细描述了主要组件，并以代码片段说明它们的使用。

3.1 对话数据结构

SDialog 的核心是表示对话构建模块的数据结构。Turn 类模拟了对话中的单一话语，同时捕捉说话者和话语内容：

```
from sdialog import Turn

turn = Turn(speaker="Alice", text="Hello, how can I help you?")
print(turn)
```

事件类捕捉对话中的动作，这些动作可能包括话语、指令或其他事件，并提供元数据用于跟踪和分析对话的流程和编排：

```
from sdialog import Event
import time

event = Event(
    agent="System",
    action="utter", actionLabel=None,
    text="Welcome!", timestamp=int(time.time())
)
print(event)
```

Dialog 类表示一个完整的对话，包括其回合序列、关联事件和场景元数据。它提供了序列化、美观打印和文件 I/O 的方法，支持人类可读和机器可处理的格式：

```python
from sdialog import Dialog, Turn

dialog = Dialog(
    scenario={"topic": "coffee order", "location": "cafe"}, # (optional) scenario metadata
    events=[...], # (optional) list of Event objects for detailed tracking
    turns=[
        Turn(speaker="Alice", text="Hi!"),
        Turn(speaker="Bob", text="Hello, Alice!"),
        ...
    ],
    ...
)

print(dialog) # print as plain text
dialog.print() # pretty print
```

这些结构可以实现对合成对话的操控、分析和再现。关于对话及本节描述的其余组件的所有可用属性的完整描述，请参阅文档和 API 参考，如表 **??** 所示。

3.2 人物角色与代理人

SDialog 通过允许定义详细的角色档案和模拟扮演这些角色的代理角色来实现基于人格的对话生成。Persona 类为对话生成中的角色扮演定义了一个角色档案，包括名字、角色、背景、个性、环境、规则和语言等字段：

```python
from sdialog import Persona

alice = Persona(
    name="Alice",
    role="barista",
    background="Works at a busy coffee shop.",
    personality="cheerful and helpful",
    circumstances="Morning shift",
    rules="Always greet the customer",
    language="English"
)
print(alice)
```

PersonaAgent 类模拟一个使用 LLM 扮演给定 Persona 的代理。它维护会话的记忆，支持用于插入指令或控制行为的协调，并且可以被设置种子以生成可重复的对话：

```python
from sdialog import Persona, PersonaAgent

alice = Persona(name="Alice", role="barista", personality="cheerful")
bob = Persona(name="Bob", role="customer", personality="curious")

alice_agent = PersonaAgent("llama2", persona=alice, name="Alice")
bob_agent = PersonaAgent("llama2", persona=bob, name="Bob")

dialog = alice_agent.dialog_with(bob_agent)
dialog.print()
```

PersonaAgent 可以序列化其配置和角色以实现可重复性，并支持灵活的问候语/首次话语配置。该抽象支持创建真实、多样化和可控的对话代理。

3.3 编排

为了实现对对话生成的细粒度控制，SDialog 引入了控制器的概念。控制器是模块化组件，可以在对话过程中注入指令、强制执行约束条件或模拟代理的特定

行为。BaseOrchestrator 是所有控制器的抽象基类，提供用于生成指令、管理持久性、事件标记和序列化的方法。例如，可以如下定义一个自定义控制器：

```
from sdialog.orchestrators import BaseOrchestrator

class AlwaysSayHelloOrchestrator(BaseOrchestrator):
   def instruct(self, dialog, utterance):
      if len(dialog) == 0:
         return "Say 'Hello!' as your first utterance."
```

SDialog 提供了几个内置的协调器用于常见的对话控制模式，例如：

- 简单反射调度器：根据条件（例如，如果话语中存在某个关键字）触发指令。
- LengthOrchestrator: 通过根据对话轮数提供继续或结束对话的指令来控制对话长度。
- ChangeMindOrchestrator：模拟代理改变主意，可以选择性地包含理由列表和概率。
- SimpleResponseOrchestrator：基于与一组可能响应的相似性，使用句子嵌入建议响应。
- 指令列表协调器：在特定时机提供一系列指令，对于模拟引导的用户行为非常有用。

编排者可以附加到 PersonaAgent 上，以在对话生成过程中影响其行为，并且可以组合多个编排者以实现复杂的行为：

```
from sdialog import Persona, PersonaAgent
from sdialog.orchestrators import LengthOrchestrator

assistant = Persona(name="Assistant", role="support agent")
assistant_agent = PersonaAgent("llama2", persona=assistant, name="Assistant")
length_orch = LengthOrchestrator(min=5, max=10)

# Add orchestrator using the | operator
assistant_agent = assistant_agent | length_orch
```

3.4 对话生成

SDialog 提供高级生成器来自动创建合成对话，这些对话可以在任意角色之间进行，或者遵循特定的场景指令。DialogGenerator 类使用 LLM 生成合成对话，给定对话细节和输出格式。它支持任意的系统和用户提示、输出模式，并通过设定种子来实现可重复性：

```
from sdialog.generators import DialogGenerator

details = "Generate a conversation between a customer and a barista about ordering
      coffee."
generator = DialogGenerator("llama2", dialogue_details=details)
dialog = generator()
dialog.print()
```

PersonaDialogGenerator 类生成两个角色之间的对话，强制执行角色扮演和场景约束。它自动构建两个角色的系统提示，并确保对话以问候开始并遵循场景指示：

```python
from sdialog.generators import PersonaDialogGenerator, Persona

persona_a = Persona(name="Alice", role="barista")
persona_b = Persona(name="Bob", role="customer")

generator = PersonaDialogGenerator("llama2", persona_a, persona_b)
dialog = generator()
dialog.print()
```

这些生成器利用上述抽象来生成适用于训练、评估和基准测试的真实且结构化的对话。

3.5 数据集和场景

SDialog 包含用于处理外部数据集和管理复杂对话场景的工具。例如，STAR 数据集工具提供用于加载、解析和描述对话、场景、流程图和来自 STAR 数据集的人物角色的功能。这些工具支持以场景为驱动的对话生成和分析，从而实现可重复性研究和模拟现实的、目标驱动的对话：

```python
from sdialog.datasets import STAR

STAR.set_path("/path/to/star-dataset")

# Get dialogue by ID (e.g. 123)
dialog = STAR.get_dialog(123)
dialog.print(scenario=True)

# Get dialogue scenario by dialogue by ID (e.g. 123)
scenario = STAR.get_dialog_scenario(123)

# Get agents for a given scenario, etc.
system_agent, user_agent = STAR.get_agents_for_scenario(scenario, "llama2")
```

SDialog 中的情景管理工具允许生成情景的自然语言描述、提取和可视化流程图，以及基于情景元数据构建角色和代理：

```python
from sdialog.datasets import STAR

scenario = {
   "Domains": ["banking"],
   "UserTask": "Open a new account",
   "WizardTask": "Assist with account opening",
   "Happy": True,
   "MultiTask": False,
   "WizardCapabilities": [{"Task": "open_account", "Domain": "banking"}]
}

system_agent, user_agent = STAR.get_agents_for_scenario(scenario, "llama2")
dialog = system_agent.dialog_with(user_agent)
dialog.print()
```

3.6 效用

为了支持完整的工作流程，SDialog 提供了用于序列化、美化打印和文件输入/输出的实用函数。对话和事件可以导出为 JSON 或纯文本，用于下游任务、训练或分析。灵活的文件 I/O 支持保存和加载对话，美化打印工具允许在控制台中通过颜色编码的说话者和事件对对话进行可视化，以便于检查和调试。还支持场景和编排可视化，使检查和分析生成的对话变得简单直接：

```python
# Export the dialogue to JSON and TXT files
dialog.to_file("output/dialogue_001.json")
dialog.to_file("output/dialogue_001.txt")

# Load a dialogue from file by extension
from sdialog import Dialog
dialog = Dialog.from_file("output/dialogue_001.txt")
dialog = Dialog.from_file("output/dialogue_001.json")
dialog.print()
```

最后，SDialog 支持使用对话流程图进行的高级定性分析和生成及真实对话的可视化。通过利用 Dialog2Flow 嵌入 [2]，用户可以从一组对话中自动提取动作转换图——无论是合成的还是自然的——而无需人工标注。此方法能够直接比较对话流，揭示生成的对话与真实人类互动的结构和复杂性有多大的匹配度。Dialog2Flow 提供静态和交互式的可视化，并允许对发现的动作步骤进行细粒度的抽象级别控制，促进对话数据集的定性和定量分析。

4 说明性例子

在本节中，我们展示了一些使用 SDialog 的实际例子。每个例子都演示了如何结合实现部分的核心 API 来实现简单而明确的用例，并简要解释了该例子旨在实现的目标。

4.1 结合多个编排器进行复杂对话控制

这个示例展示了如何结合多个协调器来模拟更复杂的代理行为和对话约束。每个协调器都会在后台记录事件，使您能够准确追踪对话过程中指令发出的时间。在此场景中，我们不仅控制对话的长度，还模拟一个可能在对话中改变主意的代理。

```python
from sdialog import Persona, PersonaAgent
from sdialog.orchestrators import LengthOrchestrator, ChangeMindOrchestrator

# Define personas for a user and an assistant
user = Persona(name="User", role="customer")
assistant = Persona(name="Assistant", role="support agent")

# Create agents for each persona
user_agent = PersonaAgent("llama2", persona=user, name="User")
assistant_agent = PersonaAgent("llama2", persona=assistant, name="Assistant")

# Orchestrator to control dialogue length (between 10 and 15 turns)
length_orch = LengthOrchestrator(min=10, max=15)

# Orchestrator to simulate the assistant changing their mind up to 2 times
mind_orch = ChangeMindOrchestrator(
    probability=0.4,
    reasons=["changed plans", "new information"],
    max_times=2
)

# Combine orchestrators to compose more complex behavior
assistant_agent = assistant_agent | length_orch | mind_orch

# Generate a dialogue between the orchestrated assistant and the user
dialog = assistant_agent.dialog_with(user_agent)
dialog.print(orchestration=True)
```

在这个例子中，助手代理将在特定长度内保持对话，并可能在对话过程中改变主意，所有的编排事件将被记录以供后续分析。

4.2 批量生成和导出合成对话

这个例子展示了如何自动化生成多个对话进行数据集创建或基准测试，并将其导出用于下游任务。

```python
from sdialog import Persona, PersonaAgent, Dialog

# Define two personas
persona_a = Persona(name="Alice", role="barista")
persona_b = Persona(name="Bob", role="customer")

# Create agents for each persona
alice_agent = PersonaAgent("llama2", persona=persona_a, name="Alice")
bob_agent = PersonaAgent("llama2", persona=persona_b, name="Bob")

# Generate and export 5 dialogues with different seeds/IDs
for i in range(5):
    dialog = alice_agent.dialog_with(bob_agent, id=i, seed=i)
    dialog.to_file(f"output/dialog_{i:03d}.json")
```

此工作流程对于构建合成数据集或使用多个对话样本进行受控实验非常有用。

4.3 基于场景驱动的评估流程与 STAR 数据集

这个例子展示了如何基于 STAR 数据集中的真实场景模拟和评估对话，支持可重复的研究。

```python
from sdialog.datasets import STAR

# Set the STAR dataset path
STAR.set_path("/path/to/star-dataset")

# Iterate over multiple scenarios and generate dialogues
for dialog_id in [101, 102, 103]:
    scenario = STAR.get_dialog_scenario(dialog_id)
    system_agent, user_agent = STAR.get_agents_for_scenario(scenario, "llama2")
    dialog = system_agent.dialog_with(user_agent)
    dialog.to_file(f"output/star_dialog_{dialog_id}.json")
```

在这里，对于 STAR 数据集中的每个场景，我们使用为该场景配置的代理生成一个新的对话，并保存结果以便进一步分析。

4.4 对话分析与可视化

这个例子展示了如何分析和可视化生成的对话，包括用于检查或报告的场景和编排信息。

```python
from sdialog import Dialog

# Load a previously saved dialogue
dialog = Dialog.from_file("output/dialogue_001.json")

# Pretty-print with scenario and orchestration details
dialog.print(scenario=True, orchestration=True)
```

通过启用 ‘scenario=True‘ 和 ‘orchestration=True‘，您可以检查场景元数据和编排事件以及对话轮次。

4.5 自定义评估：按长度或内容筛选对话

这个例子展示了如何根据特定的研究目的过滤或分析生成的对话，例如只选择那些包含最少回合数或包含某些关键词的对话。

```python
from sdialog import Dialog
import glob

# Load and filter dialogues by length
paths = glob.glob("output/dialog_*.json")
long_dialogs = []
for path in paths:
    dialog = Dialog.from_file(path)
    if len(dialog) >= 6:
        long_dialogs.append(dialog)

print(f"Found {len(long_dialogs)} dialogues with at least 6 turns.")
```

此工作流程对于后处理和分析大量生成的对话非常有用，以选择符合特定标准的对话。

5 结论

SDialog 提供了一个全面且可扩展的框架，用于合成对话的生成和分析。该工具包支持基于个性的角色扮演与 LLMs、多代理和串联对话、场景和数据集的集成、灵活的序列化和可视化，以及用于高级研究的自定义调度。SDialog 适用于构建对话数据集、评估对话模型，并支持在与对话 AI 相关的学术和工业研究环境中的实验。

Acknowledgments. This work was supported by EU Horizon 2020 project ELOQUENCE[3] (grant number 101070558).

References

1. Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y.T., Li, Y., Lundberg, S., et al.: Sparks of artificial general intelligence: Early experiments with GPT-4. arXiv preprint arXiv:2303.12712 (2023)
2. Burdisso, S., Madikeri, S., Motlicek, P.: Dialog2Flow: Pre-training soft-contrastive action-driven sentence embeddings for automatic dialog flow extraction. In: Al-Onaizan, Y., Bansal, M., Chen, Y.N. (eds.) Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing. pp. 5421–5440. Association for Computational Linguistics, Miami, Florida, USA (Nov 2024). https://doi.org/10.18653/v1/2024.emnlp-main.310, https://aclanthology.org/2024.emnlp-main.310/

[3] https://eloquenceai.eu/

3. Burdisso, S., Reyes-Ramírez, E., Villatoro-tello, E., Sánchez-Vega, F., Lopez Monroy, A., Motlicek, P.: DAIC-WOZ: On the validity of using the therapist's prompts in automatic depression detection from clinical interviews. In: Naumann, T., Ben Abacha, A., Bethard, S., Roberts, K., Bitterman, D. (eds.) Proceedings of the 6th Clinical Natural Language Processing Workshop. pp. 82–90. Association for Computational Linguistics, Mexico City, Mexico (Jun 2024). https://doi.org/10.18653/v1/2024.clinicalnlp-1.8, https://aclanthology.org/2024.clinicalnlp-1.8/
4. Caffaro, F., Rizzo, G.: Knowledge-enhanced conversational agents. Journal of Computer Science and Technology **39**(3), 585–609 (2024)
5. Callejas-Rodríguez, Á., Villatoro-Tello, E., Meza, I., Ramírez-de-la Rosa, G.: From dialogue corpora to dialogue systems: Generating a chatbot with teenager personality for preventing cyber-pedophilia. In: Text, Speech, and Dialogue: 19th International Conference, TSD 2016, Brno, Czech Republic, September 12-16, 2016, Proceedings 19. pp. 531–539. Springer (2016)
6. Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al.: Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168 (2021)
7. Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., Steinhardt, J.: Measuring massive multitask language understanding. Proceedings of the International Conference on Learning Representations (ICLR) (2021)
8. Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., Steinhardt, J.: Measuring mathematical problem solving with the math dataset. NeurIPS (2021)
9. Lu, P., Mishra, S., Xia, T., Qiu, L., Chang, K.W., Zhu, S.C., Tafjord, O., Clark, P., Kalyan, A.: Learn to explain: Multimodal reasoning via thought chains for science question answering. In: The 36th Conference on Neural Information Processing Systems (NeurIPS) (2022)
10. Melo, G., Alencar, P., Cowan, D.: Enhancing software development with context-aware conversational agents: A user study on developer interactions with chatbots. arXiv preprint arXiv:2505.08648 (2025)
11. Naik, V., Metallinou, A., Goel, R.: Context aware conversational understanding for intelligent agents with a screen. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32 (2018)
12. Ngai, E.W., Lee, M.C., Luo, M., Chan, P.S., Liang, T.: An intelligent knowledge-based chatbot for customer service. Electronic Commerce Research and Applications **50**, 101098 (2021)