

ChainBot: 기저 모델들의 동적 연결을 이용하는 자율적인 로봇 물체 조작 에이전트 시스템

ChainBot: An Agent System for Autonomous Robotic Object Manipulation by Dynamically Chaining Multiple Foundation Models

박 경 민¹, 김 인 철^{1*}

(Kyungmin Park¹ and Incheol Kim^{1,*})

¹Department of Computer Science Kyonggi University

Abstract: This paper proposes ChainBot, an agent system that autonomously plans and executes robotic object manipulation tasks based on a user's natural language instructions by dynamically chaining multiple foundation models. The ChainBot system uses a large language model (LLM) for effective task planning. To ensure the generation of correct task plans, the system enriches the LLM prompt with useful context, including plan examples and domain-specific heuristic knowledge. Before execution, ChainBot also uses a secondary LLM for pre-execution validation. If an error is detected, the system replans the task using the validation result as feedback. During task execution, ChainBot employs a large multimodal model (LMM) to monitor the process for quick error discovery and recovery. This is achieved by checking both the preconditions and postconditions of each action using real-time camera images captured before and after execution. As a result, ChainBot achieves fully autonomous robotic manipulation without user intervention when implemented as an agent system using the LangChain and LangGraph frameworks. Through various quantitative and qualitative experiments, this paper demonstrates the superiority of ChainBot compared to conventional systems.

Keywords: robotic manipulation, foundation model, agent system, task planning, plan validation and replanning, error detection and recovery

I. 서론

최근들어 거대 언어 모델(large language model, LLM) [1-3], 시각-언어 모델(vision-language model, VLM)[4,5,6], 거대 멀티모달 모델(large multimodal model, LMM)[1-3] 등 인터넷 스케일의 대용량 데이터 집합들로 학습시킨 거대 기저 모델(foundation model, FM)들이 다양한 인공지능 작업(AI tasks)들에서 놀라운 성능을 보이면서 급속도로 발전하고 있다. 이들은 대용량의 데이터 집합에서 습득한 폭넓은 의미적 지식(semantic knowledge)과 더불어 상당한 추론 능력(reasoning capability) [7,8]을 갖추고 있으면서 인간 및 다른 모델들과 자연어로도 소통(natural language interaction)할 수 있기 때문에, 단순한 전문 기능 수행 도구(AI tool)라기 보다는, 오랫동안 인간이 꿈꿔왔던 종합적인 범용 인공지능체(artificial general intelligence, AGI)를 실현하기 위한 토대로도 받아들여지고 있다. 이러한 거대 기저 모델들과 관련해서 현재 진행되고 있는 발전 방향은 크게 두 가지로 분류될 수 있다. 그 중 하나는 현재의 거대 언어 모델(LLM) 혹은 거대 멀티모달 모델(LMM)들에 더 다양한 입력 센서 데이터 및 제어 정보를 처리할 수 있도록 확장함으로써, 자율 로봇, 자율 주행 자동차 등과 같은 물리적 세계 응용(physical world application)들에도 적용 가능

한 초거대 단일 인공지능 모델을 개발하는 것이다. 현재 활발히 진행되고 있는 시각-언어-행동 모델(vision-language action model, ALV model)에 관한 연구들[9-12]이 그 대표적인 예이다. 기저 모델들과 관련한 두 번째 발전 방향은 현재 가용한 서로 다른 목적과 기능을 가진 기저 모델들을 서로 연결 혹은 연동함으로써, 사용자의 요구에 부응하는 새로운 인공지능 응용 시스템 혹은 서비스들을 효율적으로 개발하는 것이다. VisProg [13], Visual ChatGPT [14], HuggingGPT [15], InternGPT [16], Socratic Model [17], GenArtist [18]과 같은 멀티모달 에이전트 시스템들이 대표적인 예이다. 본 논문에서는 후자의 방향에 따라, 거대 언어 모델을 중심으로 다른 기저 모델들을 동적으로 연결하여 사용자의 자연어 지시에 따른 테이블 위의 물체 조작(tabletop manipulation) 작업들을 자율적으로 계획하고 실행해주는 에이전트 시스템을 제안한다.

이러한 물체 조작 로봇 에이전트 시스템을 개발하기 위해서는 고려해야 할 중요한 몇 가지 설계 이슈가 있다. 첫 번째 이슈는 작업 계획기(task planner) 역할을 수행하는 거대 언어 모델(LLM)에게 어떤 형태의 작업 계획을 어떤 방식으로 수립하도록 요청할 것이며, 이를 위해 어떤 문맥 정보(context)들로 프롬프트(prompt)를 구성할 것인가 하는 계획 수립 프롬프트

*Corresponding Author

Manuscript received November 6, 2025; revised December 17, 2025; accepted January 5, 2026

박경민: 경기대학교 일반대학원 컴퓨터학과 석사과정(gkalsrudals@kyonggi.ac.kr ORCID[®] 0000-0006-9648-2391)

김인철: 경기대학교 AI컴퓨터공학부 교수(kic@kyonggi.ac.kr, ORCID[®] 0000-0002-5754-133X)

※ This work was supported by Kyonggi University Research Grant 2023.

설계(prompt engineering for LLM-based task planning) 문제이다. 관련 선행 연구들 중에서는 작업을 위한 전체 행동 시퀀스(full action sequence)를 계획으로 수립하도록 한 경우[19,20]도 있지만, 어떤 연구들에서는 매순간 실행할 다음 행동(next action)만을 계획하도록 하는 경우[21]도 있다. 또한 일부의 초기 연구들에서는 계획의 형태를 자연어 문장(natural language sentence) [22,17]으로, 이후 어떤 연구들에서는 PDDL [23]이나 python 코드 [19]와 같은 형식 언어(formal language) 형태의 작업 계획을 생성하도록 거대 언어 모델에게 요청하기도 하였다. 이와 같이 거대 언어 모델에게 요구하는 계획 수립 방식과 계획 표현법에 따라 작업 계획 수립의 난이도에 영향을 줄 수 있다. 한편, 거대 언어 모델의 계획 수립을 도와주기 위해, 기존 연구들에서는 계획에 포함되어야 할 행동들의 리스트(action list) [19]와 대상 물체들의 리스트(object list) [17] 뿐만 아니라, 작업 계획 사례들(plan examples) [23]을 문맥 정보로 프롬프트에 포함시켜 제공하기도 하였다. 작업 계획 수립을 위한 프롬프트에 어떤 문맥 정보를 함께 제공하느냐가 거대 언어 모델의 작업 계획 수립 결과에 큰 영향을 주는 것으로 잘 알려져 있기 때문에 매우 중요한 이슈가 된다.

중요한 두 번째 설계 이슈는 거대 언어 모델이 생성한 작업 계획을 실제 작업 환경에서 실행하기에 앞서 어떤 방식으로 사전 점검할 것인가 하는 실행 이전 계획 검증(pre-execution plan validation) 문제이다. 일부 선행 연구들에서는 행동 이름(action name)이나 물체명(object name) 등 계획의 몇 가지 문법적 오류(syntax error)만을 거대 언어 모델을 이용해 사전에 간단히 검증한 경우도 있고[24], 중요한 조건(condition)들이 계획에 누락되었는지를 거대 언어 모델을 이용해 사전에 체크한 경우[25]도 있다. 하지만 이 두 선행 연구를 제외하면, 기저 모델을 이용한 실행 이전 계획 검증 방식에 관한 연구는 아직 심도있게 다루어지지 않고 있는 실정이다.

한편, 실제 물리 환경에서 작업 계획을 실행(plan execution)하는 동안에는 예상하지 못했던 환경 변화나 로봇의 인식기 및 실행 제어기 등의 오류로 인해, 계획 실행이 실패하는 경우가 종종 발생할 수 있다. 따라서 세 번째 중요한 설계 이슈는 어떻게 기저 모델들을 이용해 이러한 계획 실행 중에 발생할 수 있는 다양한 오류들을 탐지하고 이 오류들로부터 작업을 신속히 복구(error detection and recovery)할 것인가 하는 계획 실행 모니터링과 오류 처리(execution monitoring and error handling) 문제이다. 선행 연구들 중에는 작업 계획 실행 중 오류 탐지와 복구를 전혀 고려하지 않은 경우들도 있고[22,17], 작업 계획 내의 각 행동마다 실행 전에 만족해야 할 환경 상태 전조건(precondition)들을 설정해두고 이들을 체크함으로써 행동 실행 불가능한 오류 상태들을 탐지하며 오류 복구를 위해서는 미리 정의된 복구 행동(recovery action)들을 실행하는 경우도 있었다[19]. 또한 이보다 더 빠른 오류 탐지와 복구를 위해, 행동 실행 전후 뿐만 아니라 실행 중인 각 행동의 제약 조건들(constraints)을 매우 짧은 시간 주기로 검사하여 오류 발견 즉시 실행을 멈추고 작업을 재계획하도록 시도한 연구[21]도 있었다. 하지만 실제 로봇 작업 환경마다 오류 발생 빈도, 허용되는 복구 지연 시간, 오류 탐지와 복구를 위한 계산 비용, 그리고 이에 따른 작업 성공률 등에 큰 차이가 존재하므로, 이

들을 종합적으로 고려하여 효율적인 기저 모델 기반의 계획 실행 중 오류 탐지와 복구 메커니즘을 설계할 필요가 있다.

마지막 네 번째 설계 이슈는 다수의 기능별 기저 모델들을 효과적으로 엮어 능동적으로 물체 조작 작업을 수행할 수 있는 로봇 에이전트 시스템을 효율적으로 구현하려면 어떤 에이전트 개발 환경 및 도구(agent development environment or tool)를 이용하는 것이 좋을지를 결정하는 문제이다. 현재 거대 언어 모델 중심으로 에이전트 시스템 개발을 지원하는 다양한 오픈 소스 및 상용 도구들이 시중에 나와있는데, 그 중에서도 Microsoft Copilot Studio [26], Google ADK (Agent Development Kits) [27], LangChain [28] 등이 가장 많이 사용되고 있는 실정이다. 이들은 목표 작업 도메인(target task domain), 기저 모델들과의 연계성, 작업 흐름 표현력, 사용 편의성, 기능 확장성 등에서 차이가 있다. 다양한 관점에서 이들을 종합적으로 비교 평가하여, 본 연구의 목적에 가장 부합하는 에이전트 개발 도구를 선정하는 것이 중요하다.

본 논문에서는 이러한 설계 이슈들을 고려하여, 그림 1과 같이 Gemini-2.0 [2], GPT-5 [1], CLIPort [29] 등의 기저 모델들을 동적으로 연결하여 사용자의 자연어 지시에 따른 테이블 위의 물체 조작 작업들을 자율적으로 계획하고 실행해주는 에이전트 시스템 ChainBot을 제안한다. 먼저 첫 번째 설계 이슈와 관련해서 본 ChainBot 에이전트 시스템은 효과적인 작업 계획 생성을 위해 거대 언어 모델의 하나인 Gemini-2.0을 이용하며, 프롬프트에 환경 물체(environmental objects), 초기 작업 상태 묘사(initial state description), 사용자 지시(user instruction)와 같은 작업의 기본 정보들 외에 난이도가 높은 작업들에 대해서도 올바른 작업 계획 생성을 유도하기 위해 작업 계획 사례들(task plan examples)과 각 도메인 별 경험적 지식(domain-specific heuristic knowledge)과 같은 풍부한 문맥 정보들을 추가로 포함시켜 거대 언어 모델에게 제공한다. 또한 ChainBot 시스템에서는 작업 계획을 구조화된 XML 형태로 표현할 뿐만 아니라, 계획 내 각 행동마다 실행 시에 만족해야 할 전조건(precondition)들과 후조건(postcondition)들을 포함하도록 설계하였다. 또 두 번째 설계 이슈와 관련해서 ChainBot 시스템은 실행 전 작업 계획의 품질을 높이기 위해, 거대 언어 모델 Gemini-2.0을 이용한 작업 실행 전 계획 검증 단계를 수행한다. 그리고 이 단계에서는 작업 계획에 대한 구문적 검사(syntax

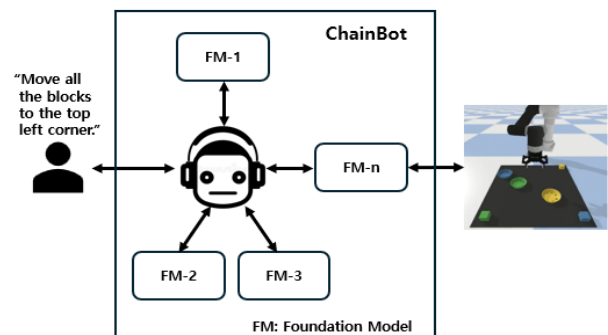


그림 1. 제안 시스템에 대한 개념적 표현.

Fig. 1. Conceptual representation of the proposed system.

check), 의미적 검사(semantics check), 실행 가능성 검사(feasibility check), 그리고 작업 달성 가능성 검사(task completion check) 까지 심도있게 계획 검증을 수행하며, 이를 통해 오류가 발견 되면 검증 결과를 피드백으로 삼아 거대 언어 모델로 하여금 작업을 다시 재계획한다.

또 세 번째 설계 이슈와 관련해서, ChainBot 시스템은 작업 실행 중 빠른 오류 발견과 복구를 위해, 거대 멀티모달 모델인 GPT-5를 이용해 각 작업 행동의 실행 전후에 실시간 카메라 영상을 토대로 해당 행동의 전조건뿐만 아니라 후조건들도 검사함으로써 오류 상태들을 탐지한다. 그리고 오류 발견 즉시 수행 중이던 작업을 멈추고, 오류 복구를 위해 오류 상태부터 시작해서 다시 사용자 지시를 달성할 수 있는 작업 계획을 재수립해서 실행한다. 따라서 ChainBot 에이전트 시스템은 풍부한 문맥 정보를 포함한 작업 계획 생성 프롬프트 이용과 실행 전 작업 계획 검증과 재계획, 그리고 작업 실행 중 신속한 오류 탐지와 복구 등을 통해 작업 성공률을 크게 향상시킬 수 있다. 마지막 네 번째 설계 이슈와 관련해서 사용자의 특별한 개입없이도 다수의 거대 기저 모델들을 유연하게 연동하여 로봇 조작 작업들을 능동적으로 수행할 수 있도록, ChainBot 에이전트 시스템은 대표적인 에이전트 개발 프레임워크인 LangChain과 LangGraph를 기반으로 효율적으로 구현되었다. 끝으로 본 논문에서는 다양한 정량적, 정성적 실험들을 통해, ChainBot 에이전트 시스템의 우수성을 입증한다.

서론에 이어서 본 논문의 2장에서는 관련 선행 연구들을 소개하고, 3장에서는 제안 시스템인 ChainBot의 구성과 설계에 대해 상세하게 설명한다. 4장에서는 ChainBot시스템의 구현에 관한 세부사항들과 성능 평가 실험 결과들을 소개하며, 마지막으로 5장에서는 결론과 향후 연구에 관해 정리한다.

II. 관련 연구

1. 기저 모델-기반의 멀티모달 에이전트

최근에는 OpenAI GPT-4 [1], Google Gemini 2.0 [2], Meta Llama 4 [3]와 같은 거대 언어 모델(LLM), 그리고 CLIP [4], ViLD [5], BLIP-2 [6]와 같은 시각-언어 모델(VLM) 등 다양한 기저 모델(foundation model, FM)들이 공개됨에 따라, 이 기저 모델들을 활용하여 사용자에게 편리한 멀티모달 작업들을 수행해주는 멀티모달 에이전트(multimodal agent)들의 개발이 활발해졌다. VisProg [13]은 GPT-3를 이용해 사용자의 자연어 지시(natural language instruction)를 고수준의 프로그램으로 자동 변환하여 실행함으로써, 사용자가 원하는 다양한 영상 작업을 대신 수행해주는 에이전트이다. VisProg가 생성하는 고수준의 프로그램은 실행하는 동안 영상 처리 및 분석을 위한 다양한 모듈들을 상호 결합하여 작업을 수행한다. Visual ChatGPT [14]와 HuggingGPT [15]는 공통적으로 OpenAI ChatGPT를 중심으로 다양한 여러 시각적 기저 모델(visual foundation model)들을 서로 연결함으로써, 사용자는 ChatGPT와만 자연어로 편리하게 소통하여도 원하는 영상 이해 및 생성 작업들을 쉽게 수행할 수 있도록 설계된 멀티모달 에이전트들이다. 한편, InternGPT [16]에서는 영상 처리 작업을 위해 거대 언어 모델 기반의 자연어 지시 인식 기능 외에 포인터 인식기를 이용한 입력 영상 포인팅(pointing) 기능까지 제공하고 있다.

SM (Socratic Model) [17]은 앞선 시스템들과 유사하게 거대 언어 모델(LLM), 시각-언어 모델(VLM), 청각-언어 모델(ALM) 등과 같은 다양한 기저 모델들을 상호 연결함으로써, 별도의 모델 추가 학습없이도 사용자에게 새롭고 편리한 작업들을 실행해줄 수 있는 에이전트 시스템이다. 하지만 이 시스템의 가장 큰 특징은 각기 다른 모달의 정보 처리 능력을 가진 기저 모델들 간의 상호 교류의 중간 매체로서 별도의 형식 언어가 아닌 자연 언어를 이용한다는 점이다. GenArtist [18]의 경우에는 영상 생성과 편집(image generation and editing) 작업들에 특화된 멀티모달 에이전트로서, 자연어만 다룰 수 있는 거대 언어 모델(LLM) 대신에 영상과 자연어를 함께 이해할 수 있는 거대 멀티모달 모델(LMM)을 채용하여 복잡한 작업을 다수의 단순 작업들로 나누어 트리 형태의 작업 계획을 수립하고 이를 실행하는 시스템이다. 한편, VisualAgentBench [30]에서는 멀티모달 에이전트의 핵심 구성요소로서 거대 멀티모달 모델(LMM)의 학습과 성능 평가를 위한 대규모 벤치마크 데이터 집합을 새롭게 제시하였다. 대부분의 기존 멀티모달 에이전트 시스템들은 서버 혹은 데스크탑 컴퓨터에서 실행되는 것을 가정하였으나, AppAgent [31]의 경우에는 스마트폰에서 동작하면서 사용자 대신 스마트폰에 설치되어 있는 다양한 멀티모달 앱들을 실행시켜주는 에이전트이다. 이 에이전트는 학습을 통해 사용자의 화면 터치(tapping)와 화면 밀기(swiping) 등과 같은 행동들을 대신 실행해줌으로써, 굳이 기존의 스마트폰 에이전트들과 같이 시스템 후단부(backend)에 접근할 필요가 없다는 점이 주요한 차별점이다.

거대 언어 모델 혹은 거대 멀티모달 모델을 이용한 멀티모달 에이전트의 개발이 활발해지자, 최근에는 이를 지원하기 위한 편리한 개발 도구 및 개발 환경들도 소개되었다. Microsoft Copilot Studio [26], Google ADK(Agent Development Kits) [27], LangChain [28] 등이 그 대표적인 예들이다. Copilot Studio [26]은 비-개발자나 비즈니스 사용자 중심의 에이전트 개발 도구이며, 거대 언어 모델은 Copilot으로 제한하는 등 주로 Microsoft 생태계 위주로 에이전트를 구성하여 확장성에 제한이 있고, 챗봇이나 고객 상담 자동화, 업무 프로세스 자동화 등에 많이 이용된다. 한편, Google ADK [27]과 LangChain [28]은 둘 다 개발자나 연구자 중심의 도구이며, 에이전트의 핵심 구성 요소인 거대 언어 모델의 선택에 대한 제한이 없고, 외부 도구나 API 등을 쉽게 연결할 수 있어 확장성이 높다. 하지만 작업 흐름인 워크플로우(workflow)를 정의할 때, 현재 Google ADK는 순차(sequential)와 병렬(parallel) 흐름 구조만 지원하지만 LangChain은 LangGraph를 이용해 보다 자유롭고 다양한 작업 흐름을 표현할 수 있다는 차이점이 있다.

2. 로봇 공학에서 기저 모델들의 활용

로봇 공학(robotics)에서 기저 모델들이 가장 많이 활용된 영역은 거대 언어 모델(LLM)들을 이용한 작업 계획 수립(task planning) 분야이다. 하지만 일반적으로 거대 언어 모델들은 작업 계획 수립에 필요한 풍부한 의미적 기초 지식을 보유하고 있으나, 실제 로봇이 작업을 수행해야 하는 현실 환경과 로봇, 그리고 관련 사물들에 대한 구체적인 정보는 갖고 있지 못하다. 이러한 현실 정보없이 거대 언어 모델이 수립해주는 작업 계획들은 현실 환경에서는 실행 불가능한 경우가 많다. 따

표 1. 관련 연구들과의 비교.

Table 1. Comparison with related works.

	Task Planning	Plan Verification	Runtime Error Detection
SayCan [22]	- LLM에 의해 로봇이 다음 실행할 행동만 계획 - 확률을 동반한 자연어 문장 형태로 행동/계획 표현 - 계획에 각 행동별 전조건과 후조건 불포함 - 행동 리스트를 계획수립 프롬프트 맥락정보로 제공	- LLM이 추천한 행동에 대해 미리 학습된 가치 함수를 적용 - 실행 가능성이 높은 행동만 실행	- 계획 실행 중 별도의 오류 탐지 및 복구 기능은 없음
ProgPrompt [19]	- LLM에 의해 로봇의 행동 시퀀스 전체를 계획 - Python 코드 형태로 조건부 계획 (conditional plan) 표현 - 계획에 각 행동별 전조건 포함 - 행동 리스트를 계획수립 프롬프트 맥락정보로 제공	- 별도의 실행 전 계획 검증 기능 없음	- 계획 실행 중 각 행동의 시점에서 전조건 만족 여부만 체크 - 해당 조건이 만족되지 않는 오류 상태이면, 계획 내의 다른 분기 행동을 실행
CodesAsPolicies [34]	- LLM에 의해 로봇 행동 정책을 계획 - Python 코드 형태로 정책/계획 표현 - 계획에 각 행동별 전조건 포함 - 행동 리스트를 계획수립 프롬프트 맥락정보로 제공	- 별도의 실행 전 정책 검증 기능 없음	- 정책 실행 중 별도의 오류 탐지 및 복구 기능은 없음
DoReMi [21]	- LLM에 의해 로봇이 다음 실행할 행동만 계획 - 자연어 문장 형태로 행동/계획 표현 - 행동 리스트를 계획수립 프롬프트 맥락정보로 제공	- 별도의 실행 전 계획 검증 기능 없음	- 계획 실행 중 각 행동이 수행되고 있는 동안에도 주기적으로 제약 조건 위배 여부를 체크하여 오류 탐지 - 오류 탐지 시 이미 정의해둔 복구 행동 수행
ChainBot [Ours]	- LLM에 의해 로봇의 행동 시퀀스 전체를 계획 - XML 형태로 계획을 표현 - 계획에 각 행동별 전조건과 후조건 포함 - 행동 리스트, 물체 리스트, 계획 사례들, 도메인별 경험적 지식 등을 계획수립 프롬프트 맥락정보로 제공	- 작업 계획의 구문적 검사 - 작업 계획의 의미적 검사 - 작업 계획의 실행 가능성 검사 - 작업 계획의 작업 완수 가능성 검사 - 이 검사들을 차례대로 수행하는 중 하나라도 오류 발견시 재계획 수립	- 계획 실행 중 각 행동의 시작 시점과 종료 시점에서 각각 전조건뿐만 아니라 후조건의 만족 여부도 체크 - 오류 탐지 시 복구 재계획 수립

라서 많은 선행 연구들에서는 이러한 문제점을 해결하기 위해, 다양한 방법들을 시도하였다.

SayCan [22] 연구에서는 작업 계획에 포함될 수 있는 이미 학습된 다수의 로봇 행위(behavior)나 스킬(skill)들마다 이들의 실행 가능성을 평가해주는 가치 함수(value function)를 두어, 거대 언어 모델이 이 가치 함수들을 참조함으로써 실행 가능성이 높은 작업 계획을 수립할 수 있도록 설계하였다. ProgPrompt [19]에서는 거대 언어 모델에게 요청할 프롬프트(prompt)에 현실 환경에서 실행할 수 있는 행동(action)들과 사물(object)들에 대한 프로그램 스타일의 구체적인 명세(program-like specification), 그리고 다수의 예제 프로그램(example program)들을 포함시킴으로써, 현실 환경에서 실행 가능성이

높은 작업 계획 수립을 유도하였다. 거대 언어 모델을 로봇 작업 계획 수립에 이용한 또 다른 연구로는 LLM-Personalize [32]가 있다. 이 연구에서는 거대 언어 모델-기반의 작업 계획기가 가정 환경에 거주하는 특정 사용자의 선호도를 반영한 로봇 작업 계획을 수립할 수 있도록, 장면 그래프(scene graph) 기반의 모방 학습(imitation learning)과 반복적 자기-주도 훈련(iterative self-training)을 수행하였다.

한편, [33]의 연구에서는 작업 계획 수립과 모션 계획 수립을 함께 수행해야 하는 작업 및 모션 계획 수립(task and motion planning, TAMP)의 관점에서 거대 언어 모델을 이용하였다. 이 연구에서는 거대 언어 모델이 고수준의 작업 계획 수립과 함께 작업 문맥 정보(context)도 추출하여 이들을 하위-레벨의 모션 조정

(motion tuning)에 이용하는 행동 맥락화(action contextualization)를 수행하였을 뿐만 아니라, 역으로 각 하위-레벨 모션의 성능 평가에 따라 거대 언어 모델이 즉각적으로 상위-레벨의 작업 계획을 변경(plan modification)하거나 재 계획(replanning)하도록 시스템을 설계하였다. LM-SymOpt [20]의 경우에는 현실 환경에서 실행가능하면서도 최적의 모션 경로(motion trajectory)를 보장하는 로봇 물체 조작 작업 계획을 얻기 위해, 거대 언어 모델을 이용하여 사용자의 자연어 지시로부터 작업 계획 수립에 필요한 핵심 정보만을 담은 기호적 표현(symbolic representation)을 추출한 후, 이 기호적 표현을 토대로 미리 정의된 실행 가능 제약 조건(feasibility constraint)들을 만족하는 작업 계획들을 먼저 생성한다. 이후 이 작업 계획들이 요구하는 모션 경로들을 비교하여, 최적의 모션 경로를 갖는 작업 계획을 최종 선정하는 방식을 취했다.

거대 언어 모델을 이용해 간결한 자연어 텍스트 형태의 로봇 작업 계획을 생성하던 기존 연구들과는 달리, [34]의 연구에서는 Python 프로그램과 같은 코드 생성이 가능한 거대 언어 모델을 이용하여, 사용자가 제시한 고수준의 자연어 명령을 현실 작업 환경에서 직접 실현해줄 수 있는 구체적인 로봇 정책 코드(robot policy code)를 작성하였다. 이 정책 코드에는 물체 탐지기(object detector)와 같은 인식 모듈들의 인식 결과를 처리할 수 있는 API (application programming interface) 함수 호출들과 행동 제어를 위한 API 함수 호출들이 포함된다.

한편, SMART-LLM [35]와 RoCo [36]의 경우에는 단일 로봇(single robot)이 아닌, 다중 로봇들의 협업(multi-robot collaboration)을 위한 작업 계획을 수립하는데 거대 언어 모델을 이용하였다. SMART-LLM [35]는 최종 작업 계획에 따라 각 로봇이 각자 맡은 부속 작업들을 실행하는 마지막 작업 실행(task execution) 단계를 제외하면, 작업 분해(task decomposition), 연합 형성(coalition formation), 작업 할당(task allocation) 등 다중-로봇 작업 계획 수립 과정의 모든 단계들을 모두 중앙의 거대 언어 모델을 이용해 수행하는 중앙-집중형 작업 계획 수립 방식을 채택하였다. 이에 반해, RoCo [36]은 협업에 참여하는 각 로봇마다 각자의 거대 언어 모델을 내포하고 있으면서 이들간에서 서로 통신으로 소통하며 상호 충돌을 피할 수 있는 각자의 작업 계획을 수립하는 분산형 작업 계획 수립 방식을 적용하였다.

거대 언어 모델들 외에 시각-언어 기저 모델(vision-language foundation model)들을 로봇 공학에 활용한 연구들도 다수 있는데, CLIP [4]를 이용한 CLIPort [29]와 PaRaGon [37]을 대표적인 예로 들 수 있다. CLIPort [29]는 정밀한 시각적 로봇 조작(visual manipulation) 작업을 수행할 수 있는 기존의 Transporter 신경망에 자연어 지시와 입력 영상을 개념적으로 이해할 수 있는 CLIP 텍스트 및 영상 인코더(text and image encoder)를 서로 연동함으로써, 사용자가 구체적인 좌표들을 지정할 필요 없이 간결하고 함축적인 자연어 지시만을 통해 테이블 위(tabletop)의 다양한 물체 조작 작업들을 수행할 수 있는 에이전트 시스템이다. PaRaGon [37] 역시 사용자의 자연어 지시에 따라 테이블 위의 pick-and-place 물체 조작 작업을 수행하는 모델인데, 자연어 지시에서 조작 대상 물체 단어를 식별해내는 소프트 파싱(soft parsing) 단계와 입력 영상에서 해당 물체를 찾아내는 시각적 그라운드링(visual grounding) 단계에서 각각

CLIP 텍스트 인코더와 CLIP 영상 인코더를 이용하였다. 한편, VLM-CaR [38]의 연구에서는 대표적인 시각-언어 모델(VLM)이자 거대 멀티모달 모델(LMM)인 GPT-4 [1]을 이용하여, 사용자가 자연어로 지시한 작업으로부터 다수의 부속 작업들(sub-tasks)과 이들에 대응되는 보상 함수(reward function) 코드들을 자동으로 생성하여, 작업 정책(task policy)을 얻기 위한 강화 학습(reinforcement learning)에 이용하기도 하였다.

한편, 거대 언어 모델이 수립한 작업 계획은 실제 실행 단계에서 환경의 갑작스런 변화나 불완전한 제어기의 작동 등 다양한 현실적인 이유들로 인해, 작업 실패의 상황에 종종 빠지기도 한다. 이때, 신속히 작업 실패를 탐지해내고 다시 작업을 복구하는 일이 매우 중요하다. 이와 같은 작업 실패 탐지와 복구(task failure detection and recovery)를 위해, 거대 언어 모델과 시각-언어 모델들을 이용하려는 연구들도 등장하였다. DoReMi [21]의 경우에는 로봇 작업 계획 수립 단계에서 거대 언어 모델을 이용하여 작업 계획뿐만 아니라 이 작업 계획을 실행하는 동안 만족되어야 할 제약 조건들(constraints)도 함께 생성하도록 한 후, 계획 실행 단계에서는 시각-언어 모델을 이용하여 지속적으로 제약 조건들의 만족 여부를 체크하도록 하였다. 그리고 제약 조건이 위배되는 실패 상황이 탐지되면, 즉시 해당 작업 계획의 실행을 멈추고 거대 언어 모델로 하여금 작업 계획을 재수립(replanning)하게 하는 방식으로 작업 실패로부터 복구를 시도하였다. RoboMD [39]와 Aha [40] 연구들에서는 로봇 물체 조작 작업에서 작업 실패 상황들을 효과적으로 탐지해내기 위해, 새롭게 수집된 데이터 집합들과 이들을 토대로 학습된 새로운 시각-언어 모델들을 각각 제시하였다.

최근에는 거대 언어 모델(LLM) 및 거대 멀티모달 모델(LMM)들을 더욱 확장하여, 언어, 영상, 비디오, 사운드와 같은 일반적인 멀티모달 데이터 이외에 추가로 다양한 로봇 센서 데이터와 모션 제어 정보까지 이해하고 생성할 수 있는 PaLM-E [9], OpenVLA [10], Gemini Robotics [11], OpenHelix [12]와 같은 초거대 시각-언어-행동 모델들(vision-language-action models, VLA models)도 등장하고 있는 실정이다. 끝으로 본 논문에서 제안하는 ChainBot 시스템과 기존 관련 연구들과의 차이점들을 표 1에 간략히 비교·정리하였다.

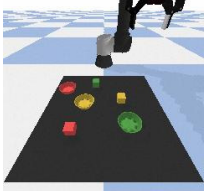
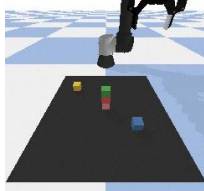
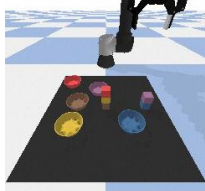
III. 물체 조작 로봇 에이전트 시스템의 설계

1. 조작 작업 도메인

본 논문에서 제안하는 에이전트 시스템 ChainBot은 표 2과 같은 서로 다른 세 가지 도메인(domain)의 로봇 물체 조작 작업들을 수행한다. 이 작업들은 사용자의 자연어 지시를 만족하기 위해서는 다양하게 초기 배치된 블록(block)들에 대해 연속적인 Pick-and-Place 행동(action)들의 실행을 공통적으로 요구한다. 하지만, 세 가지 작업 도메인별로 기본 전제(assumption)과 제약 조건(constraint), 난이도(complexity)가 서로 다르다. 표 2의 블록 그릇에 넣기(Put-blocks-in-bowls, 이하 PBB로 표기한다) 작업들은 초기 상태(initial state) 영상과 사용자 지시(user instruction)로 표시된 예시 작업들(example tasks)과 같이 대부분 다양한 색상의 블록(block)들과 그릇(bowl)들이 테이블 위에 놓여 있는 초기 상태에서 시작하여 색상에 맞추어 블록

표 2. 로봇 조작 작업 도메인들.

Table 2. Robotic manipulation task domains.

Task domains	Put-blocks-in-bowls (PBB)	Stack-blocks (SB)	Modified-PBB (MBB)
Example initial states			
Example user instructions	“Place each block into the matching bowl.” “Place the blocks in clockwise order at each corner.”	“Stack the blocks bottom-to-top as green block, yellow block, pink block, blue block, red block”	“Place each block into a bowl of a different color.” “Stack the blocks vertically with the red block at the bottom.”

들을 그릇에 넣거나 테이블 위의 특정 위치 혹은 특정 순서에 따라 배치시켜 놓는 작업들이다. 이 PBB 도메인의 작업들은 블록과 그릇의 초기 배치 상태뿐만 아니라 작업 행동의 실행 순서에도 크게 영향을 받지 않아 사용자 지시를 달성하기 용이한 작업들이다. 반면에 표 2의 블록 쌓기(Stack-blocks, 이하 SB로 표기한다) 작업들은 이미 다양한 색상의 블록들이 다양한 형태로 서로 쌓여 있거나 테이블 위에 놓여 있는 초기 상태에서 시작해서 사용자 지시에 따라 일정한 모습으로 블록 탑을 새로 쌓거나 일부 블록들은 테이블의 특정 위치들에 배치시켜야 하는 작업들이다. 이 SB 도메인의 작업들은 블록의 초기 배치 상태에 민감하게 영향을 받는다. 또한, 사용자 지시를 성공적으로 달성하기 위해서는 작업 행동들 간의 인과 관계(causal relationship)와 상호 작용(interaction)을 신중히 고려하여 실행 순서를 결정해야 하므로 난이도가 높다. 마지막으로 표 2의 변형된 블록 그릇에 넣기(Modified Put-blocks-in-bowls, 이하 MBB로 표기한다) 작업들은 PBB 도메인에 SB 도메인의 특성을 혼합하여 확장한 도메인이다. 즉, 사용자의 지시에 따라 다양한 색상의 블록들을 색상에 맞추어 그릇들에 집어 넣거나 혹은 테이블의 특정 위치들에 배치시킬 뿐만 아니라 서로 다른 블록들끼리 탑 형태로 쌓거나 하는 작업들이다. 이 MBB 도메인의 작업들은 SB 도메인과 마찬가지로 블록과 그릇의 초기 상태 배치 및 작업 행동의 실행 순서에 크게 영향을 받는다. 또한, SB 도메인보다 환경 내 물체 수가 많아 가장 난이도가 높다.

2. 모델 개요

본 논문에서는 기저 모델들을 동적으로 연결하여 사용자의 자연어 지시에 따른 테이블 위의 물체 조작 작업들을 자율적으로 계획하고 실행해주는 에이전트 시스템 ChainBot을 제안한다. 그림 2는 ChainBot 시스템의 전체 구성을 나타내며, 이 시스템은 크게 1) 웹 사용자 인터페이스(Web User Interface), 2) 작업 관리자(Task Manager), 3) 물체 탐지기(Object Detector), 4) 상태 묘사 생성기(State Descriptor), 5) 작업 계획기(Task Planner), 6) 계획 검증기(Plan Validator), 7) 작업 계획 실행기(Plan Executor), 8) 행동 제어기(Action Controller), 9) 오류 탐지기(Error Detector) 등의 모듈들로 구성된다. 그리고 웹 사용자 인터페이스, 작업 관리자, 작업 계획 실행기를 제외한 모든 모듈은 후단(backend)에 위치한 하나의 기저 모델과, 이 기저 모델에

대한 프롬프팅 및 타 모듈들과의 통신을 담당하는 Python 래퍼 코드(wrapper code)의 쌍(pair)으로 구성된다.

Streamlit [41] 구현되는 웹 사용자 인터페이스(Web User Interface)는 사용자로부터 조작 작업을 위한 자연어 지시를 받아들이고, 작업의 중간 진행 상태 및 최종 결과를 사용자에게 제시하는 웹 기반의 그래픽 사용자 인터페이스이다. 작업 관리자(Task Manager)는 사용자의 지시에서 요구하는 작업을 완수할 수 있도록 ChainBot 시스템 구성 모듈들 간의 협업과 전반적인 작업 흐름을 조정(coordinate)하는 중추적인 역할을 수행하는 모듈이다. 사용자 지시는 대부분 사용자가 원하는 작업 목표 상태를 간결하게 자연어로 묘사할 뿐, 작업 환경 구성이나 작업의 초기 상태, 그리고 세부적인 작업 절차들은 포함하고 있지 않다. 따라서 ChainBot 시스템은 사용자 지시 외의 나머지 작업 핵심 정보들을 대부분 기저 모델 기반의 인식 모듈들을 이용하여 작업 시뮬레이션 환경(Simulation Environment)으로부터 직접 추출해낸다. 즉 작업 관리자의 요청에 따라 Google Gemini-2.5 Flash 기저 모델 기반의 물체 탐지기(Object Detector)는 작업 시뮬레이션 환경의 실시간 카메라 영상으로부터 테이블 위의 작업 대상 물체들을 인식해내고, 반면에 GPT-5 기저 모델 기반의 상태 묘사 생성기(State Descriptor)는 작업 대상 물체들의 위치와 그들 간의 배치 관계들을 중심으로 작업의 초기 상태(initial state)를 나타내는 자연어 묘사 문장들을 생성해낸다.

이렇게 확보한 작업 대상 물체들의 리스트, 초기 상태 묘사 문장들, 그리고 사용자가 지시로 제공한 작업 목표를 토대로 계획 수립을 위한 기본 프롬프트를 구성할 수 있다. 하지만 ChainBot 시스템에서는 난이도가 높은 작업 계획 문제들에 대해서도 올바른 작업 계획 생성을 유도하기 위해, 작업 계획 생성 프롬프트에 동일 도메인의 다른 작업 계획 예시들(plan examples)과 도메인 별 고유한 경험적 지식(domain-specific heuristic knowledge)와 같은 유용한 문맥 정보들(contexts)도 함께 포함시킨다. 이와 같이 구성된 프롬프트를 토대로 Google Gemini-2.0 거대 언어 모델 기반의 작업 계획기(Task Planner)는 작업의 초기 상태에서 시작하여 사용자 지시가 나타내는 목표 상태에 도달 가능한 작업 계획을 자율적으로 수립한다. 구조화된 XML 형태의 작업 계획에는 목표 달성을 위한 일련의 순서화된 행동들뿐만 아니라, 각 행동별로 실행 전에 만족해야

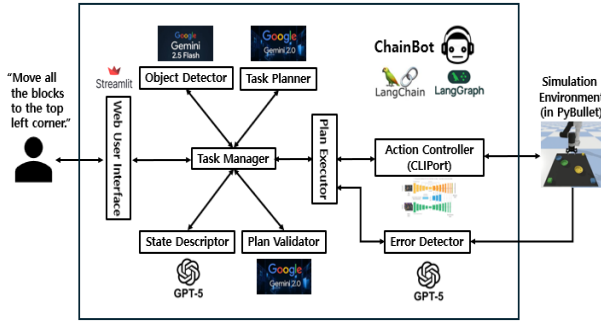


그림 2. ChainBot 시스템의 구성 모듈들.

Fig. 2. Component modules of the ChainBot system.

하는 환경 상태 조건을 나타내는 전조건(precondition)과 실행 후에 충족되어야 할 환경 상태 조건을 나타내는 후조건(postcondition)을 포함한다.

작업 계획기가 수립한 작업 계획 초안에는 구문적, 논리적 오류가 포함될 수 있다. 따라서 ChainBot 시스템에서는 미연에 이러한 작업 실패의 원인이 되는 계획 오류들을 제거하고 작업 성공률을 높이기 위해, 작업 계획 실행에 앞서 Google Gemini-2.0 거대 언어 모델 기반의 계획 검증기(Plan Validator) 모듈을 통해 미리 작업 계획을 검증하고 오류가 있으면 다시 작업 계획기를 통해 작업을 재계획한다. 허용된 횟수 범위 내에서 계획 검증과 재계획이 반복적으로 이루어져 작업 계획의 품질을 충분히 개선한 후에는, 작업 관리자가 이 계획을 작업 계획 실행기(Plan Executor)로 전달하여 실행 단계로 이행한다.

작업 계획 실행기(Plan Executor)는 계획에 기술된 순서대로 각각의 pick_and_place 작업 행동을 차례대로 CLIPort 기저 모델 기반의 행동 제어기(Action Controller)에게 요청하여 실행한다. 동시에 다른 한편으로는 GPT-5 기반의 오류 탐지기(Error Detector)로 하여금 각 행동의 실행 전후에 작업 시뮬레이션 환경의 카메라 영상을 토대로 각 행동의 전조건과 후조건들을 체크함으로써 오류 상태를 탐지해내도록 한다. 만약 작업 계획 실행 중에 오류 탐지기(Error Detector)에 의해 오류 상태가 발견되면, 작업 계획 실행기(Plan Executor)는 즉시 작업 수행을 멈추고 오류 발견을 작업 관리자(Task Manager)에게 통보한다. 그러면 작업 관리자는 이 오류 상태에서 시작하여 작업 목표를 달성할 수 있는 작업 계획 수립을 작업 계획기(Task Planner)에게 요청하고 그 결과로 얻은 복구 계획(recovery plan)을 토대로 계획 실행기(Plan Executor)는 남은 작업을 마저 수행함으로써 작업을 완수한다.

ChainBot 시스템의 각 모듈과 이들 간의 상호 작용은 모두 에이전트 개발 프레임워크인 LangChain과 LangGraph를 토대로 구현한다. ChainBot 시스템의 각 모듈은 LangChain 프레임워크를 기초로 하나의 독자적인 에이전트(agent) 단위로 구현되기 때문에, ChainBot 시스템은 하나의 멀티에이전트 시스템(multi-agent system)으로도 볼 수 있다. 또한 에이전트들 간의 상호 작용은 LangGraph 프레임워크를 기초로 그래프(graph) 형태로 표현하고 구현한다. 즉, 각각의 에이전트는 하나의 노드(node)로, 그들 간의 연결 및 상호 작용은 간선(edge)들로 표현하는 것이다. 이때 간선들은 대부분 조건부 간선(conditional

edge)으로서, 작업 진행 상황에 따라 그래프 구성이 동적으로 변화할 수 있다.

3. 작업 계획 생성을 위한 프롬프트 문맥 정보

거대 언어 모델(LLM)을 이용해 작업의 초기 상태에서 목표 상태에 이르는 일련의 행동들을 한꺼번에 작성하는 소위 행동 시퀀스 계획 수립(action sequence planning) 방식은 계획 문제의 난이도가 높아지고 행동 시퀀스가 길어질수록 계획상의 오류 발생 가능성이 높아질 뿐만 아니라, 실행 환경의 가변성과 불확실성이 높아질수록 빈번하게 계획 변경이 일어나야 한다는 문제점이 있다. 따라서 최근 선행 연구들에서는 거대 언어 모델(LLM)을 이용해 매번 다음 실행할 행동 하나만을 계획하는 다음 행동 계획 수립(next action planning) 방식을 대부분 채택하고 있다. 하지만 이러한 다음 행동 계획 수립 방식은 하나의 작업을 수행하는 동안에 거대 언어 모델(LLM)에 대한 프롬프트가 매우 빈번하게 이루어져야 하기 때문에 고 비용 발생과 작업 수행 지연 시간 등의 문제점들이 있다. 본 연구에서는 이러한 양쪽 계획 수립 방식의 문제점들을 고려하여, 거대 언어 모델(LLM) 기반의 배치 행동 계획 수립 방식을 채택하되, 난이도가 높은 계획 문제들에 대해서도 거대 언어 모델이 올바른 작업 계획을 수립할 수 있도록 프롬프트(prompt)에 유용한 문맥 정보(context)를 추가로 포함시키는 접근 방식을 사용한다.

표 3. 작업 계획의 예시.

Table 3. An example of task plan.

```
<plan>
<sequence>
<apack>
  <precondition>The red block is empty on top.
  </precondition>
  <precondition>The blue bowl is empty on top.
  </precondition>
  <action>robot.pick_and_place("red block", "blue bowl")
  </action>
  <postcondition>The red block is placed on top of the
    blue bowl. </postcondition>
  <postcondition>The blue bowl is not empty on top.
  </postcondition>
</apack>
<apack>
  <precondition>The blue block is empty on top.
  </precondition>
  <precondition>The red bowl is empty on top.
  </precondition>
  <action>robot.pick_and_place("blue block", "red bowl")
  </action>
  <postcondition>The blue block is placed on top of the
    red bowl.</postcondition>
  <postcondition>The red bowl is not empty on top.
  </postcondition>
</apack>
</sequence>
</plan>
```

표 4. 작업 계획 생성을 위한 거대 언어 모델 프롬프트의 예시.
Table 4. An example of LLM prompt for task planning.

Your role is the task planner that should generate a sequence of robotic manipulation actions to achieve the user instruction from the given initial state. The robot is assumed to be able to do only the pick_and_place action, which picks up a block and then places it on another block, a bowl, or one of the on-table locations. Valid pick_targets, place_targets, and on-table locations are as the followings: pick_targets: blocks place_targets: blocks, bowls, and on-table locations in the environment on-table locations: top left corner, top side, top right corner, left side, right side, bottom left corner, bottom side, bottom right corner A task plan must be expressed in the following XML format: {task_plan_format} Explanation of the plan XML format: {explanation_of_plan_xml_format} Now generate the plan for the current robot task. The objects in the task environment: {found_objects} The initial state: {initial_state_descriptions} The user instruction: {user_instruction} You can make use of the following additional information: Example task plans: {plan_examples} Domain heuristic rules: {domain_rules} ...

작업 계획을 표현하는 방식(task plan representation)은 작업 계획 수립(task planning) 뿐만 아니라, 이후 진행될 계획 검증(plan validation), 그리고 계획 실행(plan execution) 및 오류 탐지(error detection)에 이르기까지 모든 과정에 중요한 영향을 미친다. 선행 연구들에서는 PDDL이나 Behavior Tree와 같은 표현력이 높은 형식 언어(formal language) 형태의 작업 계획을 거대 언어 모델로 생성하려는 시도들도 있었으나, 아직은 난이도가 높은 현실적인 계획 문제들에 대해서는 구문적으로 논리적으로 올바른 작업 계획들을 얻기 어려운 실정이다. 따라서 본 ChainBot 시스템에서는 이러한 어려운 형식 언어 형태는 피하고 가급적 거대 언어 모델들이 이해하기 쉬운 자연어 문장에 가깝게 작업 계획 요소들을 표현하되, 이후 진행될 처리 단계들에서 파싱(parsing)에 용이하도록 작업 계획 전체는 표 3의 예시와 같이 XML 형태로 표현한다. 계획 내의 각 행동은 하나의 행동 패키지(action package)를 나타내는 <apack> 태그로 구분되고, 각 행동 패키지 내에는 <action> 태그로 표현되는 하나의 pick_and_place 행동 외에 <precondition> 태그로 표현되는 해당 행동의 실행 전조건들과 <postcondition> 태그로 표현되는 실행 후조건들도 함께 포함된다.

ChainBot 시스템의 작업 계획 수립에는 작업 목표를 나타내는 사용자 지시(user instruction) 외에, 물체 탐지기(Object

Detector)를 이용해 얻는 작업 대상 물체들의 리스트(object list), 상태 묘사 생성기(State Descriptor)를 이용해 얻는 작업 초기 상태 묘사들(initial state descriptions) 등의 기본 정보가 사용된다. 여기에 ChainBot 시스템은 동일 도메인의 다른 작업 계획 예시들(task plan examples)과 도메인별로 고유한 경험적 지식(domain-specific heuristic knowledge)들과 같은 문맥 정보들을 추가하여 거대 언어 모델에게 보낼 작업 계획 생성 프롬프트를 구성한다. 도메인별 고유한 경험적 지식의 한 예로는 블록 쌓기 SB 도메인 작업들의 경우 “탑(tower)을 구성하는 아래쪽 블록들부터 먼저 쌓아라” 라고 하는 것이다.

표 4는 작업 계획 생성 프롬프트의 한 예를 보여준다. 이 프롬프트의 앞부분에는 세 가지 도메인 모두에 공통적인 작업 환경 구성 요소들과 XML 형태의 작업 계획 표현 형식에 대한 소개들이 등장하고, 이어서 계획 수립을 해야 하는 현재 작업의 기본 명세 정보들(입력 영상에서 탐지한 작업 대상 물체들(found_object), 입력 영상을 기초로 생성된 초기 상태 묘사 문장들(initial_state_descriptions), 입력받은 사용자 지시(user_instruction) 등)이 제시된다. 프롬프트의 마지막에는 해당 도메인의 작업 계획 수립에 도움을 줄 수 있는 작업 계획 예시 및 도메인별 경험적 지식과 같은 추가적인 문맥 정보들이 제공된다.

4. 계획 검증과 재계획

유용한 프롬프트 문맥 정보를 함께 제공하더라도 난이도가 높은 문제들에 대해서는 아직까지 거대 언어모델 기반의 작업 계획 생성기가 수립하는 작업 계획에는 다양한 형태의 오류들이 포함되어 있을 수 있다. 만약 이러한 불완전한 작업 계획들을 그대로 실행한다면, 빈번한 작업 실패와 더불어 그로 인한 큰 비용적 손실을 초래할 수 있다. 따라서 본 ChainBot 시스템에서는 작업 계획 실행 전에 미리 거대 언어 모델(LLM) 기반의 계획 검증기를 이용해 작업 계획을 심도있게 검증한다. 그리고 만약 오류가 발견되면 작업 계획 생성기로 하여금 검증 결과를 피드백(feedback)받아 해당 오류를 배제하고 올바른 작업 계획을 수립할 수 있도록 다시 재계획(replanning)을 시도한다.

ChainBot 시스템의 계획 검증기에서 수행하는 검사들로는 구문적 검사(syntax check), 의미적 검사(semantics check), 실행 가능성 검사(feasibility check), 작업 완수 가능성 검사(task completion check) 등 네 가지가 있다. 구문적 검사는 미리 정의된 XML 형태의 작업 계획 형식을 잘 준수하는 지 검사하는 것이며, 의미적 검사는 작업 계획 내 각 pick_and_place 행동마다 적법한 전조건들과 후조건들이 붙어 있는지 검사하는 것이다. 실행 가능성 검사는 작업 초기 상태에서 시작해서 계획에 기술된 행동들을 순서대로 실행해나간다고 가정하였을 때, 계획 내 특정 행동을 실행할 순서가 되었을 때 그 행동의 실행 전조건들이 만족될 수 있을 것인지, 또 해당 행동을 실행한 후에는 실행 후조건들이 만족될 수 있을 것인지를 검사하는 것이다. 끝으로 작업 완수 가능성 검사는 작업 초기 상태에서 시작해서 계획에 따라 행동들을 순서대로 실행하였을 때, 최종적으로 사용자의 지시를 충족하는 작업 목표 상태에 도달할 수 있을 것인지를 검사하는 것이다.

구문적 검사와 의미적 검사는 검증 대상인 작업 계획만 있

으면 충분히 수행이 가능하다. 하지만, 실행 가능성 검사와 작업 완수 가능성 검사를 위해서는 계획 검증기가 작업 초기 상태에서 시작해서 작업 계획의 각 행동을 가상으로 실행해가면서 이에 따른 작업 상태의 변화도 함께 추적할 수 있어야만 검사가 가능하다. 따라서 ChainBot 시스템의 계획 검증기는 내부적으로 작업 계획 행동들의 실행에 따른 작업 상태 변화를 상태 묘사 문장들의 집합으로 시뮬레이션한다. 즉 작업 초기 상태 묘사 문장들에서 시작하여 작업 계획의 각 행동이 실행되었을 때의 효과를 이 상태 묘사 문장들의 집합에 반영하여 지속적으로 변경해나간다.

표 5는 거대 언어 모델에 보내는 계획 검증 프롬프트의 한 예이다. 이 프롬프트에는 본래 작업 계획 문제(task planning problem) 정보와 검증 대상인 작업 계획(task plan) 외에, 실행 가능성 검사와 작업 완수 가능성 검사를 위해 계획 검증기가 내부적으로 시뮬레이션을 통해 생성한 행동 실행에 따른 작업 상태 전이(simulated state transitions) 정보도 함께 제공된다.

이어서 거대 언어 모델이 계획 검증에 이용할 구문적 검사, 의미적 검사, 실행 가능성 검사, 그리고 작업 완수 가능성 검

표 5. 계획 검증을 위한 거대 언어 모델 프롬프트의 예시.

Table 5. An example of LLM prompt for plan validation.

Your role is the plan validator ...
...
You should use the following information for plan validation:

1. The task planning problem
{task_planning_problem}
2. The task plan to be validated
{task_plan}
3. Simulated state transitions
{simulated_state_transitions}
4. Validation rules
{syntax_checking}
{semantics_checking}
{feasibility_checking}
{task_completion_checking}

You should show the validation results for each action package <apack> of the task plan in the following format:

apack number
* Syntax: Correct/Wrong (+reason if Wrong)
* Semantics: Correct/Wrong (+reason if Wrong)
* Feasibility: Success/Failure (+reason if Wrong)

You should also summarize the final report of the plan validation in the following format:

Final Report
- Syntax Check: Success/Failure (+ brief reasoning)
- Semantics Check: Success/Failure (+ brief reasoning)
- Feasibility Check: Success/Failure (+ brief reasoning)
- Task Completion: Success/Failure

표 6. 계획 검증 규칙들의 예시.

Table 6. some examples of plan validation rules.

1. Syntax checking
 - Check if each action package <apack> contains three parts: <precondition>, <action>, <postcondition>.
 - <action> has an exact one robot.pick_and_place("pick_target","place_target") action.
 - ...
2. Semantics checking
 - Check the correctness of the <precondition> and <postcondition> statements in the following way.
 - Exact two conditions must be included in <precondition> statements:
 - "The pick_target is empty on top."
 - "The place_target is empty on top."
 - Exact two conditions must be included in <postcondition> statements:
 - "The pick_target is placed on top of the place_target."
 - "The place_target is not empty on top."
 - ...
3. Feasibility checking
 - Check the satisfiability of the <precondition> statements based upon the Current State immediately before this <action> in the State transitions.
 - And then check the satisfiability of the <postcondition> statements based upon the Current State immediately after this <action> in the State transitions.
 - <action> is feasible only if both the <precondition> and <postcondition> statements are satisfied.
 - If infeasible, mark Wrong and explain briefly why.
 - ...
4. Task completion checking
 - Check whether the overall task required by the user instruction can be successfully accomplished if all actions in the given plan will be executed from the first to the last.
 - ...

사를 위한 검증 규칙들(validation rules)이 포함된다. 계획 검증에 사용된 각 검사별 구체적인 내용은 표 6의 예들과 같다.

한편, 표 5의 예에서 보듯이, 계획 검증을 위한 프롬프트의 후반부에는 각 행동 패키지 <apack>별로 수행한 구문적 검사, 의미적 검사, 그리고 실행 가능성 검사 각각의 결과를 출력하도록 요청하고, 각 행동 패키지별 검사가 모두 완료되면 모든 검사 결과들을 종합하여 최종 보고서(final report)를 출력하도록 요청한다.

이와 같은 계획 검증 프롬프트에 따라 계획 검증기가 작업 계획 검증 중에 오류를 발견하면, 표 7의 예와 같이 작업 계획 내의 어떤 행동 패키지에 대한 어떤 검사에서 오류가 발생했는지 뿐만 아니라, 오류로 판정한 이유(reason)도 함께 검증 결과로 출력한다. 그러면 이 검증 결과를 토대로 해당 오류들을 배제할 수 있는 새로운 작업 계획을 수립하도록 표 8의 예와 같은 재계획 프롬프트를 작업 계획 생성기(task planner)에게 보낸다. 이와 같은 과정을 통해 작업 계획 생성기는 재계획 프롬프트에 포함된 계획 검증 결과인 오류 정보들을 참고하여 작업을 다시 재계획함으로써, 해당 오류들을 피할 수 있는 올바른 작업 계획을 수립할 수 있게 된다.

표 7. 계획 검증 결과의 예시.

Table 7. An example of the plan validation result.

apack 1
* Syntax: Correct
* Semantics: Correct
* Feasibility: Correct
apack 2
* Syntax: Correct
* Semantics: Correct
* Feasibility: Correct
apack 3
* Syntax: Correct
* Semantics: Correct
* Feasibility: Wrong
Place infeasible: yellow bowl is not empty on top.
Final Report
* Syntax Check: Success
* Semantics Check: Success
* Feasibility Check: Failure
* Task Completion: Failure
Reason: The plan execution can be stopped due to an infeasible action.

표 8. 재계획을 위한 거대 언어 모델 프롬프트의 예시.

Table 8. An example of LLM prompt for task replanning.

...
You should re-plan the task for avoiding errors using the plan validation results on the previous task plan.
Refer to the following information for replanning:
1. The task planning problem {task_planning_problem}
2. The previous task plan {task_plan}
3. The validation results {validation_results}
...

5. 작업 계획 실행 중 오류 탐지와 복구

검증 과정을 거쳐 다양한 구문적, 논리적 오류들을 제거한 작업 계획은 작업 관리자(Task manager)를 경유해서 계획 실행기(Plan Executor)에게 전달되어 실제 시뮬레이션 작업 환경에서 로봇에 의해 실행된다. 계획 실행 중 발생 가능한 실행 오류들을 가능한 조기에 발견해서 복구하기 위해 ChainBot 시스템의 계획 실행기에서는 작업 계획을 처음부터 끝까지 한꺼번에 일괄 수행하지 않고 대신 작업 계획을 개별 행동(action) 단위로 나누어 1) CLIPort 기반의 행동 제어기(Action Controller)를 이용한 행동 실행과 2) GPT-5 기반의 오류 탐지(Error Detector)를 이용한 해당 행동의 전조건 및 후조건 검사를 번갈아 수행한다. 즉, 계획 실행기는 작업 계획 내 k번째

행동을 실행하기에 앞서 GPT-5 기반의 오류 탐지를 이용해 해당 행동의 실행 전조건들이 현재 작업 환경 상태에서 만족되는지 입력 카메라 영상을 토대로 판정하도록 한다. 그리고 만약 행동 실행을 위한 전조건들이 모두 만족된다면, CLIPort 기반의 행동 제어기를 이용해 해당 pick_and_place 행동을 시뮬레이션 환경에서 실행한다. 행동 실행이 완료된 후에는 다시

표 9. 작업 계획 실행 중 오류 탐지를 위한 거대 멀티모달 모델 프롬프트의 예시.

Table 9. An example of LMM prompt for error detection during task plan execution.

Your role is the run-time error detector which detects incorrect states by checking its preconditions and postconditions based on input camera images just before and after the given action of a task plan is executed.

...

You should use the following information for error detection:

1. The action to be executed
{pick_and_place_action}
2. The conditions of the action
{pre_or_post_conditions}
3. The camera image before or after the action execution
{camera_image}

Based on the camera image, you should check the satisfiability of the given action conditions.

If you find any error state that do not satisfy the given conditions, notify the error detection together with the natural language descriptions of the current state in the following format:

{error_detection_notification}
{error_state_descriptions}

Otherwise, notify briefly the condition satisfaction in the following format:

{condition_satisfaction_notification}

표 10. 복구 계획 수립을 위한 거대 언어 모델 프롬프트의 예시.

Table 10. An example of LLM prompt for recovery planning.

...

Due to the following error detection during plan execution, the task has stopped.

{error_detection_notification}

Therefore, you should re-plan the task starting from the error state using the following information:

1. The original task planning problem
{task_planning_problem}
2. The error state descriptions
{error_state_descriptions}

...

GPT-5 기반의 오류 탐지를 이용해 해당 행동의 실행 후조건들이 만족되는지 새로운 입력 카메라 영상을 토대로 판정하도록 한다.

표 9는 각 행동 실행 이전과 이후에 오류 상태 발견을 위해 GPT-5와 같은 거대 멀티모달 모델에게 행동 전조건들 혹은 후조건들을 검사하도록 요청하는 오류 탐지 프롬프트의 한 예를 나타낸다. 이와 같은 프롬프트를 토대로 각 행동 실행 이전과 이후의 작업 상태들에 대한 영상 기반의 검사들을 통해 예상과는 다른 비-정상적인 상태가 발견되면, 표 9의 예처럼 오류 탐지기는 오류 발생 통지(error detection notification)와 더불어 그때 오류 상태에 대한 자연어 묘사(error state descriptions)도 함께 계획 실행기에 보낸다. 그러면 작업 계획기는 해당 작업 계획의 실행을 즉시 멈추고, 표 10의 예와 같이 오류 상태 묘사를 프롬프트에 포함시켜 해당 오류 상태에서부터 시작해서 사용자 지시가 요구하는 작업 목표 상태에 도달할 수 있는 복구 작업 계획을 다시 수립하도록 작업 계획기에게 요청한다.

이와 같이 작업 계획 실행 중 각 행동의 실행 전후에만 전조건들과 후조건들을 검사하는 실행 오류 탐지 방식으로는 작업 계획 실행 중 발생 가능한 다양한 종류의 오류들을 모두 발견해내기는 사실상 어렵다. 또한 각 개별 행동을 실행하고 있는 중간에 빠른 오류 발견과 복구도 어렵다. 하지만, 현실적으로 현재 ChainBot 시스템의 작업 실행 환경에서는 각 행동의 실행 시간이 비교적 길지 않고 오류 발생 빈도도 크게 높지 않다는 점, 또 만약 각 행동 실행 중에도 몇 초 주기로 GPT-5와 같은 유료 외부 멀티모달 모델을 이용해 오류 검사를 계속 수행한다면 발생할 비용과 작업 지연 시간 등을 고려할 때, 현재 ChainBot 시스템이 채택하는 오류 탐지 방식은 비용 대비 작업 성공률 면에서 높은 효율성을 기대해 볼 수 있다. 한편, PyBullet으로 구현된 시뮬레이션 작업 환경에서 취득하는 카메라 영상은 일반적으로 해상도가 낮아서 이를 토대로 오류 상태를 탐지해내는 GPT-5의 성능에 한계가 있다. 따라서 이 문제를 극복하기 위해 본 연구에서는 RealESRGAN 기반의 영상 업스케일링(upsampling) 기법[42]을 적용한다. 구체적으로 ChainBot 시스템에서는 입력 카메라 영상에 RealESRGAN_x2plus 모델을 적용하여 선명도와 디테일을 향상시키며, 이 과정을 통해 원본 영상 대비 시각적 품질을 크게 개선함으로써 비교적 안정적인 GPT-5의 오류 탐지 성능을 얻을 수 있다.

IV. 구현 및 실험

1. 시스템 구현

ChainBot 시스템을 구성하는 각 모듈들과 이들 간의 상호작용은 Python 기반의 에이전트 개발 프레임워크 LangChain 0.3.27과 LangGraph 0.2.76을 이용해 구현하였다. 그리고 작업 계획 생성 모듈과 계획 검증 모듈의 후단부(backend)로는 거대 언어 모델인 Google Gemini 2.0-Flash를 이용하였으며, 계획 실행 중 오류 탐지 모듈의 후단부로는 거대 멀티모달 모델인 OpenAI GPT-5를 이용하였다. 또한 PyBullet로 구현된 도메인별 시뮬레이션 작업 환경들에서의 물체 조작 pick_and_place 행동 실행 구현은 사전 학습된(pretrained) CLIPort 모델을 활용하였다. 마지막으로 ChainBot 시스템은 GeForce RTX 2060 GPU 1대로 구성된 컴퓨터 환경에서 성능 테스트 실험들을 진행하였다.

2. 성능 평가 실험

첫 번째 정량적 실험은 ChainBot 시스템의 프롬프트 문맥(prompt context) 구성 방식이 작업 계획 성능에 미치는 영향을 평가하기 위한 것이다. ChainBot은 프롬프트 문맥에 사용자 지시(user instruction), 작업 환경의 주요 물체들(objects), 수행 가능한 물체 조작 행동들(actions), 그리고 작업 초기 상태 묘사(initial state description) 등의 필수 요소들 외에도, 작업 계획 예시들(examples)과 작업 도메인별 경험적 지식(domain heuristic knowledge) 등도 추가로 포함시킨다. 표 11은 서로 다른 프롬프트 문맥 구성을 적용했을 때 생성된 작업 계획의 정확도(plan correctness, PC)를 비교한 결과를 보여준다. 계획 정확도(PC)는 생성된 작업 계획이 정답 작업 계획(ground-truth task plan)과 구성 행동들과 그들의 순서, 그리고 각 행동의 전조건, 후조건 측면에서 얼마나 일치하는지를 평가하는 지표이다. 이 실험에서는 각 도메인 별로 난이도가 다른 10개씩의 작업들로 구성된 총 30개의 작업들로 실험을 진행하였으며, 작업 계획 생성에는 Google Gemini-2.0을 사용하였다. 성능 평가 실험에 사용된 각 도메인별 작업들의 구체적인 명세(specification)는 공개 드라이브에서 확인할 수 있다.

표 11의 결과를 살펴보면, 먼저 ChainBot과 같이 계획 수립 프롬프트에 환경 물체, 행동, 초기 상태 묘사 외에 작업 계획 예시들(examples)과 작업 영역별 경험적 지식(domain heuristic knowledge)을 함께 포함시킨 경우가 PBB, SB, MBB 도메인 작업들 각각 계획 정확도(PC) 100%, 90%, 100%를 기록하여 가장 높은 성능을 보여주었다.

한편, 작업 계획 사례(examples)만 추가로 제공한 경우가 도메인별 경험적 지식만 추가로 제공한 경우보다 세 가지 도메인 작업들 모두에서 약간 더 높은 계획 정확도를 보였다. 이러한 결과는 주로 도메인별 추상적인 문제풀이 전략(domain-specific problem-solving strategy)들을 나타내는 경험적 지식들에 비해, 유사한 다른 작업들의 계획 사례는 더 구체적이고 세부적인 작업별 계획 수립 정책(task-specific planning policy)을 제공할 수 있을 뿐만 아니라, 작업 계획 표현(plan representation)을 위한 구체적인 사례도 함께 제공할 수 있기 때문으로 판단된다.

또 부속 목표(subgoal) 간의 간섭이 거의 없는 PBB 도메인에 비해, 쌓아야 할 블록들 간의 강한 상호작용들이 존재하는 어려운 작업들인 SB와 MBB 도메인에서 상대적으로 조금 더 낮은 계획 정확도를 보여주고 있다. 심지어 PBB 도메인 작업

표 11. 작업 계획 수립을 위한 프롬프트 문맥 구성 방식별 성능 비교.

Table 11. Performance comparison between different prompt contexts for task planning.

Prompt Contexts				Task Domains		
Objects, Actions	Initial State Des.	Plan Examples	Domain Heuristics	PBB	SB	MBB
○				90	20	30
○	○			90	50	60
○	○	○		100	80	90
○	○		○	100	70	80
○	○	○	○	100	90	100

들에서는 별도의 작업 계획 사례들과 추가적인 경험적 지식은 물론이고 일반적으로 계획 수립에 꼭 필요한 초기 상태 묘사(initial state description)마저 제공하지 않았을 때에도 90%의 높은 계획 정확도를 나타내었다. 결과적으로, 난이도가 높은 작업일수록 ChainBot과 같이 작업 계획 예시와 도메인별 경험적 지식을 프롬프트 문맥에 함께 제공하는 것이 계획 정확도 향상에 효과적임을 알 수 있다.

두 번째 실험은 ChainBot 시스템의 작업 실행 전 계획 검증(plan validation) 및 재계획(replanning)이 작업 계획 정확도 향상에 미치는 영향을 분석하기 위한 것이다. 본 실험에서는 표 12과 같이 서로 다른 조합의 검사들로 계획 검증을 수행하고 오류 제거를 위해 재계획하였을 때, 작업 계획들의 정확도가 어느 정도 개선될 수 있는지 최종 작업 계획들의 정확도(PC)를 측정해보았다. 이 계획 검증 실험에 이용한 작업 계획들은 계획 생성 실험 결과표인 표 11의 두 번째 행(row)에 해당하는 것들로서, 별도의 작업 계획 사례들이나 경험적 지식을 추가로 제공하지 않은 상태에서 생성된 작업 계획들이다. 따라서 이들은 표 11의 결과와 같이, PBB, SB, MBB 세 가지 도메인 작업들에서 각각 10%, 50%, 40%의 오류 계획(incorrect plan)들을 포함하고 있다.

표 12의 실험 결과들을 살펴보면, 기존 작업 계획들의 정확도(90%, 50%, 60%)에 비해 계획 검증 단계에서 단순히 구문적 검사(syntax check)와 의미적 검사(semantics check)만을 수행하였을 때는 세 가지 도메인 작업들 모두에서 작업 계획의 품질에 큰 변화를 발견할 수 없으나, 작업 계획의 실행 가능성 검사(feasibility check)와 사용자 지시 달성 검사(completion check)까지 수행했을 때는 작업 계획의 정확도가 각각 100%, 90%, 90%까지 크게 개선된 것을 확인할 수 있다. 특히 PBB와 같은 난이도가 비교적 낮은 작업들에서는 계획 검증과 오류 발견시 재계획을 통해 100% 계획 정확도를 확보할 수 있었다. 하지만, SB와 MBB 도메인 작업들의 경우, 거대 언어 모델(LLM)을 이용한 ChainBot 시스템의 현재 계획 검증으로는 여전히 약 10%의 작업 계획 오류를 해결하지 못했다는 것도 본 실험을 통해 알 수 있다. 이와 같은 실험 결과들을 종합해볼 때, 작업 계획 실행에 앞서 ChainBot 시스템이 수행하는 계획 검증 방식과 재계획 과정이 계획 품질 개선에 미치는 긍정적 효과를 확인할 수 있다.

세 번째 실험은 ChainBot 시스템이 작업 계획 실행 중에 수행하는 오류 탐지(error detection) 및 복구(recovery) 방식의 성능을 평가하기 위한 실험이다. 앞서 3절에서 설명한 것과 같이,

표 12. 계획 검증 방식들 간의 성능 비교.

Table 12. Performance comparison between different plan validation methods.

Plan Validation				Task Domains		
Syntax Check	Semantics Check	Feasibility Check	Completion Check	PBB	SB	MBB
○				90	50	60
○	○			90	50	60
○	○	○		100	60	80
○	○	○	○	100	90	90

ChainBot 시스템에서는 작업 계획 실행 중에 실시간 작업 상태 영상을 토대로 계획 내 각 행동의 전조건(precondition)들과 후조건(postcondition)들의 만족 여부를 체크함으로써 오류를 탐지해내고, 오류를 발견하면 해당 오류 상태에서 남은 작업을 재계획하여 수행함으로써 오류를 복구하는 방식을 취한다. 본 실험에서는 각 행동의 전조건만 체크하는 방식, 후조건만 체크하는 방식, 그리고 ChainBot 시스템과 같이 전조건과 후조건을 모두 체크하는 방식 등 서로 다른 오류 탐지 방식들을 서로 비교해본다. 또한 오류 발견 시 해당 계획을 처음부터 다시 새로 실행해보는 방식(ReDo), ChainBot 시스템과 같이 오류 상태에서 시작해서 남은 작업을 새로 재계획하여 실행해보는 방식(RePlan) 등 서로 다른 오류 복구 방식들도 비교해본다. 계획의 불완전성과 같은 다른 요인들에 의한 오류 가능성은 배제하고 오직 실행 오류에 대한 탐지 성능만을 분석해보기 위해, 이 실험에서는 모두 논리적으로 올바른 최적의 작업 계획들을 실험 대상으로 삼았다.

구체적으로 본 실험에서는 CLIPort를 이용해 계획 내 각 행동을 차례대로 실행하면서 변화하는 실시간 작업 상태 영상을 사이드-뷰 카메라(side-view camera)로 포착한 후 GPT-5 기반의 오류 탐지 모듈을 적용하여 행동의 전조건 및 후조건의 만족 여부를 체크하였고, 오류 복구를 위한 재계획(RePlan)은 Gemini-2.0 기반의 작업 계획 생성 모듈을 이용하였다. 또 이 실험에서는 오류 탐지 및 복구 과정을 포함한 작업 실행 후, 최초 사용자 지시의 달성 여부를 기준으로 작업의 성공과 실패를 판정하였고, 각 방식들의 성능은 평균 작업 성공률(success rate, SR)로 측정하였다. 또 각 작업 당 재실행(ReDo)을 통한 오류 복구의 최대 허용 횟수도 3회로, 재계획(RePlan)을 통한 오류 복구의 최대 허용 횟수도 3회로 제한하였다. 마지막으로 본 실험은 실세계 물리 환경과는 달리 PyBullet으로 구현된 로봇 시뮬레이션 환경에서 수행됨으로써, 행동 실행 오류 발생이 거의 희소하다는 한계점이 있다. 따라서 본 실험에서는 미리 설정된 오류 발생율에 따라 인위적으로 각 행동의 실행 오류를 발생시켜 실험을 진행하였다. 본 실험에서 설정한 오류 발생율은 0.1이다.

표 13의 실험 결과들을 살펴보면, 먼저 ChainBot 시스템과 같이 각 행동의 전조건과 후조건들을 모두 검사하는 오류 탐지 방식과 재계획을 통한 오류 복구 방식을 적용한 경우가 모든 도메인의 작업들에서 가장 높은 100% 작업 성공률을 보여준 것을 알 수 있다. 한편 어떤 복구 방식을 썼느냐에 크게 관계

표 13. 계획 실행 중 오류 탐지와 복구 방식들 간의 성능 비교.

Table 13. Performance comparison between different error detection and recovery methods during plan execution.

Error Detection		Recovery		Task Domains		
Precondition Check	Postcondition Check	ReDo	RePlan	PBB	SB	MBB
○		○		60	60	50
			○	70	60	40
	○	○		80	70	80
			○	80	80	100
○	○	○		80	80	80
			○	100	100	100

없이 각 행동의 후조건만 검사한 경우가 전조건만을 검사한 경우에 비해, 조금 더 높은 작업 성공률을 나타낸다. 추측해볼 수 있는 원인으로는 행동의 후조건으로는 해당 행동의 실행 성공 여부를 직접 검사할 수 있으나, 행동의 전조건만으로는 해당 행동의 실행 가능성을 검사할 수 있어도 실행 결과를 직접 검사할 수 없다는 점, 또 본 실험에서는 모두 논리적으로 인과관계 순서관계가 올바른 작업 계획들을 실험에 사용하였기 때문에 계획 내 각 행동의 실행 결과만 행동의 후조건로 검사하면 전체 작업의 성공을 보장하기 용이하다는 점 등이 있다.

재실행(ReDo)과 재계획(RePlan) 오류 복구 방식들 간의 성능을 비교해보면, 전체적으로 ChainBot 시스템과 같이 재계획을 통한 오류 복구 방식을 적용한 경우가 재실행을 적용한 경우보다 더 높은 작업 성공률을 보여준 것을 확인할 수 있다. 이는 재실행이 동일 계획을 반복하여 오류가 재발할 가능성이 높은 반면, 재계획은 오류 상태에서 남은 작업만 새로 계획하기 때문으로 판단된다.

하지만 행동의 전조건만 검사하는 오류 탐지 방식의 경우, 앞서 설명한대로 각 행동의 실행 실패 성공 여부를 직접 확인할 수 없다는 점과 더불어 GPT-5 기반의 오류 탐지 모듈과 Gemini-2.0 기반의 재계획 모듈에도 실질적으로 존재하는 소량의 탐지 오류 및 재계획 오류 등이 나쁜 영향을 주어서 MBB 도메인 작업들에 대해서는 재계획 오류 복구 방식의 작업 성공률이 40%로 재실행 오류 복구 방식보다 조금 더 낮은 성능을 보여주었다. 이 결과를 종합하면, ChainBot 시스템에서 채택한 계획 실행 중 행동의 전조건과 후조건 체크를 통한 오류 탐지 방식과 재계획을 이용한 오류 복구 방식의 우수성을 확인할 수 있다.

네 번째 정량적 실험은 (1) 풍부한 문맥 정보를 이용한 작업 계획 생성(task planning with rich contexts) 모듈, (2) 작업 실행 전 계획 검증과 재계획(plan validation and replanning) 모듈, (3) 작업 실행 중 오류 탐지와 복구(error detection and recovery) 모듈 등 ChainBot 시스템의 3 가지 핵심 기술 모듈들을 서로 다르게 구성한 시스템들의 성능을 비교해봄으로써, 각 모듈이 작업 성능에 미치는 효과를 분석해보는 실험이다. 이 실험에서는 표 14와 같이 작업 계획 생성 모듈에서 별도의 문맥 정보를 이용하지 않을 뿐만 아니라, 작업 실행 전 계획 검증과 재계획 모듈도, 작업 실행 중 오류 탐지와 복구 모듈도 포함하지 않는 첫 번째 시스템의 경우를 포함해, 총 4 가지 서로 다른 모듈 구성을 가진 시스템들의 최종 작업 성공률(success, SR)을 비교해본다. 앞선 표 13의 정량적 실험들 때와 마찬가지로, 이 실험에서도 작업 계획 실행시에 인위적인 오류 발생률을 0.1로 설정하여 실험한다. 또 이 실험에서는 모든 작업들에 대해 각 작업 당 작업 실행 전 계획 검증에서 오류 발견시 재계획의 시도 횟수도, 작업 실행 중 오류 탐지 시 복구 시도 횟수도 각각 최대 3회로 제한하여 실험한다.

표 14의 실험 결과들을 살펴보면, 예상한대로 ChainBot 시스템과 같이 3 가지 핵심 기술 모듈들을 모두 채용한 시스템의 경우가 PBB, SB, MBB 도메인별로 각각 100%, 80%, 90%로 가장 높은 작업 성공률을 보여준다. 이 경우는 3 가지 핵심 기술 모듈들을 전혀 적용하지 않은 첫 번째 시스템의 경우와

표 14. 서로 다른 시스템 구성들 간의 성능 비교.

Table 14. Performance comparison between different system configurations.

Task Planning with Rich Contexts	Plan Validation & Replanning	Error Detection & Recovery	Task Domains		
			PBB	SB	MBB
			50	20	40
○			50	40	60
○	○		70	50	60
○	○	○	100	80	90

비교해볼 때 각 도메인별로 $100 \times (100-50)/50 = 100\%$, $100 \times (80-20)/20 = 300\%$, $100 \times (90-40)/40 = 125\%$ 의 큰 성능 향상을 나타낸 것이다.

또 본 연구에서 제안한 각 핵심 기술 모듈들이 추가됨에 따라 각 도메인의 작업 성공률은 전체적으로 조금씩 증가한 것을 알 수 있다. 그리고 3 가지 작업 도메인들 모두에서 드러난 공통점은 작업 계획 실행 중 오류 탐지와 복구 모듈이 추가되었을 때가 가장 큰 폭의 성능 향상을 보여준 것이다. 이 결과를 통해 세 가지 핵심 기술 모듈들이 모두 작업 성공률에 긍정적인 영향을 주었지만, 그 중에서도 작업 실행 중 오류 탐지와 복구가 작업 성공에 가장 큰 도움을 준 것으로 판단된다. 한편, 작업 계획 생성 난이도가 비교적 낮은 PBB 도메인에서는 계획 생성을 위한 문맥 정보가 성능 향상에 별 도움이 되지 못했고, 작업 계획의 길이가 상대적으로 긴 MBB와 SB 도메인에서는 계획 검증과 재계획 모듈이 다른 모듈들에 비해 작업 성공에 미치는 영향이 크지 않았음을 알 수 있다. 이 결과를 종합해보면, 본 연구에서 제안한 ChainBot 시스템의 핵심 기술 모듈들이 모두 작업 성공률 개선에 도움을 준다는 것을 재차 확인할 수 있다.

다음 실험은 본 연구의 ChainBot 시스템이 작업 계획 생성(task planning)을 위해 채택한 거대 언어 모델(LMM)인 Google의 Gemini-2.0을 다른 거대 언어 모델과 성능 비교를 해보기 위한 실험이다. 이 실험에서는 대표적인 오픈 소스 거대 언어 모델인 Meta의 Llama3.1과 Google의 Gemini-2.0가 생성한 작업 계획의 정확도(PC)를 비교해본다.

표 15의 실험 결과들을 살펴보면, ChainBot 시스템에서 채용한 Google Gemini-2.0에 비해 Meta의 Llama3.1은 훨씬 낮은 성능을 보여준다. PBB와 SB 도메인의 작업들에서 Gemini-2.0이 100%의 계획 정확도를 보여준데 반해, Llama3.1은 각각 60%, 20%의 계획 정확도를 보여주는데 그쳤다. 또 난이도가 가장 높은 MBB 도메인 작업들에 대해서는 성능 차이가 더 벌어져, Gemini-2.0은 80%의 계획 정확도를 보였으나 Llama3.1

표 15. 작업 계획 수립을 위한 거대 언어 모델들 간의 성능 비교.

Table 15. Performance comparison between different large language models for task planning.

Large Language Models	Task Domains		
	PBB	SB	MBB
Llama3.1-8b-instruct	60	20	10
Gemini-2.0	100	100	80

은 단 10%의 계획 정확도를 보였다. 그 결과 Llama3.1은 무료 서비스라는 장점에도 불구하고 ChainBot의 작업 계획기(task planner) 역할에는 한계가 있음을 확인하였다. 따라서 현재 상황을 감안하면 ChainBot 시스템의 작업 계획 생성을 위해 Google Gemini-2.0을 선택한 결정은 타당하다고 볼 수 있다.

다음 실험은 ChainBot 시스템에서 작업 공간 내의 물체들을 탐지(object detection)해내기 위해 채용한 시각-언어 모델(vision-language model)인 Google의 Gemini-2.5 Flash를 다른 전용 시각 모델(vision model)과 성능을 비교하기 위한 실험이다. 이 실험에서는 대표적인 개방형-어휘 물체 탐지 모델(open-vocabulary object detection model)인 ViLD와 Gemini-2.5 Flash의 성능을 비교해보고, 작업 영상별 물체들의 평균 인식 정확도인 mAP (mean average precision)를 성능 지표로 사용한다. 표 16은 세 가지 도메인별로 10개씩 총 30개의 작업들에 대해, 사이드 뷰(side view) 카메라 영상을 이용해 각 작업 환경에 매우 다양하게 배치되어있는 각기 다른 색상의 그릇(bowl)들과 블록(block)들을 탐지해본 결과들을 나타낸다.

표 16의 실험 결과들을 살펴보면, 먼저 모든 도메인의 작업들에서 본 ChainBot에서 채용한 시각-언어 모델인 Google Gemini-2.5 Flash가 물체 탐지 전용 시각 모델인 ViLD에 비해 매우 높은 인식 성능을 보여준 것을 확인할 수 있다. 상대적으로 ViLD는 예상외로 각 도메인별로 약 50% 내외의 매우 낮은 인식 정확도를 보여주었다. 이 실험 결과표에는 포함하지 않았으나, Grounding DINO-v1.5, YOLO-v11 등 다른 물체 탐지 전용 시각 모델들도 ViLD와 마찬가지로 ChainBot 작업 환경들에서 요구하는 충분한 인식 성능을 보여주지 못하였다. 따라서 현재 상황에서는 물체 탐지 전용 시각 모델을 대신 시각-언어 모델인 Google Gemini-2.5 Flash를 채용하기로 한 ChainBot 시스템의 결정을 이해할 수 있다.

다음 실험은 ChainBot 시스템에서 작업 환경 영상으로부터 자연어 상태 묘사 문장들을 생성(state description generation)해내는 역할을 담당하는 OpenAI의 GPT-5를 다른 멀티모달 모델과 성능을 비교하기 위한 실험이다. 이 실험에서는 OpenAI GPT-5를 또 다른 거대 멀티모달 모델인 Google Gemini-2.5

표 16. 작업 공간 물체 탐지를 위한 시각 모델들 간의 성능 비교.

Table 16. Performance comparison between vision models for workspace object detections.

Object Detection Models	Task Domains		
	PBB	SB	MBB
ViLD	56.33	39.32	48.04
Gemini-2.5 Flash	92.12	97.01	89.05

표 17. 상태 묘사 생성을 위한 거대 멀티모달 모델들 간의 성능 비교.

Table 17. Performance comparison between large multimodal models for state description generation.

Large Multimodal Models	Task Domains		
	PBB	SB	MBB
Gemini-2.5 Flash	94.28	81.40	71.79
GPT-5	100.00	93.02	82.05

Flash와 성능을 비교해보며, 작업 영상별 생성된 문장들의 평균 정확도인 mAP를 성능 평가 지표로 사용한다.

표 17의 실험 결과들을 살펴보면, 먼저 모든 도메인의 작업들에서 본 ChainBot에서 채용한 OpenAI GPT-5가 비교 대상인 Google Gemini-2.5 Flash에 비해 더 높은 상태 묘사 정확도를 보여준 것을 알 수 있다. 그리고 두 모델 모두 SB 도메인과 같이 비슷한 색상의 블록들이 여러 개 상하로 쌓여 있거나 MBB 도메인과 같이 많은 수의 블록들이 동일 혹은 유사 색상의 그릇들 안에 놓여 있는 복잡한 작업 환경들에서 상태 묘사 정확도가 상대적으로 감소한 것도 확인할 수 있다. 본래 이 영상 기반 상태 묘사 생성은 컴퓨터 비전 분야에서 잘 알려져 있는 영상 캡션 생성(image captioning) 작업의 하나로서, 이 작업을 위한 다양한 전용 모델들이 존재하고 있다. 하지만 ChainBot 작업 환경에서 요구하는 정확한 작업 상태 묘사를 생성하기 위해서는 입력 영상(input image) 이외에 묘사 대상 물체들의 목록(object list)과 더불어 묘사 문장 서술 방식(description syntax), 그리고 제약 조건(constraint) 등을 추가로 입력으로 제공할 수 있어야 하고 모델은 이러한 다양한 형태의 입력 데이터들을 모두 이해할 수 있어야 하는데, 현재 대부분의 영상 캡션 생성 전용 모델들은 이러한 기능을 제공하고 있지 않다. 따라서 영상과 텍스트들로 이루어진 멀티 모달 입력 데이터들로부터 형식에 맞는 정확한 정면 설명문을 생성하기 위해서는 결국 본 실험의 OpenAI GPT-5, Google Gemini-2.5 Flash와 같은 거대 멀티모달 모델들을 선택할 수 밖에 없었다. 하지만 표 17의 실험 결과들에서 보듯이, 상태 묘사의 정확도가 매우 중요한 ChainBot 작업 환경에서 Gemini-2.5 Flash와 같은 다른 멀티모달 모델들로는 충분한 정확도를 확보하기 어려워, GPT-5 선택이 최선임을 확인하였다.

다음 실험은 작업 계획 실행 중 오류 탐지(error detection)를 위해 ChainBot 시스템에서 채용한 OpenAI GPT-5를 nstructBlip-Vicuna-7b, Llava-Llama-3-8b 등 다른 시각-언어 모델들과 성능을 정량적으로 비교하기 위한 실험이다. 본 실험에서는 CLIPort를 이용해 작업 계획 상의 행동들을 실제로 수행해가면서 각 행동 실행 단계별 정상 상태의 환경 영상들과 더불어 오류 상태의 영상들을 대상으로 각 행동의 전조건과 후조건들을 검사하며 이들의 평균 정확도(mAP)로 각 모델의 오류 탐지 성능을 평가한다.

작업 환경의 입력 영상을 토대로 텍스트로 정의된 각 행동의 전조건과 후조건의 만족 여부를 검사하는 일은 컴퓨터 비전의 영상 기반 질문-응답(visual question-answering, VQA) 작업과 유사하다. 따라서 공개되어 있는 다양한 영상 기반 질문-응답(VQA) 전용 시각-언어 모델들이 있다. 하지만 이 경우에도

표 18. 계획 실행 중 오류 탐지를 위한 시각-언어 모델들 간의 성능 비교.

Table 18. Performance comparison between vision-language models for error detection during plan execution.

Vison-Language Models	Test Domains		
	PBB	SB	MBB
InstructBlip-Vicuna-7b	53.16	56.57	53.52
Llava-Llama-3-8b	75.32	72.73	74.65
GPT-5	84.81	86.54	85.91

질문-응답(VQA) 전용 시각-언어 모델들을 ChainBot 작업 환경에 적용해본 결과 충분한 성능을 얻기 어려워서, 본 실험에서는 InstructBlip-Vicuna-7b, Llava-Llama-3-8b, GPT-5와 같이 거대 언어 모델을 확장한 거대 멀티모달 모델들로 후보를 제한하였다.

표 18은 이 실험의 결과들을 나타낸다. 세 가지 서로 다른 도메인의 작업들에서 OpenAI GPT-5가 상대적으로 다른 모델들에 비해 가장 높은 오류 탐지 성능을 나타내었다. 반면에, Llava-Llama-3-8b는 약 70%대, InstructBlip-Vicuna-7b는 약 50%대의 낮은 오류 탐지 성능을 보여주었다. 따라서 현재는 ChainBot 시스템의 계획 실행 중 오류 탐지를 위해 가용한 기저 모델들 중에서 GPT-5를 선택할 수 밖에 없는 상황이다. 하지만 현재의 GPT-5 탐지 성능으로도 ChainBot 시스템의 올바른 운영이 어렵기 때문에, 앞서 3절 ChainBot 시스템 설계에서 설명한대로 다른 신경망 모델인 RealESRGAN_x2plus의 추가 도입을 통한 입력 영상의 해상도 및 품질 개선, 카메라 포즈의 미세 조정 등 다양한 개선책을 ChainBot 시스템 구현에 추가로 적용하였다.

다음은 몇 가지 대표적인 작업 사례들을 중심으로 ChainBot 시스템의 특징과 한계를 정성적으로 분석해본다. 첫 번째 작업 사례는 표 19 (a)와 같은 초기 상태에서 시작하여 사용자의 자연어 지시 “Stack the blocks bottom-to-top as brown block, yellow block, cyan block, and green block.”를 완수해야 하는 SB 도메인 작업의 하나이다.

ChainBot 시스템의 작업 계획 생성기가 최초로 생성한 작업 계획은 표 20 (a)와 같고, 이 계획에 대해 계획 검증기는 표 20 (b)와 같이 이 계획으로는 사용자 지시를 달성할 수 없다고 (task completion: failure) 판정하여 1차 재계획 과정을 수행함으로써 표 20 (c)와 같은 재계획 결과를 얻었다. 하지만 이 재계획 결과마저도 표 20 (d)와 같이 2차 계획 검증 과정에서 5 번째 행동 robot.pick_and_place(“green block”, “brown block”)의 전조건(precondition)인 “The brown block is empty on top.”이 만족되지 못해 실행 불가능(feasibility: wrong)하다는 판정을 내린다.

표 19. ChainBot 시스템의 정성적 평가: 사례 A.

Table 19. Qualitative analysis of ChainBot system: case A.

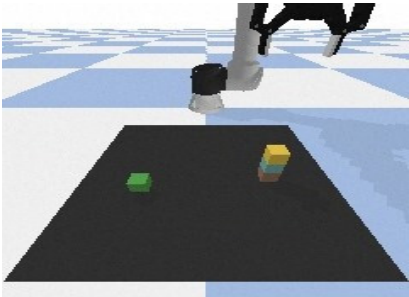

(a) The initial state.
“Stack the blocks bottom-to-top as brown block, yellow block, cyan block, and green block.”
(b) User instruction.

표 20. 작업 계획 생성, 검증과 재계획: 사례 A.

Table 20. Task planning, validation and replanning: case A.

<pre> <plan> <sequence> <apack> <precondition>The yellow block is empty on top.</precondition> <precondition>The top left corner is empty on top.</precondition> <action>robot.pick_and_place("yellow block", "top left corner")</action> <postcondition>The yellow block is placed on top of the top left corner.</postcondition> <postcondition>The top left corner is not empty on top.</postcondition> </apack> <apack> <precondition>The cyan block is empty on top.</precondition> <precondition>The green block is empty on top.</precondition> <action>robot.pick_and_place("cyan block", "green block")</action> <postcondition>The cyan block is placed on top of the green block.</postcondition> <postcondition>The green block is not empty on top.</postcondition> </apack> <apack> <precondition>The yellow block is empty on top.</precondition> <precondition>The brown block is empty on top.</precondition> <action>robot.pick_and_place("yellow block", "brown block")</action> <postcondition>The yellow block is placed on top of the brown block.</postcondition> <postcondition>The brown block is not empty on top.</postcondition> </apack> <apack> <precondition>The cyan block is empty on top.</precondition> <precondition>The yellow block is empty on top.</precondition> <action>robot.pick_and_place("cyan block", "yellow block")</action> <postcondition>The cyan block is placed on top of the yellow block.</postcondition> <postcondition>The yellow block is not empty on top.</postcondition> </apack> </sequence> </plan> </pre>
(a) The initial task plan.
<pre> validation result: apack 1 * Syntax: Correct * Semantics: Correct * Feasibility: Correct apack 2 * Syntax: Correct * Semantics: Correct * Feasibility: Correct apack 3 * Syntax: Correct * Semantics: Correct * Feasibility: Correct apack 4 * Syntax: Correct * Semantics: Correct * Feasibility: Correct Final Report - Syntax Check: Success - Semantics Check: Success - Feasibility Check: Success - Task Completion: Failure The user instruction is to stack blocks in the order brown, yellow, cyan, green from bottom to top. The final stack is brown, yellow, cyan. The green block is not on the stack. </pre>
(b) The result of the first plan validation.
<pre> <plan> <sequence> <apack> <precondition>The yellow block is empty on top.</precondition> <precondition>The top left corner is empty on top.</precondition> <action>robot.pick_and_place("yellow block", "top left corner")</action> <postcondition>The yellow block is placed on top of the top left corner.</postcondition> <postcondition>The top left corner is not empty on top.</postcondition> </apack> <apack> <precondition>The cyan block is empty on top.</precondition> <precondition>The green block is empty on top.</precondition> <action>robot.pick_and_place("cyan block", "green block")</action> <postcondition>The cyan block is placed on top of the green block.</postcondition> <postcondition>The green block is not empty on top.</postcondition> </apack> <apack> <precondition>The yellow block is empty on top.</precondition> <precondition>The brown block is empty on top.</precondition> <action>robot.pick_and_place("yellow block", "brown block")</action> <postcondition>The yellow block is placed on top of the brown block.</postcondition> <postcondition>The brown block is not empty on top.</postcondition> </apack> <apack> <precondition>The cyan block is empty on top.</precondition> <precondition>The yellow block is empty on top.</precondition> <action>robot.pick_and_place("cyan block", "yellow block")</action> <postcondition>The cyan block is placed on top of the yellow block.</postcondition> <postcondition>The yellow block is not empty on top.</postcondition> </apack> <apack> <precondition>The green block is empty on top.</precondition> <precondition>The brown block is empty on top.</precondition> <action>robot.pick_and_place("green block", "brown block")</action> <postcondition>The green block is placed on top of the brown block.</postcondition> <postcondition>The brown block is not empty on top.</postcondition> </apack> </sequence> </plan> </pre>
(c) The result of the first replanning.

```

validation result: apack 1
* Syntax: Correct
* Semantics: Correct
* Feasibility: Correct

apack 2
* Syntax: Correct
* Semantics: Correct
* Feasibility: Correct

apack 3
* Syntax: Correct
* Semantics: Correct
* Feasibility: Correct

apack 4
* Syntax: Correct
* Semantics: Correct
* Feasibility: Correct

apack 5
* Syntax: Correct
* Semantics: Correct
* Feasibility: Wrong (Reason: The brown block is not empty on top in the current state.)

Final Report
- Syntax Check: Success (All apacks have correct syntax.)
- Semantics Check: Success (All apacks have correct semantics.)
- Feasibility Check: Failure (apack 5 is infeasible.)
- Task Completion: Failure (The final stack is [brown block, yellow block, cyan block] on the table, and the green block is also on the table. The goal stack is [brown block, yellow block, cyan block, green block]. The green block is not on top of the cyan block.)

```

(d) The result of the second plan validation.

```

<plan>
<sequence>
  <apack>
    <precondition>The yellow block is empty on top.</precondition>
    <precondition>The top left corner is empty on top.</precondition>
    <action>robot.pick_and_place("yellow block", "top left corner")</action>
    <postcondition>The yellow block is placed on top of the top left corner.</postcondition>
    <postcondition>The top left corner is not empty on top.</postcondition>
  </apack>
  <apack>
    <precondition>The cyan block is empty on top.</precondition>
    <precondition>The green block is empty on top.</precondition>
    <action>robot.pick_and_place("cyan block", "green block")</action>
    <postcondition>The cyan block is placed on top of the green block.</postcondition>
    <postcondition>The green block is not empty on top.</postcondition>
  </apack>
  <apack>
    <precondition>The yellow block is empty on top.</precondition>
    <precondition>The brown block is empty on top.</precondition>
    <action>robot.pick_and_place("yellow block", "brown block")</action>
    <postcondition>The yellow block is placed on top of the brown block.</postcondition>
    <postcondition>The brown block is not empty on top.</postcondition>
  </apack>
  <apack>
    <precondition>The cyan block is empty on top.</precondition>
    <precondition>The yellow block is empty on top.</precondition>
    <action>robot.pick_and_place("cyan block", "yellow block")</action>
    <postcondition>The cyan block is placed on top of the yellow block.</postcondition>
    <postcondition>The yellow block is not empty on top.</postcondition>
  </apack>
  <apack>
    <precondition>The green block is empty on top.</precondition>
    <precondition>The cyan block is empty on top.</precondition>
    <action>robot.pick_and_place("green block", "cyan block")</action>
    <postcondition>The green block is placed on top of the cyan block.</postcondition>
    <postcondition>The cyan block is not empty on top.</postcondition>
  </apack>
</sequence>
</plan>

```

(e) The result of the second replanning.

```

validation result: apack 1
* Syntax: Correct
* Semantics: Correct
* Feasibility: Correct

apack 2
* Syntax: Correct
* Semantics: Correct
* Feasibility: Correct

apack 3
* Syntax: Correct
* Semantics: Correct
* Feasibility: Correct

apack 4
* Syntax: Correct
* Semantics: Correct
* Feasibility: Correct

apack 5
* Syntax: Correct
* Semantics: Correct
* Feasibility: Correct

Final Report
- Syntax Check: Success
- Semantics Check: Success
- Feasibility Check: Success
- Task Completion: Success
The final stack is [brown block, yellow block, cyan block, green block].
The target stack is [brown block, yellow block, cyan block, green block].
The order of the blocks in the final stack matches the target stack.

```

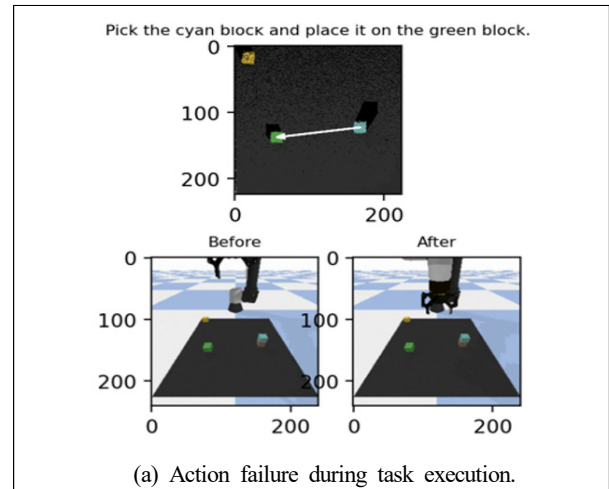
(f) The result of the third plan validation.

이 피드백을 토대로 다시 2차 재계획을 수립한 결과, 표 20 (e)와 같은 재계획 결과를 얻었다. 그리고 이어서 진행된 3차 계획 검증 과정에서 표 20 (f)와 같이 이 수정된 계획은 실행 가능할뿐만 아니라(feasibility check: success), 사용자 지시도 달성 가능한(task completion: success) 올바른 계획으로 최종 판정함으로써 작업 계획 실행 단계로 들어갔다. 많지는 않지만 이렇게 초기에 잘못된 작업 계획을 생성한 경우에도, ChainBot 시스템의 계획 검증과 재계획 기능을 통해 최종적으로 올바른 작업 계획을 수립해 실행할 수 있음을 이 사례를 통해 다시 한번 확인할 수 있었다.

또 이 작업 계획의 실행 과정을 따라가보면, 작업 계획의 두 번째 행동 robot.pick_and_place("cyan block", "green block")을 실행하는 중에 표 21 (a)와 같이 인위적인 실행 오류가 발생하여 cyan block이 제대로 옮겨지지 못하였고, 표 21 (b)와 같이 ChainBot 시스템의 GPT-5 기반 오류 탐지기가 이 행동의 후조건(post-condition)을 체크하여 해당 오류를 발견함으로써, 재계획 과정에 들어갔다. 그 결과 표 22 (a)와 같은 복구 계획(recovery plan)을 수립하여 실행함으로써, 최종적으로 표 22 (b)와 같이 마지막 행동까지 무사히 실행함으로써 해당 작업을 성공적으로 완수하였다. 이 사례를 통해서도 ChainBot 시스템의 GPT-5 기반 오류 탐지와 복구 방식이 작업 계획 실행 중 오류 상황에서도 작업 완수에 효과적임을 다시 한번 확인할 수 있었다.

표 21. 작업 계획 실행 중의 오류 탐지: 사례 A.

Table 21. Error detection during task plan execution: case A.



(a) Action failure during task execution.

```

=== GPT raw output (step:2, label:post) ===
Checked conditions:
- The cyan block is placed on top of the green block.
- The green block is not empty on top.

Model output:
fall

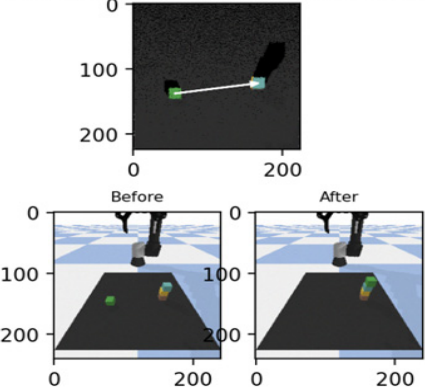
description:
the cyan block is placed on top of the brown block.
the cyan block is empty on top.
the brown block is placed on top of the table.
the brown block is not empty on top.
the green block is placed on top of the table.
the green block is empty on top.
the yellow block is placed on top of the table.
the yellow block is empty on top.
=====

```

(b) Error detection by GPT-5.

표 22. 작업 계획 실행 중의 오류 복구: 사례 A.

Table 22. Error recovery during task plan execution: case A.

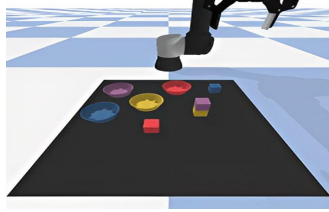
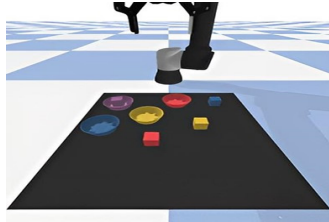
<pre> <plan> <sequence> <apack> <precondition>The cyan block is empty on top.</precondition> <precondition>The green block is empty on top.</precondition> <action>robot.pick_and_place("cyan block", "green block")</action> <postcondition>The cyan block is placed on top of the green block.</postcondition> <postcondition>The green block is not empty on top.</postcondition> </apack> <apack> <precondition>The yellow block is empty on top.</precondition> <precondition>The brown block is empty on top.</precondition> <action>robot.pick_and_place("yellow block", "brown block")</action> <postcondition>The yellow block is placed on top of the brown block.</postcondition> <postcondition>The brown block is not empty on top.</postcondition> </apack> <apack> <precondition>The cyan block is empty on top.</precondition> <precondition>The yellow block is empty on top.</precondition> <action>robot.pick_and_place("cyan block", "yellow block")</action> <postcondition>The cyan block is placed on top of the yellow block.</postcondition> <postcondition>The yellow block is not empty on top.</postcondition> </apack> <apack> <precondition>The green block is empty on top.</precondition> <precondition>The cyan block is empty on top.</precondition> <action>robot.pick_and_place("green block", "cyan block")</action> <postcondition>The green block is placed on top of the cyan block.</postcondition> <postcondition>The cyan block is not empty on top.</postcondition> </apack> </sequence> </plan> </pre>	(a) The recovery plan.
 Pick the green block and place it on the cyan block. 0 100 200 0 200 Before After 0 100 200 0 100 200 0 100 200	(b) The final action execution.

다음 작업 사례는 표 23 (a)와 같은 초기 상태에서 사용자 지시 “Place each block into the matching bowl.”를 달성하기 위한 MM 도메인 작업의 하나이다. 이 사례에서 ChainBot 시스템의 작업 계획 생성기는 purple block, blue block red block, yellow block를 각각 동일한 색상의 purple bowl, blue bowl red bowl, yellow bowl에 차례대로 옮겨놓은 올바른 작업 계획을 생성하였다. 그리고 이어서 진행된 계획 검증 단계에서도 오류 발견없이 검증을 마쳤다.

하지만 계획 실행 단계에서는 표 24 (a)와 같이 첫 번째 행동인 robot.pick_and_place(“purple block“, ”purple bowl“)을 성공적으로 실행한 후, 행동의 후조건을 체크한 결과 표 24 (b)와 같이 실제 작업 상태와는 달리 행동 실행의 오류가 발생한 것으로 오인하게 되었다. 따라서 ChainBot 시스템은 오류 복구를 위한 작업 재계획과 실행을 제한 횟수만큼 반복하였으나 끝내 작업을 완수하는데 실패하였다. 이러한 결과의 주된 원인은 현재 GPT-5를 이용한 작업 실행 중 오류 탐지의 불안정성에 기인한 것으로, 동일한 색상의 그릇(purple bowl) 안에 놓인 동일 색상의 블록(purple block) 상태를 정확히 인식하지

표 23. ChainBot 시스템의 정성적 평가: 사례 B.

Table 23. Qualitative analysis of ChainBot system: case B.

 (a) The initial state.	“Place each block into the matching bowl.” (b) User instruction.
 (a) The state after the first action has been done successfully.	=== GPT raw output (step:1, label:post) === Checked conditions: - The purple block is placed on top of the purple bowl. - The purple bowl is not empty on top. Model output: fail description: the blue block is placed on top of the table. the blue block is empty on top. the red block is placed on top of the table. the red block is empty on top. the yellow block is placed on top of the table. the yellow block is empty on top. the purple block is placed on top of the table. the purple block is empty on top. the blue bowl is placed on top of the table. the blue bowl is empty on top. the red bowl is placed on top of the table. the red bowl is empty on top. the yellow bowl is placed on top of the table. the yellow bowl is empty on top. the purple bowl is placed on top of the table. the purple bowl is empty on top. ===== (b) Error detection by GPT-5.

못하는 경우가 종종 발생하기 때문이다. 이것은 현재 ChainBot 시스템의 몇 가지 한계점 중 하나를 드러낸 것으로, 영상 기반 오류 탐지 성능을 좀 더 개선하기 위한 추가 연구가 필요하다고 판단한다.

V. 결론

본 논문에서는 다수의 기저 모델들을 동적으로 연결하여 사용자의 자연어 지시에 따른 로봇 물체 조작 작업들을 자율적으로 계획하고 실행해주는 에이전트 시스템 ChainBot을 제안하였다. ChainBot 시스템은 효과적인 작업 계획 생성을 위해 거대 언어 모델의 하나인 Gemini-2.0을 이용하며, 계획 생성 프롬프트에 작업의 기본 정보들 외에 작업 계획 사례들과 각 도메인 별 경험적 지식과 같은 풍부한 문맥 정보들을 추가

로 포함시킴으로써 올바른 작업 계획 생성을 유도한다. 또 ChainBot 시스템은 실행 전 작업 계획의 품질을 높이기 위해, 거대 언어 모델 Gemini-2.0을 이용한 작업 실행 전 계획 검증 단계를 수행한다. 그리고 이 단계에서는 작업 계획에 대한 구문적 검사, 의미적 검사, 실행 가능성 검사, 그리고 작업 달성 가능성 검사 등 심도있는 계획 검증을 수행하며, 오류가 발견되면 검증 결과를 피드백으로 삼아 작업을 다시 재계획한다. 또 ChainBot 시스템은 작업 실행 중에도 빠른 오류 발견과 복구를 위해, 거대 멀티모달 모델인 GPT-5를 이용해 각 작업 행동의 실행 전후에 실시간 카메라 영상을 토대로 해당 행동의 전조건뿐만 아니라 후조건들도 검사함으로써 오류를 탐지한다. 그리고 오류가 발견되면 그 오류 상태부터 시작해서 다시 사용자 지시를 달성할 수 있는 작업 계획을 재수립해서 실행한다. 또 ChainBot은 하나의 에이전트 시스템으로서, 사용자의 특별한 개입없이도 다수의 거대 기저 모델들을 유연하게 연동하여 로봇 조작 작업들을 능동적으로 수행할 수 있도록 대표적인 에이전트 개발 프레임워크인 LangChain과 Lang Graph를 이용하여 효율적으로 구현하였다. 본 논문에서는 다양한 정량적, 정성적 실험들을 통해, ChainBot 에이전트 시스템의 우수성을 입증하였다.

한편, 앞서 정성적 평가에서 언급한 것과 같이, 현재의 ChainBot 시스템은 작업 계획 실행 중 오류 탐지를 위한 GPT-5와 같은 시각-언어 모델(VLM) 혹은 거대 멀티 모달 기저 모델(LMM)의 인식 정확도의 한계성으로 인해 작업에 실패하는 경우가 종종 있다. 이러한 한계성을 극복하고 ChainBot 시스템의 성능을 보다 향상시키기 위해서는 현재의 멀티 모달 기저 모델을 ChainBot 작업 환경에 맞추어 파라미터-효율적인 미세 조정 추가 학습(parameter-efficient finetuning)을 수행하거나 별도의 ChainBot 작업 환경 영상들로 구축된 멀티 모달 RAG (multimodal retrieval-augmented generation) 체계를 구성하는 방안 등 다양한 해결 방안을 찾기 위한 향후 연구들이 필요한 것으로 판단한다.

REFERENCES

- [1] Z. Yang, L. Li, K. Lin, J. Wang, C. Lin, Z. Liu, and L. Wang, "The dawn of LLMs: preliminary explorations with GPT-4V," arXiv preprint arXiv:2309.17421, 2023.
doi: <https://doi.org/10.48550/arXiv.2309.17421>
- [2] Gemini Team Google, "Gemini: a family of highly capable multimodal models," arXiv preprint arXiv:2312.11805, 2023.
doi: <https://doi.org/10.48550/arXiv.2312.11805>
- [3] A. Singh, "Meta Llam 4: the future of multimodal AI," SSRN, 2025.
doi: <http://dx.doi.org/10.2139/ssrn.5208228>
- [4] A. Radford, J. Kim, C. Hallacy, A. Ramesh, G. Goh, and S. Agarwal, et al., "Learning transferable visual models from natural language supervision," *Proc. of the International Conference on Machine Learning (ICML)*, vol. 139, pp. 8748-8763, 2021.
- [5] X. Gu, T. Lin, W. Kuo, and Y. Cui, "Open-vocabulary object detection via vision and language knowledge distillation," *Proc. of the International Conference on Learning Representations (ICLR)*, 2022.
- [6] J. Li, D. Li, S. Savarese, and S. Hoi, "BLIP-2: boosting language-image pre-training with frozen image encoders and large language models," *Proc. of the International Conference on Machine Learning (ICML)*, vol. 202, pp. 19730-19742, 2023.
- [7] J. Choi, H. Baek, C. Park, and I. Kim, "Utilizing AI foundation models for language-driven zero-shot object navigation tasks," *J. of Korea Robotics Society*, vol. 19, no. 3, pp. 293-310, 2024.
- [8] H. Bae, K. Park, and I. Kim, "Large language model-based semantic text decoupling and cross attention-based bidirectional multimodal feature fusion for generalize three-dimensional referring expression segmentation," *Journal of Institute of Control, Robotics and Systems (in Korean)*, vol. 31, no. 5, pp. 463-480, 2025.
- [9] D. Driess, F. Xia, M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, et al., "PaLM-E: an embodied multimodal language model," arXiv preprint arXiv:2303.03378, 2023.
doi: <https://doi.org/10.48550/arXiv.2303.03378>
- [10] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, et al., "OpenVLA: an open-source vision-language-action model," *Proc. of the Conference on Robot Learning (CoRL)*, 2024.
- [11] Gemini Robotics Team, "Gemini Robotics: bringing AI into the physical world," arXiv preprint arXiv:2503.20020, 2025.
doi: <https://doi.org/10.48550/arXiv.2503.20020>
- [12] C. Cui, P. Ding, W. Song, S. Bai, X. Tong, Z. Ge, et al., "OpenHelix: a short survey, empirical analysis, and open-source dual-system VLA model for robotic manipulation," arXiv preprint arXiv:2505.03912, 2025.
doi: <https://doi.org/10.48550/arXiv.2505.03912>
- [13] T. Gupta and A. Kembhavi, "Visual programming: compositional visual reasoning without training," *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14953-14962, 2023.
doi: <https://doi.org/10.1109/CVPR52729.2023.01436>
- [14] C. Wu, S. Yin, W. Qi, X. Wang, Z. Tang, and N. Duan, "Visual ChatGPT: talking, drawing and editing with visual foundation models," arXiv preprint arXiv:2303.04671, 2023.
doi: <https://doi.org/10.48550/arXiv.2303.04671>
- [15] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "HuggingGPT: solving AI tasks with ChatGPT and its friends in Hugging Face," *Proc. of the International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 38154 - 38180, 2023.
- [16] Z. Liu, Y. He, W. Wang, W. Wang, Y. Wang, S. Chen, et al., "InternGPT: solving vision-centric tasks by interacting with ChatGPT beyond language," arXiv preprint arXiv:2305.05662, 2023.
doi: <https://doi.org/10.48550/arXiv.2305.05662>
- [17] A. Zeng, M. Attarian, B. Ichter, K. Choromanski, A. Wong, S. Welker, et al., "Socratic models: composing zero-shot multimodal reasoning with language," *Proc. of the International Conference on Learning Representations*

- (ICLR), 2023.
- [18] Z. Wang, A. Li, Z. Li, and X. Liu, "GenArtist: multimodal LLM as an agent for unified image generation and editing," *Proc. of the International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 128374 - 128395, 2024.
 - [19] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, and J. Tremblay, "ProgPrompt: generating situated robot task plans using large language models," *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523-11530, 2023.
 - [20] J. Tang, Z. Ye, Y. Yan, Z. Zheng, T. Gao, and Y. Jin, "Zero-shot robotic manipulation with language-guided instruction and formal task planning," arXiv preprint arXiv: 2501.15214, 2025.
doi: <https://doi.org/10.48550/arXiv.2501.15214>
 - [21] Y. Guo, Y. Wang, L. Zha, Z. Jiang, and J. Chen, "DoReMi: grounding language model by detecting and recovering from plan-execution misalignment," *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*, pp. 12124-12131, 2024.
 - [22] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, et al., "Do as I can, not as I say: grounding language in robotic affordances," arXiv preprint arXiv: 2204.01691, 2022.
doi: <https://doi.org/10.48550/arXiv.2204.01691>
 - [23] T. Silver, V. Hariprasad, R. Shuttleeworth, N. Kumar, T. L. Perez, and L. P. Kaelbling, "Generalized planning in PDDL with pretrained large language models," *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 20256 - 20264, 2024.
doi: <https://doi.org/10.1609/aaai.v38i18.30006>
 - [24] R. Wang, Z. Yang, Z. Zhao, X. Tong, Z. Hong, and K. Qian, "LLM-based robot task planning with exceptional handling for general prurpose service robots," arXiv preprint arXiv:2405.15646, 2024.
doi: <https://doi.org/10.48550/arXiv.2405.15646>
 - [25] F. Ahmad, H. Ismail, J. Styruud, M. Stenmark, and V. Krueger, "A unified framework for real-time failure handling in robotics using vision-language models, reactive planner and behavior trees," arXiv preprint arXiv:2503.15202, 2025.
doi: <https://doi.org/10.48550/arXiv.2503.15202>
 - [26] Microsoft, Copilot Studio, <https://www.microsoft.com/en-us/microsoft-copilot/microsoft-copilot-studio>
 - [27] Google, Agent Development Kits(ADK), <https://google.github.io/adk-docs/>
 - [28] LangChain, <https://www.langchain.com>
 - [29] M. Shridhar, L. Manuelli, and D. Fox, "CLIPort: what and where pathways for robotic manipulation," *Proc. of the Conference on Robot Learning (CoRL)*, 2021.
 - [30] X. Liu, T. Zhang, Y. Gu, I. L. Iong, Y. Xu, and X. Song, et al., "VisualAgentBench: towards large multimodal models as visual foundation agents," *Proc. of the International Conference on Learning Representations (ICLR)*, 2025.
 - [31] C. Zhang, Z. Yang, J. Liu, Y. Han, X. Chen, Z. Huang, et al., "AppAgent: multimodal agents as smartphone users," *Proc. of the International Conference on Human Factors in Computing Systems (CHI)*, pp. 1-20, 2025.
doi: <https://doi.org/10.1145/3706598.3713600>
 - [32] D. Han, T. McInroe, A. Jelley, S. V. Albrecht, P. Bell, and A. Storkey, "LLM-personalize: aligning LLM planners with human preferences via reinforced self-training for housekeeping robots," *Proc. of the 31st International Conference on Computational Linguistics (COLING)*, pp. 1465-1474, 2025.
<https://aclanthology.org/2025.coling-main.98/>
 - [33] S. Gupta, K. Yao, L. Niederhauser, and A. Billard, "Action contextualization: adaptive task planning and action tuning using large language models," *IEEE Robotics and Automation Letters*, vol. 9, no. 11, pp. 9407-9414, 2024.
doi: <https://doi.org/10.1109/LRA.2024.3460408>
 - [34] Robotics at Google, "Codes as policies: language model programs for embodied control," *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493-9500, 2023.
doi: <https://doi.org/10.1109/ICRA48891.2023.10160591>
 - [35] S. S. Kannan, V. L. N. Venkatesh, and B. C. Min, "SMART-LLM: smart multi-agent robot task planning using large language models," *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12140-12147, 2024.
doi: <https://doi.org/10.1109/IROS58592.2024.10802322>
 - [36] Z. Mandi, S. Jain and S. Song, "RoCo: dialectic multi-robot collaboration with large language models," *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 286-299, 2024.
doi: <https://doi.org/10.1109/ICRA57147.2024.10610855>
 - [37] Z. Zhao, W. S. Lee, and D. Hsu, "Differentiable parsing and visual grounding of natural language instructions for object placement," *Proc. of the Conference on Robot Learning (CoRL)*, 2022.
 - [38] D. Venuto, S. N. Islam, M. Klissarov, D. Precup, S. Yang, and A. Anand, "Code as reward: empowering reinforcement learning with VLMs," *Proc. of the International Conference on Machine Learning (ICML)*, vol. 235, pp. 49368-49387, 2024.
 - [39] S. Sagar, J. Duan, S. Vasudevan, Y. Zhou, H. B. Amor, and D. Fox, et al., "From mystery to mastery: failure diagnosis for improving manipulation policies," *Proc. of Robotics: Science and Systems (RSS)*, 2025.
 - [40] J. Duan, W. Pumacay, N. Kumar, Y. R. Wang, S. Tian, and W. Yuan, et al., "AHA: a vision-language-model for detecting and reasoning over failures in robotic manipulation," *Proc. of the International Conference on Learning Representations (ICLR)*, 2025.
 - [41] Streamlit documentation, <https://docs.streamlit.io/>
 - [42] X. Wang, L. Xie, C. Dong, and Y. Shan, "Real-ESRGAN: training real-world blind super-resolution with pure synthetic data," arXiv preprint arXiv:2107.10833, 2021.
doi: <https://doi.org/10.48550/arXiv.2107.10833>

**박 경 민**

2024년 경기대 컴퓨터공학부 졸업. 2024년~현재 경기대 일반대학원 컴퓨터학과 석사과정. 관심분야는 인공지능, 로봇지능, 컴퓨터비전, 상식추론.

**김 인 철**

1985년 서울대 수학과 졸업. 1987년 동대학원 이학석사. 1995년 동 대학원 이학박사. 1996년~현재 경기대학교 컴퓨터공학부 교수. 관심 분야는 인공지능, 로봇지능, 학습 및 지식추론.