TASK 2.1 – Prompt Structure Experimentation (Cohere)

Prompt Format 1: Standard Text-Based

Example: "Manual excerpt:\n ...\n DEFUSER sees or asks:\n ...\n"

Result: Moderate accuracy. Reasonable for casual interaction, but lack of structure sometimes caused the model to get confused about what task it was supposed to complete.

But sometimes the model produced: <|im_start|>assistant Here's how the module works: 1 The user is presented with a sequence of flashing colors. 2 The user presses the buttons to repeat the sequence. 3 The sequence is presented again, and the user asks if they want

This suggests the model fell back into generic instruction behavior, likely due to ambiguity in the input format.

Prompt Format 2: Structured Markdown

Example: "### Manual \n... \n### DEFUSER sees or asks \n..."

Result: Slightly higher accuracy. Markdown-style sectioning helped the model separate sections from each other, leading to a clearer understanding of what to focus on. The model produced shorter, more direct responses.

Prompt Format 3: JSON-Formatted

Example (simplified structure):
{ "role": "Expert", "input": ..., "output": ... }

Result: This basic structure didn't affect results in any significant way. Implementing a more detailed JSON-style prompt like:

{ "role": "Expert", "manual": "...", "bomb_state": { "wires": [...], "serial_number": "PAABB4" }, "task": "Which wire should be cut?" }

was not practical with the current testing setup.

TASK 2.2 – Parameter Tuning (Cohere)

timestamp	model	max_tokens	temperature	top_p	top_k	bombs_disarmed	bombs_exploded	correct_actions	incorrect_actions
2025-05-01 00:03:48	Cohere	50	0.1	0.9	50	0	4	6	0
2025-05-01 00:04:54	Cohere	50	0.7	0.9	50	0	5	4	1
2025-05-01 00:06:04	Cohere	50	0.92	0.9	50	0	3	3	4
2025-05-01 00:09:12	Cohere	50	0.4	0.8	50	0	4	6	0
2025-05-01 00:10:21	Cohere	50	0.4	0.9	50	0	1	8	1
2025-05-01 00:11:32	Cohere	50	0.4	1.0	50	0	0	5	5
2025-05-01 00:59:38	Cohere	50	0.4	0.9	20	0	0	3	7
2025-05-01 01:00:47	Cohere	50	0.4	0.9	50	0	2	6	2
2025-05-01 01:01:54	Cohere	50	0.4	0.9	100	0	4	6	0

Incorrect actions are actions that didn't follow the proper format or that the model didn't understand—such as using a tool incorrectly. Actual mistakes in reasoning are labeled as bombs_exploded. Correct actions are actions like cut, press, hold, etc., which did not result in the bomb exploding.

Temperature had the most significant effect on the results. Low temperature leads to the highest number of correct actions, which makes sense, as we want our model to focus on a single correct answer. On the other hand, low temperature have a negative effect by making the model more prone to getting stuck in repetitive loops(wich happend very rarely). However, this is relatively well handled by CrewAI, which detects when such a situation occurs and prompts the model to avoid repeating itself.

top_p and max_tokens didn't affect preformance in any sygnificant way, on when pushed to the extrimes there was signifficant performance degradation

TASK 2.3 & Task 3.3 – Methodology, Analysis,CrewAI Agent Behavior (Qwen3:8b CREWAI)

A suggested template in the two-agent.py setup directly fed manual_text and bomb_state to the model, which made any conversation between models redundant. Therefore, for this analysis, we focus on AI agents from CrewAI, which actually needed to communicate with each other to complete the task.

The model achieved a success rate of approximately 63%, with nearly all mistakes occurring on the bomb expert's side—even under optimal settings: temperature=0.3, top_p=0.8, and top_k=50. Several models were tested:

SmolLM-135M-Instruct — impossible. Model completely didn't understand the task at hand (very often starting to write Python code to solve a problem) or was starting a long answer that wasn't based on any of the information provided in the prompt.

Cohere/deepseek-r1:7 — didn't understand the format of how to execute given tools, problems with running tools, letalon tool arguments.

lamma3.2 — performed much better in CrewAI env but had issues with selecting correct instruction (or sometimes format) for DefuserTool.

qwen3:8b — performed the best from all the rest and that model is used/discussed from this point on (with temperature=0.3, top_p=0.8, and top_k=50).

(qwen3) While it's a rather poor score, it was a relatively small LLM model, and the model, even though it was providing wrong recommendations to the defuser expert, stayed in character and always understood game rules and what was asked of it.

In terms of conversation efficiency, the model performed very well and behaved as expected. However, this was only achieved after significant prompt tuning and by enforcing a strict conversation protocol. The only major issue was that actions which could be completed in a single cycle (e.g., "press and hold for 1 second") were often split into two separate conversation cycles.

The simple task of retrieving current bomb state was problematic for almost all models. And in order to speed the process and reduce the number of mistakes, the DefuserTool.run method was modified so it always returns bomb state—even when the wrong command is given (very often LLM used get state, current state, even though after incorrect use, help message stated what is the correct format). Which solved that issue completely but shows that used models have trouble with fully understanding env.

To improve performance, the flow of conversation might be changed to force LLM to run ExpertTool only once in a conversation cycle and use the result of that task for ask_relevant_question_task and action_proposition_task. But it wouldn't affect Crew success rate. Other issues that occurred were: when the model had a problem with argument semantics and had to run it multiple times (cut_wire_3 -> help -> state), it caused it to 'forget' the initial task, and for example returned just current bomb state without performing any action. It most likely is because of relatively small context window.

Output logs showing agent interaction of CrewAI agents are in the file: /crew_ai_log outputs for Parameter Tuning are generated by two_agents.py and are saved in /results/runlog.csv

running this version was used to run on windows for server url are changed to localhost (from 0.0.0.0) and python version used was Python 3.10.6