Bewertungskriterien für den sechsten Anwendungsfall

Bewertung

Tests:

	OpenCodeInterpreter DS-33B	**Codestral-22B-v0.1**	**DeepSeek-V2-Lite**	**Llama3.1-70B**	**Starcoder**	**Summe:**
Durchschnittlich behobene Bugs in %	**40,0%**	**66,67%**	**60,0%**	**80,0%**	**20,0%**	
Durchschnittlich richtig implementierte Funktionalitäten in %	**93,34%**	**100,0%**	**100,0%**	**100,00%**	**13,34%**	
SUMME (10 möglich)	**5**	**7**	**5**	**4**	**7**	**28**

Anmerkungen

OpenCodeInterpreterDS-33B:

Actor-Relationship-Game:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 nicht gefixt; Bug3 gefixt (1)

Perfektive Wartung: Erste Funktionalität erfolgreich implementiert. Es musste aber erst die Datei und die konkrete Methode genannt werden, da das LLM beispielsweise nicht die richtige Datei für die Anpassung vorgesehen hatte. Zweite Funktionalität auch nach der Angabe der richtigen Datei und der richtigen Methode nicht richtig hinzugefügt. Dritte Codeanpassung erfolgreich umgesetzt. (2 Erfolgreich)

Hone:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 nicht gefixt; Bug3 nicht gefixt (0)

Perfektive Wartung: Richtige Datei erkannt und erste Funktionalität richtig implementiert. Bei der zweiten Funktionalität wurde nicht die richtige Datei erkannt aber die richtige Funktion implementiert. Für die dritte Funktionalität gilt das Gleiche wie bei der ersten. (3)

Hybrid-Images:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 nicht gefixt; Bug3 gefixt (1)

Perfektive Wartung: Es wird bei der ersten Funktionalität die richtige Datei erkannt. Die implementierte Funktion ist dann auch nach einigen Anpassungen lauffähig. Bei der zweiten Funktionalität wird die Datei richtig erkannt und die Funktionalität unter der richtigen Verwendung der „low_pass“ Funktion implementiert. Auch bei der dritten Funktionalität wird direkt die richtige Datei erkannt. Die generierte Implementierung ist nach kleinen Anpassungen auch lauffähig und entspricht dabei nicht der Implementierung aus dem eigentlichen Projekt. (3)

Login-Registration:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 nicht gefixt; Bug3 gefixt (1)

Perfektive Wartung: Es wurde die erste Funktion erfolgreich implementiert. Selbiges gilt für alle weiteren Funktionalitäten. (3)

Readtime:

Korrektive Wartung: Bug 1 gefixt; Bug2 gefixt; Bug3 gefixt (3)

Perfektive Wartung: Es wurde für die erste Funktionalität die zu verändernde Datei richtig erkannt und die Funktionalität richtig implementiert. Auch wenn bei der zweiten Funktionalität zunächst die eigentlich falsche Datei für die Anpassung genannt wurde, so wurden von dem LLM jeweils eine Methode für die „utils.py“ Datei und eine weitere für die „api.py“ Datei implementiert. Da der Aufruf dann in der „api.py“ Datei stattfindet, wurde somit auch die zweite Funktionalität richtig implementiert. Bezüglich der dritten Funktionalität wurde die falsche Datei für die Veränderung vorgesehen. Jedoch ist die Implementierung der Funktionalität richtig. (3)

Codestral-22B-v0.1:

Actor-Relationship-Game:

Korrektive Wartung: Bug 1 gefixt; Bug2 nicht gefixt; Bug3 gefixt (2)

Perfektive Wartung: Nach Angabe der richtigen Datei wurde die erste Funktionalität richtig implementiert. Die zweite Funktionalität wurde nach der Angabe der richtigen Datei fast richtig implementiert. Es wurde lediglich vergessen bei dem „size()“ Aufruf das Ergebnis mit „1“ zu subtrahieren. Dritte Codeanpassung wurde richtig implementiert. (3)

Hone:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 gefixt; Bug3 nicht gefixt (1)

Perfektive Wartung: Bei der ersten Funktion wird die richtige Datei erkannt und die Funktionalität richtig implementiert. Selbiges gilt auch für die zwei weiteren Funktionalitäten. (3)

Hybrid-Images:

Korrektive Wartung: Bug 1 gefixt; Bug2 nicht gefixt; Bug3 gefixt (2)

Perfektive Wartung: Es wird bei der ersten Funktionalität die richtige Datei erkannt und die Funktionalität richtig implementiert. Selbiges gilt auch bei den zwei weiteren Funktionalitäten. (3)

Login-Registration:

Korrektive Wartung: Bug 1 gefixt; Bug2 nicht gefixt; Bug3 gefixt (2)

Perfektive Wartung: Es wurde die erste Funktion erfolgreich implementiert. Selbiges gilt für alle weiteren Funktionalitäten. (3)

Readtime:

Korrektive Wartung: Bug 1 gefixt; Bug2 gefixt; Bug3 gefixt (3)

Perfektive Wartung: Es wird bei der ersten Funktionalität die falsche Datei erkannt aber der richtige Code generiert. Bei der zweiten Funktionalität wird die richtige Datei erkannt. Es muss aber der Code angepasst werden um die implementierten Methoden ohne Fehler ausführen zu können. Bei der dritten Funktionalität wird erneut die falsche Datei für die Änderung vorgesehen aber die richtige Implementierung generiert. (3)

DeepSeek-V2-Lite:

Actor-Relationship-Game:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 gefixt; Bug3 nicht gefixt (1)

Perfektive Wartung: Der Code für die erste Funktionalität ist zwar richtig, bezieht sich jedoch auf die falsche Datei. Die zweite Funktionalität wird nach der Angabe der richtigen Datei und Methode richtig implementiert. Auch die dritte Codeanpassung ist richtig. (3)

Hone:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 gefixt; Bug3 gefixt (2)

Perfektive Wartung: Es wird die richtige Datei erkannt und die richtige Funktionalität implementiert. Gleiches gilt auch für die weiteren zwei Funktionalitäten. Es ist die umfangreiche Dokumentation der implementierten Methoden hervorzuheben. (3)

Hybrid-Images:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 nicht gefixt; Bug3 gefixt (1)

Perfektive Wartung: Es wird direkt die richtige Datei für die Veränderung erkannt, aber erst in dem zweiten Versuch die richtige Funktionalität implementiert. Bei der zweiten Funktionalität wird die richtige Datei für die Veränderung erkannt und die Funktionalität richtig implementiert. Letzteres gilt auch für die dritte Funktionalität. (3)

Login-Registration:

Korrektive Wartung: Bug 1 gefixt; Bug2 nicht gefixt; Bug3 gefixt (2)

Perfektive Wartung: Es wurde die erste Funktion erfolgreich implementiert. Selbiges gilt für alle weiteren Funktionalitäten. (3)

Readtime:

Korrektive Wartung: Bug 1 gefixt; Bug2 gefixt; Bug3 gefixt (3)

Perfektive Wartung: Es wird die falsche Datei für die neue Funktionalität vorgesehen. Die Implementierung selbst ist jedoch richtig. Bei der zweiten Funktionalität werden sowohl die „util.py“ als auch die „api.py“ Dateien angepasst. Beide Anpassungen sind dabei richtig. Obwohl ursprünglich

nur die Anpassung der „api.py“ vorgesehen war, wird somit auch die zweite Funktionalität richtig implementiert. Gleiches gilt für die dritte Funktionalität. (3)

Llama3.1-70B:

Actor-Relationship-Game:

Korrektive Wartung: Bug 1 gefixt; Bug2 gefixt; Bug3 gefixt (3)

Perfektive Wartung: Es wird direkt die richtige Datei für die Veränderung erkannt und die erste Funktionalität richtig implementiert. Bei der zweiten Funktionalität führt erst die Angabe der richtigen Datei und Methode zu der korrekten Implementierung der zweiten Funktionalität. Dritte Codeanpassung wird richtig implementiert. (3)

Hone:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 gefixt; Bug3 nicht gefixt (1)

Perfektive Wartung: Es wird die richtige Datei für die neue Funktionalität erkannt und die Funktionalität selbst auch richtig implementiert. Selbiges gilt auch für die zwei weiteren Funktionalitäten. (3)

Hybrid Images:

Korrektive Wartung: Bug 1 gefixt; Bug2 nicht gefixt; Bug3 gefixt (2)

Perfektive Wartung: Es wird für die erste Funktionalität die richtige Datei erkannt. Der generierte Code für die Funktionalität muss jedoch leicht angepasst werden, um den Code lauffähig zu bekommen. Für die zweite Funktionalität gilt selbiges. Bei der dritten Funktionalität wird ebenso die richtige Datei erkannt. Entgegen den vorherigen Funktionalitäten wird in diesem Fall jedoch direkt der richtige Code implementiert, welcher ohne Anpassungen lauffähig ist. (3)

Login-Registration:

Korrektive Wartung: Bug 1 gefixt; Bug2 gefixt; Bug3 gefixt (3)

Perfektive Wartung: Es wurde die erste Funktion erfolgreich implementiert. Selbiges gilt für alle weiteren Funktionalitäten. (3)

Readtime:

Korrektive Wartung: Bug 1 gefixt; Bug2 gefixt; Bug3 gefixt (3)

Perfektive Wartung: Es wird bei der ersten Funktionalität die falsche Datei für die Anpassungen ausgewählt aber die generierte Implementierung ist richtig. Gleiches gilt für die zweite Funktionalität. Bei der dritten Funktionalität wird sowohl die richtige Datei für die Anpassungen erkannt, als auch die richtige Implementierung generiert. (3)

Starcoder:

Actor-Relationship-Game:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 nicht gefixt; Bug3 gefixt (1)

Perfektive Wartung: Erste Funktionalität wird nicht richtig implementiert. LLM fängt während der Konversation an Buchstaben in Mandarin auszugeben. Auch die zweite Funktionalität wird nicht richtig implementiert. Auch die dritte Codeanpassung findet nicht wie gewünscht statt. (0)

Hone:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 nicht gefixt; Bug3 nicht gefixt (0)

Perfektive Wartung: Es wird für die erste Funktionalität nicht die richtige Datei erkannt und ebenso wenig die richtige Funktionalität implementiert. Bei der zweiten Datei wird die richtige Implementierung gewählt aber initial nicht die richtige Datei erkannt. Bei der dritten Funktionalität ist Starcoder erneut am Halluzinieren. (1)

Hybrid-Images:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 nicht gefixt; Bug3 gefixt (1)

Perfektive Wartung: Es wird bei der ersten Funktionalität zwar die richtige Datei für die Anpassung erkannt. Allerdings findet die Anpassung selbst nicht statt. Bei den weiteren zwei Funktionalitäten wird die zu verändernde Datei nicht erkannt. Bei der dritten Funktionalität entsteht dann zwar nach einigen Nachfragen Code. Jedoch ist dieser selbst nach einigen Anpassungen nicht lauffähig. (0)

Login-Registration:

Korrektive Wartung: Bug 1 nicht gefixt; Bug2 nicht gefixt; Bug3 nicht gefixt (0)

Perfektive Wartung: Es wurde keine der vorgesehenen Funktionalitäten richtig umgesetzt. (0)

Readtime:

Korrektive Wartung: Bug 1 gefixt; Bug2 nicht gefixt; Bug3 nicht gefixt (1)

Perfektive Wartung: Es wird die falsche Datei erkannt und eine fast richtige Implementierung für die erste Funktionalität geliefert. Dabei muss jedoch die Implementierung angepasst und in eine Funktion geschrieben werden, da Starcoder diese Implementierung innerhalb einer Main-Guard generiert hat. Bei der zweiten Funktionalität hat Starcoder erneut mit dem Halluzinieren begonnen. Gleiches gilt auch für die dritte Funktionalität. (1)