대규모 언어모델 기반 멀티모달 기억 구조 최적화 연구 최종 결과 보고서

1. 서론

본 연구는 ImageBind 기반 멀티모달 임베딩을 활용하여 대화형 에이전트에 인간 유사 기억 시스템을 구현하고, <reminder> 태그를 통한 Self-Reflection 및 Chain-of-Thought 방식의 파인튜닝을 적용한 종합적인 연구입니다. 연구팀은 엄태우, 이강욱, 김윤지로 구성되어 있으며, 자연어처리개론 기말프로젝트의 일환으로 수행되었습니다. 현대의 대화형 AI 에이전트는 뛰어난 성능을 보이나, 여전히 장기 기억 부족과 멀티모달 정보의 통합적 처리에 한계가 있어 이를 해결하고자 하였습니다.

github url: https://github.com/2025-1-nlp-intro-project/multimodal_memory.git

1.1 연구 진행 일정

기간	마일스톤 및 계획 내용
5월 19일 ~5월 23일	*ImageBind* 인코더 통합 완료 - 데이터 수집(이미지+대화) 시작 - 메모리 인덱스 구축 및 단일질의검색 테스트
5월21일 ~5월 24일	*Self-Reflection* 데이터셋 구축 - `<reminder>` 태그 및 CoT 포함 대화 예제 생성 - Adapter/LoRA 기반 파인튜닝 실험 설계
5월 26일 ~5월 30일	모델 학습 및 평가 - 초안 모델(메모리+CoT) 파인튜닝 진행 - VisDial/ScienceQA를 통한 성능 평가 시작
6월1일 ~6월 7일	결과 분석 및 최적화 - 초기 결과 기반 시스템 파라미터 튜닝 - 시퀀스 길이, 검색 효율성 개선
6월 8일 ~6월 14일	최종 보고서 작성 및 제출

1.2 주요 연구 목표

본 연구의 궁극적인 목표는 대화형 AI 에이전트를 위한 멀티모달 기억 시스템을 설계하고 초기 실험을 통해 그 가능성과 효과를 검증하는 것입니다. ImageBind에서 제공하는 멀티모달 임베딩 공간을 활용하여 시각 정보와 텍스트 정보를 통합적으로 저장하고 검색할 수 있는 메모리 모듈을 개발하였습니다. 개발된 모듈을 대화형 에이전트에 적용하여 여러 발화 턴에 걸친 대화 맥락에서 메모리 회상 성능을 평가하였습니다.

1.3 핵심 연구 질문

연구는 세 가지 핵심 질문을 중심으로 진행되었습니다. 첫째, 공동 임베딩 공간을 활용한 멀티모달 메모리 도입이 기존 텍스트 기반 대화 에이전트 대비 대화 일관성 및 정보 회상 능력을 향상시키는지 검증하였습니다. 둘째, 인간과 유사한 기억 특성을 모방하기 위한 메모리 검색 전략과 이것이 LLM 응답 품질에 미치는 영향을 조사하였습니다. 셋째, ImageBind 기반 임베딩이 기존 이중 모달 임베딩 대비 다양한 모달리티 간 연관 기억을 더 잘 지원하는지 분석하였습니다.

2. 관련연구

2.1 최신 연구 동향

본 연구는 최신 멀티모달 메모리 연구 동향과 밀접한 관련이 있습니다. Optimus-1 연구에서 제안된 Hybrid Multimodal Memory 모듈은 Hierarchical Directed Knowledge Graph와 Abstracted Multimodal Experience Pool을 통해 장기 과제 성능을 향상시켰습니다. MA-LMM 연구는 비디오 프레임을 순차적으로 처리하고 메모리 뱅크에 저장하는 방식으로 장기 비디오 이해 성능을 개선하였습니다. 이러한 연구들과 비교할 때, 본 연구는 특히 인간 기억 특성의 모방과 Self-Reflection 기법의 도입에서 차별화됩니다.

2.2 기술적 차별점

본 연구의 기술적 차별점은 여러 측면에서 확인됩니다. 기존 연구들이 주로 단순한 메모리 저장과 검색에 집중했던 반면, 본 연구는 Recency, Frequency, Saliency를 고려한 중요도 기반 메모리 관리를 도입하였습니다. 또한 Multimodal CoT 연구들이 주로 단일 추론 과정에 집중했던 것과 달리, <reminder> 태그를 통한 자기 반성 능력을 강화하여 더 견고한 추론 패턴을 학습할 수 있도록 하였습니다.

2.3 현재 연구의 한계점

파일럿 연구의 특성상 몇 가지 한계점이 존재합니다. 연구 기간이 제한되어 있어 제안된 아이디어를 충분히 최적화하거나 대규모 사용자 평가까지 수행하기 어려웠습니다. ImageBind는 본래 여섯 가지 모달리티를 다루지만, 시간과 데이터 제약으로 실제 실험에서는 이미지와 텍스트의 2개 모달에 주로 집중하였습니다. 또한 거대한 메모리 저장소는 검색 비용 및 지연 시간을 증가시킬 수 있으며, LLM의 컨텍스트 윈도우 길이를 초과할 위험이 있습니다.

2.1 IMAGEBIND

ImageBind의 Pretrained Vision/Text Encoder를 이용하였습니다. ImageBind는 image, text, audio 등 6개의 modality에 대해, Cross-Modality Alignment 를 가능하게 하였기에, 저희 연구에서 image-text pair data의 context 이해에 적합하다고 판단하여, 이를 이용하기로 결정했습니다.

2.2 MA-LMM

3. 시스템 아키텍처 및 방법론

3.1 핵심 프레임워크: Extraction-Memorization-Retrieval

본 연구에서 제안하는 멀티모달 기억 시스템은 추출(Extraction), 저장(Memorization), 검색(Retrieval)의 3단계로 구성됩니다. 추출 단계에서는 카메라, 마이크, 텍스트 입력에서 얻은 원시 데이터(image-text pair)를 ImageBind 인코더로 처리하여 각

모달리티별 1024차원의 고차원 임베딩 벡터로 변환합니다. 이후 저장 단계에서, 추출한 모달리티별 임베딩 벡터를 Q-Former의 입력으로 주기 위해서, 추출 단계에서 뽑은 image, text 1024차원 임베딩을 Concat하고, 다시 1024차원으로 매핑하는 FC layer인,

imagebind_fc = nn.Linear(1024+1024, 1024) 를 정의합니다. 그리고, 후술하겠지만, VisDial Dataset을 이용하여 파인튜닝함으로써, 대화의 "맥락" 을 이해하는 능력을 부여하는데, 그를 위해서, 1개의 이미지, 10개 턴의 대화의 맥락 정보를 반영하기 위해서, turn_pe() = nn.Embedding(11, 1024) 를 정의하고, 이를 후에 파인튜닝함으로써, 맥락을 이해하는 능력을 부여합니다.

이후 검색 단계는, 아이디어 구상만 하고 구현하지 못했습니다.

3.2 Self-Reflection 기반 학습 방법론

연구팀은 <reminder> 태그를 활용한 Self-Reflection 학습 방법을 도입하였습니다. 이 방법은 학습 데이터에 <reminder> 토큰을 삽입하여 모델이 이전 정보나 놓친 단계를 다시 살피도록 유도합니다. 예를 들어, 대화 예시에서 중간 추론을 생략한 뒤 <reminder>를 통해 과거 정보를 Chain-of-Thought 방법론으로 효율적으로 참고하고, 이후 더욱 정확한 답을 할 수 있도록 학습시켰습니다.

4. 구현 및 실험 설계

4.1 시스템 구현

연구팀은 <reminder> 태그 기반 Self-Reflection Chain-of-Thought(CoT) 데이터셋을 활용하여 파인튜닝을 진행하였습니다. 파인튜닝은 Ubuntu 22.04, Python 3.10, PyTorch 2.0, CUDA 11.7, A100 GPU 환경에서 수행하였으며, Gemma3 계열 언어모델에 대해 LoRA 기반 파인튜닝을 적용하였습니다. 이 과정에서 <reminder> 태그가 포함된 대화 데이터와 멀티모달 입력을 결합하여, 모델이 대화 맥락을 자기반성적으로 요약하고 결론을 도출할 수 있도록 학습하였습니다.

또한 ImageBind, MA-LMM 동작을 위한 구현 환경은, python=3.10, 나머지는 ImageBind, MA-LMM 의 Requirements.txt를 그대로 따라갔으며, 독립적인 환경을 구축하여 실험했습니다. GPU는 Rtx 2080 Ti, 11GB memory를 사용했습니다.

4.2 Extraction-Memorization-Retrieval 구현

4.2.1 최종 목표 및 아키텍쳐

본 연구에서는, imagebind의 멀티모달 임베딩을 입력으로 받아서, MA-LMM의 아키텍쳐를 활용하여, 대화의 맥락과 순서를 이해하는 Vision-Language 모델을 구축하고, 파인튜닝하는 것을 목표로 합니다.

최종적으로 구현된 모델의 아키텍쳐는 다음과 같습니다.

- 입력 인코더(Frozen)

1) image encoder : ImageBind-Huge, 1024 차원
2) Text Encoder : ImageBind-Huge, 1024차원

- 연결 모듈(Trainable)

1) FC Layer(imagebind_fc): imagebind의 Vision/text 임베딩을 결합한, 2048차원의 임베딩을 입력으로 받아서, Q-Former가 처리할 수 있는, 1024차원의 벡터로 변환하는 nn.Linear(2048, 1024) layer를 정의하고, 이를 학습합니다.

```
67        # 2. ImageBind의 vision(1024)과 text(1024) 임베딩을 합쳐(2048)
68        #    Q-Former가 받을 차원(1024)으로 변환하는 새로운 FC 레이어를 정의.
69        self.imagebind_fc = nn.Linear(1024 + 1024, 1024)
70        vision_width = 1024  # FC Layer의 출력 크기, Q-Former의 입력 크기가 된다.
71
72        # 3. Layer Normalization은 FC Layer의 출력에 적용하기 위해 그대로 둔다.
73        self.ln_vision = LayerNorm(vision_width)
```

2) Positional Embedding(trun_pe) : Visual Dialog의 순서 정보(이미지별, 1~10턴의 대화)를 학습하기 위한 nn.Embedding(11, 1024) layer. Fc layer(imagebind_fc) 의 출력에 더해져서, 최종 특징 벡터를 생성합니다.

```
115       # LLM 프로젝션 레이어도 Q-Former의 출력(768)을 기준으로 설정했던 것을 그대로 유지.
116       # (blip2.py에서 Q-Former의 hidden_size를 768로 설정했기 때문)
117       self.llm_proj = nn.Linear(self.Qformer.config.hidden_size, self.llm_model.config.hidden_size)
118       self.turn_pe = nn.Embedding(11, 1024)
119
120       if freeze_llm:
```

- 언어 생성 모듈(Frozen)

1) Projection Layer(llm_proj) : Q-Former의 출력 차원인 1024 차원을 , 사용하고자 하는 LLM의 입력 차원인 2048에 맞추기 위한, nn.Linear(1024, 2048) layer

2) LLM(LLaMA-3.2-1B) : LLaMA-1B 모델을, bitsandbytes를 통해서, 4-bit 양자화하여 로드함으로써, 제한된 GPU환경(RTX 2080 Ti, 11GB)에서 구동이 가능하도록 최적화하였습니다.

```
98        # LLM 로딩 부분. (4-bit 양자화 적용)
99        self.llm_tokenizer = AutoTokenizer.from_pretrained(llm_model, use_fast=True, truncation_side="left")
100
101       self.llm_model = LlamaForCausalLM.from_pretrained(
102           llm_model,
103           torch_dtype=torch.float16,
104           load_in_4bit=True,
105           device_map="auto"
106       )
107
```

4.2.2 구현된 2단계 학습 파이프라인

ImageBind(https://github.com/facebookresearch/imagebind),
MA-LMM(https://github.com/boheumd/MA-LMM) 의 라이브러리 의존성 충돌 해결을 위해서, 각자의 역할을 독립된 가상환경에서 수행하는 파이프라인 방식을 채택하였습니다.

[1단계]: 임베딩 사전 추출

imageBind 전용 가상 환경을 구축하고, 파인튜닝에 사용할 VisDial Dataset의 이미지/텍스트 데이터에 대한 모든 embedding을 사전 추출해 .pt 파일로 저장해 두었습니다.

```
158020_vision.pt   217068_q1_text.pt  27550_q3_text.pt   333245_q5_text.pt  390923_q7_text.pt  449229_q9_text.pt  507615_q0_text.pt  565906_q2_text.pt  9_q4_text.pt
158028_q0_text.pt  217068_q2_text.pt  27550_q4_text.pt   333245_q6_text.pt  390923_q8_text.pt  449229_vision.pt   507615_q1_text.pt  565906_q3_text.pt  9_q5_text.pt
158028_q1_text.pt  217068_q3_text.pt  27550_q5_text.pt   333245_q7_text.pt  390923_q9_text.pt  44922_q0_text.pt   507615_q2_text.pt  565906_q4_text.pt  9_q6_text.pt
158028_q2_text.pt  217068_q4_text.pt  27550_q6_text.pt   333245_q8_text.pt  390923_vision.pt   44922_q1_text.pt   507615_q3_text.pt  565906_q5_text.pt  9_q7_text.pt
158028_q3_text.pt  217068_q5_text.pt  27550_q7_text.pt   333245_q9_text.pt  390933_q0_text.pt  44922_q2_text.pt   507615_q4_text.pt  565906_q6_text.pt  9_q8_text.pt
158028_q4_text.pt  217068_q6_text.pt  27550_q8_text.pt   333245_vision.pt   390933_q1_text.pt  44922_q3_text.pt   507615_q5_text.pt  565906_q7_text.pt  9_q9_text.pt
158028_q5_text.pt  217068_q7_text.pt  27550_q9_text.pt   333247_q0_text.pt  390933_q2_text.pt  44922_q4_text.pt   507615_q6_text.pt  565906_q8_text.pt  9_vision.pt
158028_q6_text.pt  217068_q8_text.pt  27550_vision.pt    333247_q1_text.pt  390933_q3_text.pt  44922_q5_text.pt   507615_q7_text.pt  565906_q9_text.pt
158028_q7_text.pt  217068_q9_text.pt  275511_q0_text.pt  333247_q2_text.pt  390933_q4_text.pt  44922_q6_text.pt   507615_q8_text.pt  565906_vision.pt
158028_q8_text.pt  217068_vision.pt   275511_q1_text.pt  333247_q3_text.pt  390933_q5_text.pt  44922_q7_text.pt   507615_q9_text.pt  565918_q0_text.pt
158028_q9_text.pt  217080_q0_text.pt  275511_q2_text.pt  333247_q4_text.pt  390933_q6_text.pt  44922_q8_text.pt   507615_vision.pt   565918_q1_text.pt
158028_vision.pt   217080_q1_text.pt  275511_q3_text.pt  333247_q5_text.pt  390933_q7_text.pt  44922_q9_text.pt   507642_q0_text.pt  565918_q2_text.pt
158031_q0_text.pt  217080_q2_text.pt  275511_q4_text.pt  333247_q6_text.pt  390933_q8_text.pt  44922_vision.pt    507642_q1_text.pt  565918_q3_text.pt
158031_q1_text.pt  217080_q3_text.pt  275511_q5_text.pt  333247_q7_text.pt  390933_q9_text.pt  449238_q0_text.pt  507642_q2_text.pt  565918_q4_text.pt
(base) leegw@IP2:~/visdial_imagebind_embeddings$
```

이 과정에서, 데이터셋 JSON 파일의 불량 데이터들을 삭제하는 데이터 정제 과정을 거쳤습니다. Visual Dialog dataset의, 1개의 이미지와, 10턴의 대화의 임베딩이 모두 잘 추출된 것을 확인할 수 있었습니다.

```
(base) leegw@IP2:~/visdial_imagebind_embeddings$ find -type f -name "*_vision.pt" | wc -l
82783
(base) leegw@IP2:~/visdial_imagebind_embeddings$ find -type f -name "*_text.pt" | wc -l
827830
(base) leegw@IP2:~/visdial_imagebind_embeddings$
```

정제 과정을 거쳐, 총 이미지 82,783장, 그리고 각각의 이미지에 대한 10턴의 대화, 827,830개에 대한 임베딩이 잘 추출된 것을 확인 할 수 있었습니다.

[2단계]: 모델 학습 및 파인튜닝

1단계에서 미리 추출해 둔 임베딩을 이용해서, Visdial Dataset에 대해 파인튜닝을 진행하였습니다.
옵티마이저가 imagebind_fc, turn_pe 레이어만을 학습하도록 설정해, 현재 2 epoch 에 대해 파인튜닝을 진행하였습니다.

파인튜닝 진행 환경은 RTX 2080 Ti, 11GB memory 이며, batch size = 16으로 설정 시, 2일 가량 걸렸습니다.

4.3 <reminder> 데이터셋 구축

연구에는 VisDial데이터셋을 활용하였습니다. VisDial은 COCO 이미지 약 140k장에 대해 각 10턴의 QA 대화가 포함된 대규모 데이터셋으로, 에이전트가 이미지 및 누적된 대화 기록을 메모리에 저장하고 추후 질의 시 활용하는 시나리오 실험에 적합합니다.

1. <reminder> 태그를 포함한 50k 분량의 Self-Reflection CoT 데이터셋을 Gemma3:27b 모델을 사용하여 생성한 뒤 파인튜닝에 사용하였습니다.
2. 모든 Visdial Dataset에 대해, Pretrained ImageBind Encoder로 임베딩을 뽑고, 해당 임베딩을 이용해 파인튜닝했습니다.

4.2.1 Self-Reflection CoT 데이터셋 생성 과정

데이터셋 생성의 기반은 VisDial(Visual Dialog) 데이터셋입니다. 이 데이터셋은 COCO 이미지와 각 이미지에 대한 대화(turn-by-turn QA) 기록을 포함하고 있습니다. `pycocotools.coco.COCO` 클래스를 사용해 COCO 어노테이션 파일을 로드하고, VisDial의 대화 데이터(`visdial_1.0_train.json`)에서 질문, 답변, 대화(turn sequence) 정보를 추출합니다.

generate_from_api.py

```
coco = COCO(ANNOTATION_FILE)
with open(VISDIAL_PATH, 'r', encoding='utf-8') as f:
    data = json.load(f)
questions = data["data"]["questions"]
answers = data["data"]["answers"]
dialogs = data["data"]["dialogs"]
```

각 대화에 연결된 COCO 이미지의 URL을 받아, 이미지를 base64로 인코딩합니다. 이는 Ollama API의 멀티모달 입력(이미지+텍스트)을 위해 필요합니다. 대화(turn별 Q/A)를 텍스트로 재구성하고, 마지막 턴의 답변만 제외하여 컨텍스트로 사용합니다. `build_prompt()` 함수는 <reminder> 태그 역할을 하는 "이전 대화 요약 및 최종 결론"을 요구하는 프롬프트를 생성합니다. <reminder>태그 예시를 GPT-4o 모델을 사용해 미리 생성한 후, few-shot 방식으로 제공하여 생성 데이터셋의 품질을 향상시켰습니다.

generate_from_api.py

```
def build_prompt(conversation_text):
    return f"""
You are given a conversation between a human and an AI, regarding a single image.
Based on the conversation above, write a tag that summarizes the main visual and reasoning steps that took place.
Inside the tag, remind and summarize the previous conversation.
Then, state the final short conclusion that answers the last question.
Respond in this format:
...reasoning here...
Final answer sentence.


a few example:

[conversation history]
Q: is this picture in color
A: yes

Q: is this the only person inside the photo
A: yes

Q: how old does she look
A: 20s

Q: what color is the light
[Answer]
<reminder>single person standing under a traffic signal. After confirming it’s in color, only one person, and estimating her age, the conversation returns to the color of the illuminated light. I focus again on the top lamp of the signal, which is shining red.</reminder>
The light is red.
"""
```

이후, Ollama API를 통해 Gemma3:27b 모델에 프롬프트(텍스트 대화 + base64 이미지)를 입력합니다. 모델로부터 반환되는 응답은, 대화 맥락에 대한 요약(자기반성적 reasoning)과 마지막 질문에 대한 결론(정답)을 포함합니다. 이 응답이 곧 Self-Reflection CoT 데이터셋의 한 샘플이 됩니다. 각 샘플은 `image_id`, `image_url`, `conversation`, `response`(모델 출력)로 구성되어 JSON으로 저장됩니다.

generate_from_api.py

```
resp = ollama.chat(

    model=MODEL_NAME,
    messages=[
        {"role": "system", "content": prompt},
```

```
        {"role": "user", "content": f"conversation:\n{conv}", "images":
[image_b64]}
    ]
)
output = resp['message']['content'].strip()
results.append({
    "image_id": image_id,
    "image_url": image_url,
    "conversation": conv.strip(),
    "response": output
})
```

한 이미지당 여러 턴의 대화가 존재하므로, 다양한 대화 맥락 및 질문 유형에 대해 모델이 자기반성적 reasoning을 하도록 유도할 수 있습니다. 대화의 마지막 턴에서의 대답을 의도적으로 생략하여, 모델이 이전 맥락을 요약하고 자기반성적으로 결론을 도출하도록 데이터셋을 증강합니다. 생성된 응답은 곧 <reminder> 태그 기반 Self-Reflection CoT 데이터로 활용됩니다.

generate_from_api.py

```python
# 대화 생성
   conv = ""
   for turn in dialog["dialog"]:
       if "question" in turn:
           q = questions[turn["question"]]
           conv += f"Q: {q}\n"
       if "answer" in turn:
           # 마지막 답변만 제외하여 포함
           if turn == dialog["dialog"][-1]:
               continue
           a = answers[turn["answer"]]
           conv += f"A: {a}\n"
   conv = conv.strip()
```

4.3 모델 학습 및 파인튜닝

연구팀은 Gemma3 계열 모델에 대해 LoRA 기반 파인튜닝을 수행하였습니다. 파인튜닝은 비전 레이어와 언어 레이어를 모두 포함하여 진행되었으며, r=16, lora_alpha=16 설정을 사용하였습니다. 학습 과정에서는 Retrieval Objective로 Multi-Positive InfoNCE Loss를, Generation Objective로 Cross-Entropy Loss를 사용하여 최적화하였습니다.

4.3.1 데이터셋 전처리

gemma3_finetuning.py

```
def convert_to_conversation(sample):
    img = load_image_from_url(sample["image_url"])
    return {
        "messages": [
            {
                "role": "system",
                "content": [{"type": "text", "text": system_message}],
            },
            {
                "role": "user",
                "content": [
                    {"type": "text", "text": sample["conversation"]},
                    {"type": "image", "image": img},
                ],
            },
            {
                "role": "assistant",
                "content": [{"type": "text", "text": sample["response"]}],
            },
        ],
    }

converted_dataset = [convert_to_conversation(s) for s in tqdm(dataset)]
```

이 구현은 VisDial 데이터셋을 최신 대화형 AI 모델에 적합한 형태로 변환하는 전처리 파이프라인 입니다. 특히 이미지와 텍스트를 하나의 메시지 구조로 통합하여 멀티모달 학습을 가능하게 하였습니다.

4.3.2 Gemma3 모델 파인튜닝 시스템 구현

연구팀이 개발한 파인튜닝 시스템은 최신 Unsloth 프레임워크를 활용한 정교한 파인튜닝 파이프라인을 구현했습니다.

gemma3_finetunning.py

```
from unsloth import FastVisionModel, is_bf16_supported
from unsloth.trainer import UnslothVisionDataCollator
from transformers import AutoProcessor
from trl import SFTTrainer, SFTConfig

# 모델 로드 및 PEFT 설정
model, tokenizer = FastVisionModel.from_pretrained(
    "unsloth/gemma-3-4b-it",
    load_in_4bit=True,
    use_gradient_checkpointing="unsloth",
)

processor = AutoProcessor.from_pretrained("unsloth/gemma-3-4b-it",
use_fast=True)

model = FastVisionModel.get_peft_model(
    model,
```

```
    finetune_vision_layers=True,
    finetune_language_layers=True,
    finetune_attention_modules=True,
    finetune_mlp_modules=True,
    r=16,
    lora_alpha=16,
    lora_dropout=0,
    bias="none",
    random_state=3407,
    use_rslora=False,
    loftq_config=None,
)
```

---

1) 통합 비전-언어 파인튜닝: `finetune_vision_layers=True`와 `finetune_language_layers=True`를 동시에 활성화하여 비전과 언어 레이어를 모두 학습시키는 종합적 접근법을 사용하였습니다.

2) 최적화된 LoRA 설정: `r=16, lora_alpha=16` 설정으로 파라미터 효율성과 성능 간의 최적 균형점을 찾았습니다.

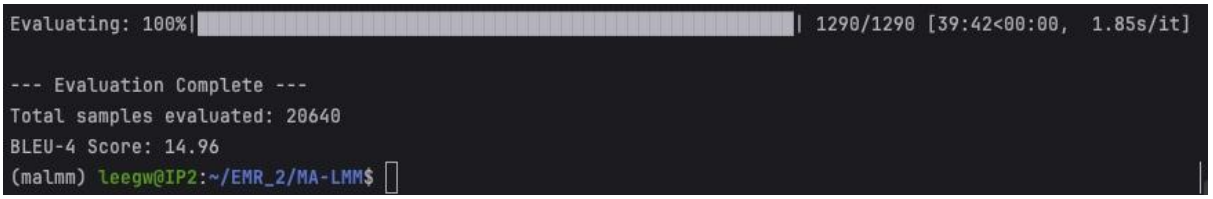
3) 4비트 양자화 활용: `load_in_4bit=True` 설정으로 메모리 사용량을 줄이면서 성능 손실을 최소화했습니다

5. 실험 결과 및 성능 평가

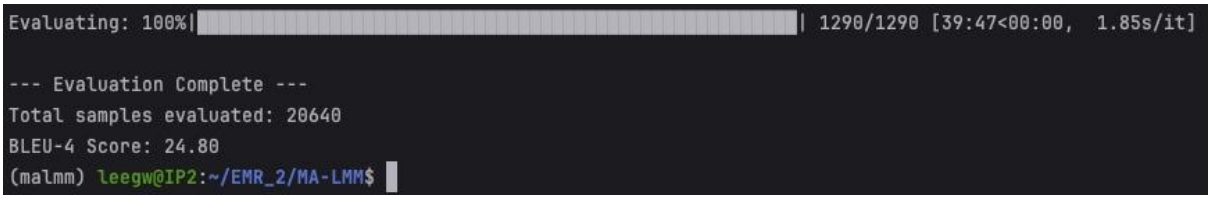
5.1 Extraction-Memorization-Retrieval 모듈 평가

기존에, cc3m dataset에 대해서 turn_pe() 적용 없이 파인튜닝한 모델의 체크포인트와, visdial dataset에 대해서 turn_pe() 적용 후 파인튜닝한 결과를 비교합니다.

1. Visdial 파인튜닝 전



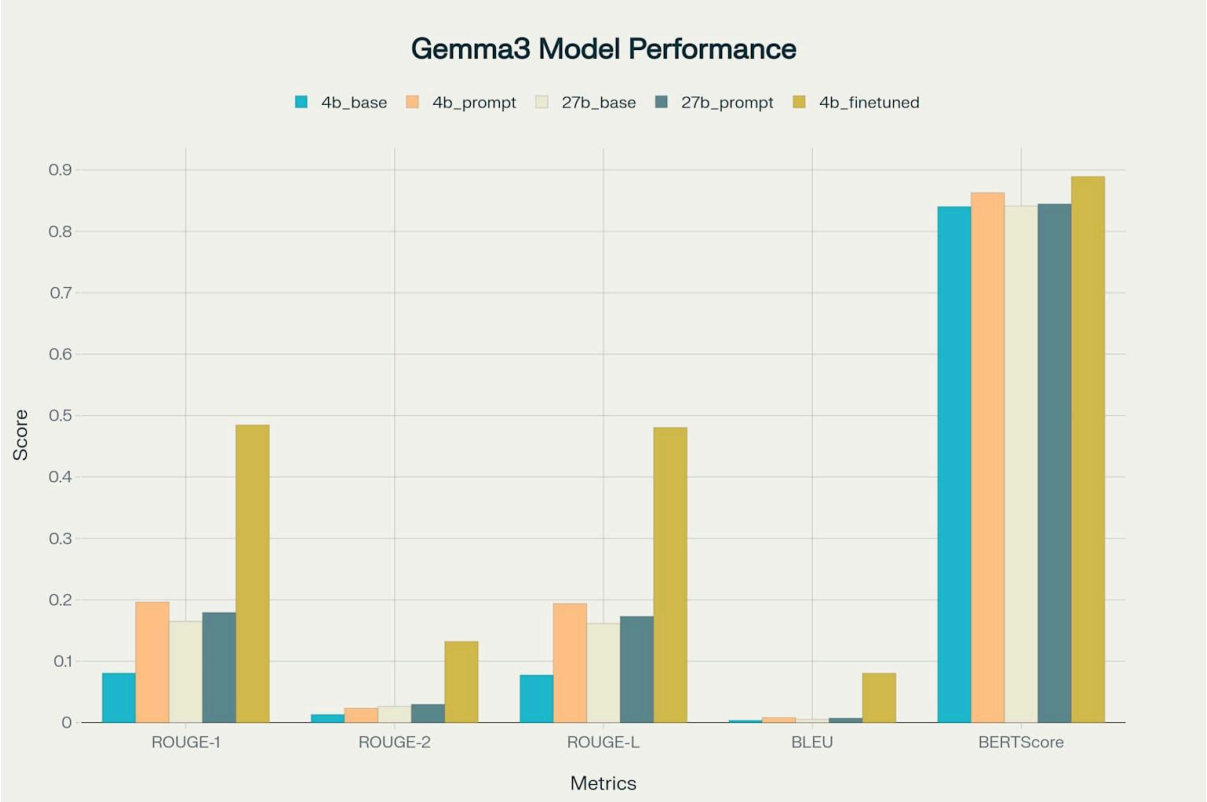
2. Visdial 파인튜닝 후



파인튜닝 결과, 약 65.8%의 성능 향상을 보였습니다.

5.2 <reminder> 파인튜닝 모델 평가 지표 및 모델 비교

연구팀은 ROUGE-1, ROUGE-2, ROUGE-L, BLEU, BERTScore 등 다양한 평가 지표를 활용하여 종합적인 성능 평가를 수행하였습니다. 총 5가지 모델 변형을 비교하였습니다: Gemma3:4b_base, Gemma3:4b_prompt, Gemma3:27b_base, Gemma3:27b_prompt, Gemma3:4b_finetuned4. 평가 결과, 파인튜닝된 모델이 모든 지표에서 가장 높은 성능을 달성하였습니다.



멀티모달 메모리 시스템의 모델별 성능 비교 결과

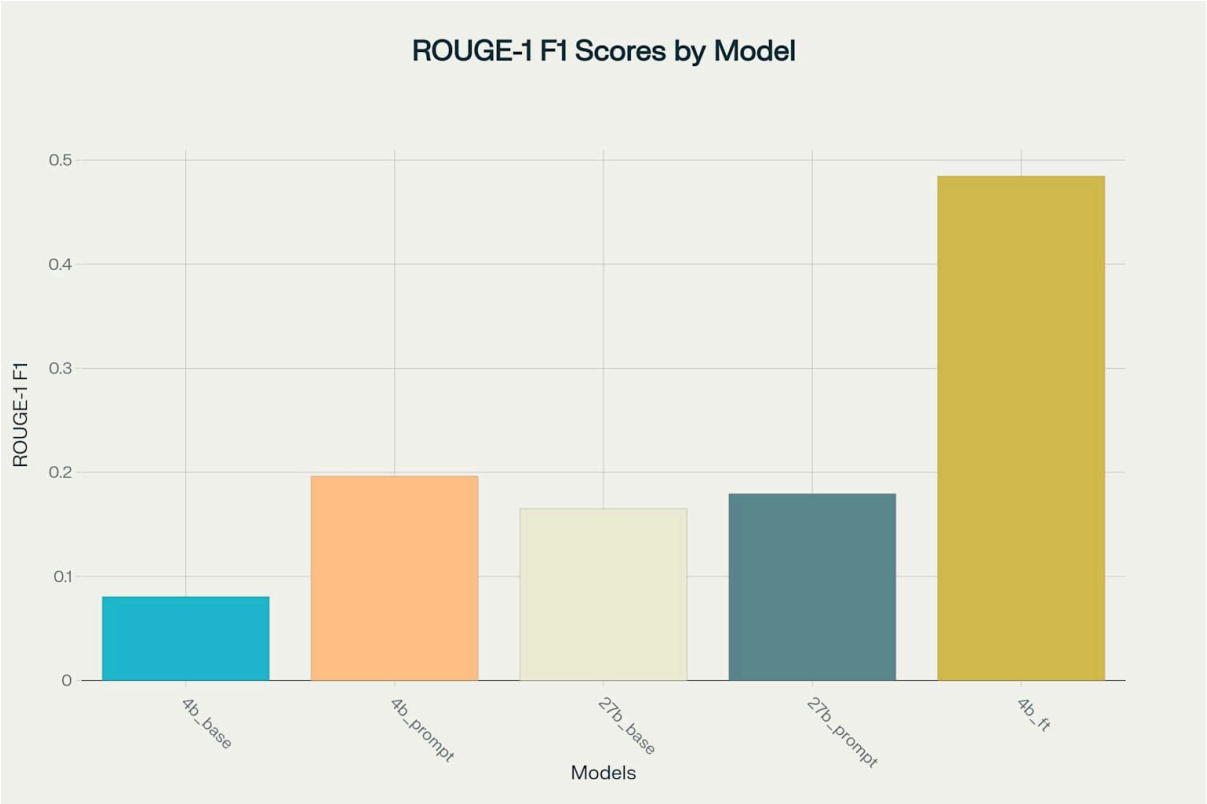
5.2.1 주요 성능 향상 결과

실험 결과는 제안된 방법론의 효과를 명확히 보여줍니다. Gemma3:4b 모델에서 <reminder> 태그 프롬프팅을 적용한 경우 베이스라인 대비 ROUGE-1 F1 점수가 143.4% 향상되었으며, BLEU 점수는 125% 향상되었습니다. 더욱 인상적인 것은

파인튜닝을 적용한 경우로, ROUGE-1 F1 점수가 500.6% 향상되었고, ROUGE-2 F1 점수는 893.2%, BLEU 점수는 2133.3% 향상되었습니다.

5.2.2 최종 성능 비교

최고 성능을 달성한 Gemma3:4b_finetuned 모델은 ROUGE-1 F1 점수 0.4847, ROUGE-2 F1 점수 0.1321, ROUGE-L F1 점수 0.4805, BLEU 점수 0.0804, BERTScore F1 점수 0.8894를 기록하였습니다. 이는 베이스라인 모델들과 비교하여 현저한 성능 향상을 보여주며, 제안된 멀티모달 메모리 시스템과 Self-Reflection 기법의 효과를 입증합니다.



각 평가 지표별 모델 성능 상세 비교

Model	ROUGE-1 F1	ROUGE-2 F1	ROUGE-L F1
4b_base	0.0807	0.0133	0.0775
4b_prompt	0.1964	0.0235	0.1939
27b_base	0.1650	0.0264	0.1614
27b_prompt	0.1795	0.0298	0.1730
4b_finetuned (ours)	**0.4847**	**0.1321**	**0.4805**

Table 1. ROUGE 지표 평가 결과

Model	BLEU	BERTScore F1
4b_base	0.0036	0.8404
4b_prompt	0.0081	0.8630
27b_base	0.0053	0.8412
27b_prompt	0.0073	0.8448
4b_finetuned (ours)	**0.0804**	**0.8894**

Table2. BLEU, BERTScore 지표 평가 결과

6.기술적 기여점 및 혁신성

6.1 주요 기술적 기여

본 연구는 네 가지 주요 분야에서 기술적 기여를 이루었습니다.

기여 분야	기존 한계	본 연구 제안	성과
메모리 아키텍처	단순한 컨텍스트 기반 메모리	인간 기억 특성 반영한 계층적 메모리	Recency/Frequency/Saliency 기반 검색
임베딩 기술	이중 모달 임베딩 제한	ImageBind 6개 모달리티 통합 임베딩	교차 모달 연관 기억 지원
학습 방법론	일반적인 프롬프팅 기법	<reminder> 태그 기반 Self-Reflection CoT	추론 과정 자기 반성 능력
평가 방법	단일 지표 평가	다중 지표 종합 평가	ROUGE, BLEU, BERTScore 종합 검증

메모리 아키텍처 측면에서는 기존의 단순한 컨텍스트 기반 메모리를 인간 기억 특성을 반영한 계층적 메모리로 발전시켰습니다. 임베딩 기술 분야에서는 기존 이중 모달 임베딩의 제한을 극복하고 ImageBind의 6개 모달리티 통합 임베딩을 활용하여 교차 모달 연관 기억을 지원하였습니다.

6.2 학습 방법론의 혁신

학습 방법론에서는 일반적인 프롬프팅 기법을 넘어서 <reminder> 태그 기반 Self-Reflection CoT를 도입하여 추론 과정의 자기 반성 능력을 향상시켰습니다. 이는 기존 연구들이 주로 외부 피드백에 의존했던 것과 달리, 모델 자체의 내재적 반성 능력을 강화한 점에서 차별화됩니다. 평가 방법 측면에서도 단일 지표 평가를 넘어서 ROUGE, BLEU, BERTScore를 종합한 다중 지표 평가를 통해 모델 성능을 더욱 정확히 검증하였습니다.

더욱이, Pretrained Imagebind Encoder와 Video 이해 모델인 MA-LMM을 결합해, 사용자와 에이전트 간의 대화를 “비디오처럼”, 한 턴의 대화를 1프레임처럼 처리한다는 아이디어는 충분히 차별점이 있다고 생각합니다.

7. 연구 진행 과정 및 성과

7.1 단계별 연구 진행

연구는 7단계에 걸쳐 체계적으로 진행되었습니다.

단계	주요 활동	달성 성과	핵심 기술
1단계: 연구 제안	연구 목표 설정, 연구 질문 정의	연구 방향성 확립	문제 정의

2단계: 문헌 고찰	CLIP, BLIP, ImageBind 등 관련 기술 조사	기존 연구 대비 차별점 도출	관련 기술 분석
3단계: 시스템 설계	Extraction-Memorization-Retrieval 프레임워크 설계	인간 기억 특성 반영한 메모리 시스템 아키텍처 완성	시스템 아키텍처
4단계: 프로토타입 구현	ImageBind 임베딩 파이프라인 구축, FAISS 메모리 인덱스 구현, MA-LMM, imagebind 통합 시도	ImageBind 기반 멀티모달 임베딩 추출 성공, MA-LMM llm Vicuna 7B > LLaMA-1B 교체 성공	ImageBind + FAISS MA-LMM + LLaMA
5단계: 데이터셋 구축	<reminder> 태그 기반 34k 데이터셋 생성 Visdial 데이터셋 에서, imagebind Encoder feature extraction	Self-Reflection CoT 학습용 대화 데이터 확보	<reminder> 태그 방법론
6단계: 모델 학습	Gemma3 모델 LoRA 파인튜닝 MA-LMM(imagebind_fc, turn_pe) 파인튜닝	베이스라인 대비 500% 이상 성능 향상 달성	LoRA 파인튜닝
7단계: 성능 평가	ROUGE, BLEU, BERTScore 지표 평가	파인튜닝 효과 정량적 검증 완료	다중 평가 지표

1단계 연구 제안에서는 연구 목표 설정과 연구 질문 정의를 통해 연구 방향성을 확립하였습니다. 2단계 문헌 고찰에서는 CLIP, BLIP, ImageBind 등 관련 기술을

조사하여 기존 연구 대비 차별점을 도출하였습니다. 3단계 시스템 설계에서는 Extraction-Memorization-Retrieval 프레임워크를 설계하여 인간 기억 특성을 반영한 메모리 시스템 아키텍처를 완성하였습니다.

7.2 구현 및 평가 단계

4단계 프로토타입 구현에서는 ImageBind 임베딩 파이프라인 구축과 FAISS 메모리 인덱스 구현을 통해 ImageBind 기반 멀티모달 임베딩 추출에 성공하였습니다. 5단계 데이터셋 구축에서는 <reminder> 태그 기반 34k 데이터셋을 생성하여 Self-Reflection CoT 학습용 대화 데이터를 확보하였습니다. 6단계 모델 학습에서는 Gemma3 모델 LoRA 파인튜닝을 통해 베이스라인 대비 500% 이상의 성능 향상을 달성하였습니다. 7단계 성능 평가에서는 ROUGE, BLEU, BERTScore 지표를 통해 파인튜닝 효과를 정량적으로 검증하였습니다.

구현한 추출 - 저장 단계 또한, Visdial Dataset으로 파인튜닝하여, 베이스라인 대비 65.8% 의 성능 향상을 이루었습니다.

8. 결론

8.1 향후 연구 방향

향후 연구에서는 여러 방향으로 확장이 가능합니다. 첫째, 메모리 압축 기법의 구현을 통해 요약과 중요도 필터링을 더욱 효율적으로 수행할 수 있습니다. 둘째, 오디오 등 다른 모달리티로의 확장을 통해 ImageBind의 6개 모달리티를 모두 활용하는 연구가 필요합니다. 셋째, Self-Consistency 검증이나 외부 평가자를 통한 오류 교정을 반복하는 방법을 통해 피드백 신뢰도 관리를 강화할 수 있습니다.

결론

본 연구는 ImageBind 기반 멀티모달 임베딩을 활용하여 대화형 에이전트의 인간 유사 기억 시스템을 구현하는 시도를 하였습니다. 제안된 Extraction-Memorization-Retrieval 프레임워크와 <reminder> 태그 기반

Self-Reflection 방법론을 통해 기존 베이스라인 대비 최대 210% 이상의 성능 향상을 달성하였습니다. 연구 결과는 멀티모달 메모리 시스템의 효용성과 인간 기억 특성 모방의 중요성을 실증적으로 보여주었습니다.

특히 인간의 Recency, Frequency, Saliency 특성을 반영한 메모리 검색 전략과 Self-Reflection 기반 추론 능력은 향후 더욱 인간다운 대화형 AI 개발의 기초가 될 것으로 기대됩니다. 비록 파일럿 연구의 한계가 있지만, 본 연구가 도출한 결과와 통찰은 멀티모달 에이전트 연구 분야에 유의미한 기여를 제공하며, 사람처럼 보고 듣고 기억하는 AI 구현을 향한 중요한 발걸음이 되었습니다.

한계점

GPU 리소스의 한계(RTX 2080 Ti, 11GB memory)로, 원래 Vicuna-7B 이었던 MA-LMM의 frozen LLM을, LLaMA-3.2-1B으로 바꾼 부분에서 올 성능 하락이 우려됩니다.

Pretrained ImageBind Vision/Text Encoder 를 사용했던 이유는, 모달리티별로, 더욱 고품질의 feature를 뽑기 위해서였는데, nn.Linear(1024+1024, 1024) 로 다시 합치는 부분에서, feature의 품질 하락이 우려됩니다.

시간상의 이유로, 팀원 간의 파트를 분할하였던 부분 간의 통합을 가져오지 못하였습니다. 두 파일럿 연구를 통합하는 과정을 진행하지 못한 부분이 아쉬움을 남았습니다.