Bewertungskriterien für den vierten Anwendungsfall

Bewertung

Tests:

	CodeQwen1.5-7B	**Qwen2-72B**	**Nxcode-CQ7B-orpo**	**Codestral-22B-v0.1**	**Llama3.1-70B**	**Summe:**
Durchschnittliche Testabdeckung in %	**23,4%**	**80,0%**	**22,2%**	**23,4%**	**67,8%**	
Durchschnittlich richtig generierte Testfälle in %	**16,0%**	**92,0%**	**0,0%**	**62,0%**	**96,0%**	
Syntaktische Richtigkeit	**3**	**4**	**3**	**2**	**4**	**16**
Semantische Richtigkeit	**2**	**3**	**2**	**2**	**3**	**12**
SUMME (10 möglich)	**5**	**7**	**5**	**4**	**7**	**28**

Anmerkungen

CodeQwen1.5-7B:

Actor-Relationship-Game:

Syntax: Es fehlen erneut ein paar Umbrüche. Ebenso muss den einzeiligen Kommentaren eine blanke Zeile vorausgehen. Ansonsten den Standards entsprechend.

Semantische Richtigkeit: Erscheint richtig. Zwei der vier Tests für die TMDB-API laufen durch. Zwei tun dies nicht. Letzteres ist aber aufgrund der gewählten Daten und Methodennamen auch intendiert. Somit ist die „TDMB-API“ Klasse vollständig abgedeckt. Gleiches gilt auch für den „ActorTest“, in dem sogar mehr Tests generiert wurden als in der ursprünglichen Referenzimplementierung. In diesem Fall mussten aber die Datentypen für die „ActorID“ von einem Integer in einen String verändert werden. Ansonsten wurde lediglich die Implementierung für die „GraphCreation“ Klasse generiert. Diese ist aber für den betrachteten Anwendungsfall unerheblich, da diese keine Testimplementierungen enthält.

Abdeckung durch Unittests in %: 474LOC insgesamt. 80LOC abgedeckt. 80/4,74 = 16,877 ≈17%

Generierte Testfälle: 0

Richtige Testfälle: 0

Hone:

Syntax: Es fehlen Umbrüche. Ansonsten akzeptabel.

Semantische Richtigkeit: Es wurde zwar Code für die „test_utils.py“ generiert und dieser ist auch voll funktionsfähig. Allerdings handelt es sich bei dieser Datei nicht um die Unit-Tests selbst, sondern um Util-Funktionen für die zu generierenden Unittests.

Abdeckung durch Unittests in %: 249LOC insgesamt. 0LOC abgedeckt. 0/2,49 = 0 ≈ 0%

Generierte Testfälle: 0

Richtige Testfälle: 0

Hybrid-Images:

Syntax: Es wurden nur Methodenköpfe implementiert. Daraufhin folgt lediglich „pass # TODO“.

Semantische Richtigkeit: Es wurden nur Methodenköpfe implementiert. Daraufhin folgt lediglich „pass # TODO“.

Abdeckung durch Unittests in %: 144LOC insgesamt. 0LOC abgedeckt. 0/1,44 = 0 ≈ 0%

Generierte Testfälle: 8

Richtige Testfälle: 8 (Mit Anpassungen an der Benennung von Variablen)

Login-Registration:

Syntax: Es wurde nichts implementiert.

Semantische Richtigkeit: Es wurde nichts implementiert.

Abdeckung durch Unittests in %: 706LOC insgesamt. 0LOC abgedeckt. 0/7,06 = 0 ≈ 0%

Generierte Testfälle: 0

Richtige Testfälle: 0

Readtime:

Syntax: Gemäß der PEP8 Standards.

Semantische Richtigkeit: Es wurden alle nötigen Tests implementiert. Für das Ausführen mussten einige Anpassungen an den Imports durchgeführt werden. Dann sind die Tests ausführbar und schlagen dann jedoch fehl. Der Grund hierfür liegt jedoch lediglich darin, dass der Eingabetext zu einer Lesedauer von zwei Sekunden führt. Somit schlägt jede „assert“ Operation mit der Bedingung „result.seconds == 60“ fehl. Ebenso mussten falsche Funktionsaufrufe entfernt werden, welche bei dem Versuch auf die Parameter von dem „result“ Objekt zugreifen zu können, genutzt wurden. Beispielsweise wurde so aus „result.seconds() == 60“ dann result.seconds == 60. Nach diesen Anpassungen laufen alle Tests fehlerfrei.

Abdeckung durch Unittests in %: 284LOC insgesamt. 284LOC abgedeckt. 284/2,84 = 100,00 ≈ 100%

Generierte Testfälle: 0

Richtige Testfälle: 0

Qwen2-72B:

Actor-Relationship-Game:

Syntax: Es fehlen Umbrüche und Kommentare. Ansonsten entspricht es dem Oracle-Standard.

Semantische Richtigkeit: Es wird lediglich für die implementierten Klassen der jeweilige Konstruktor getestet. Beispiele bilden die Klassen „Movie“ und „Actor“. Die Tests waren auch nur nach der Anpassung der Datentypen von Integer auf String lauffähig. Für die „TMDB-API“ Klasse wurden

dahingegen auch weitere Tests implementiert. Diese erscheinen richtig, sind aber dennoch nicht lauffähig. Auch nach weiteren Anpassungen ist dies nicht der Fall.

Abdeckung durch Unittests in %: 474LOC insgesamt. 37LOC abgedeckt. 37/4,74 = 7,805 ≈8%

Generierte Testfälle: 10

Richtige Testfälle: 10

Hone:

Syntax: Es fehlen Kommentare. Zudem fehlen auch einige Indentations. Ansonsten akzeptabel.

Semantische Richtigkeit: Es wurde Code für das Testen der „csv_utils.py“ und der „hone.py“ Dateien generiert. Es mussten dabei Imports und Dateipfade angepasst werden. Daraufhin sind aber die Tests erfolgreich.

Abdeckung durch Unittests in %: 249LOC insgesamt. LOC abgedeckt. 230/2,49 = 92,369 ≈ 92%

Generierte Testfälle: 10

Richtige Testfälle: 10

Hybrid-Images:

Syntax: Es fehlen einige Indentations. Ansonsten gemäß dem vorgegebenen Standard.

Semantische Richtigkeit: Die Tests werden zwar richtig implementiert. Jedoch schlagen die Tests aufgrund der falsch gewählten Beispiele fehl. Sobald also richtige Numpy-Arrays genutzt werden, funktionieren alle generierten Unittests.

Abdeckung durch Unittests in %: 144LOC insgesamt. 144LOC abgedeckt. 144/1,44 = 100,00 ≈ 100%

Generierte Testfälle: 10

Richtige Testfälle: 10

Login-Registration:

Syntax: Es wurden die Standards eingehalten. Zum Beispiel ist dies daran erkennbar, dass keine Wildcard Imports vorhanden sind.

Semantische Richtigkeit: Es wurden alle nötigen Tests implementiert. Auch die Implementierung der Tests ähnelt sehr stark der vorgegebenen Referenz. Allerdings kommt es bei dem Ausführen der jeweiligen „.spec.js“ Dateien mit „Jest“ zu Fehlern. Da diese nicht klar nachvollziehbar sind, muss Aufwand investiert werden, um die Tests lauffähig zu bekommen.

Abdeckung durch Unittests in %: 706LOC insgesamt. 706LOC abgedeckt. 706/7,06 = 100 ≈ 100%

Generierte Testfälle: 10

Richtige Testfälle: 6

Readtime:

Syntax: Gemäß des PEP8 Standards.

Semantische Richtigkeit: Es wurden alle nötigen Tests implementiert. Für das Ausführen mussten einige Anpassungen an den Imports durchgeführt werden. Dann sind die Tests ausführbar und schlagen dann jedoch fehl. Der Grund hierfür liegt in falschen Bedingungen und falschen Funktionsaufrufen. Letztere mussten entfernt werden. Beispielsweise wurde so aus „self.assertEqual(result.minutes(), 2)“ dann „self.assertEqual(result.minutes, 2)“. Ebenso wurden die Bedingungen angepasst. Nach diesen Anpassungen laufen alle Tests fehlerfrei.

Abdeckung durch Unittests in %: 284LOC insgesamt. 284LOC abgedeckt. 284/2,84 = 100,00 ≈ 100%

Generierte Testfälle: 10

Richtige Testfälle: 10

Nxcode-CQ7B-orpo:

Actor-Relationship-Game:

Syntax: Es fehlen Umbrüche und Kommentare. Das Fehlen der Umbrüche ist vor allem bei einigen Imports sehr auffällig. Ansonsten entspricht es dem Oracle-Standard.

Semantische Richtigkeit: Für die TMDB-API laufen alle generierten Tests nach der Anpassung des „Assertion“ Imports durch. Für die „Actor“ Klasse funktionieren die Tests auch nach einigen Anpassungen nicht, da diese implementierte Methoden voraussetzen die es in der dem LLM zu Verfügung gestellten Referenzimplementierung nicht gibt. Als Beispiel kann die „getFriends()“ Methode genannt werden, welche in der „Actor“ Klasse nicht existiert. Es werden abgesehen davon keine weiteren Tests implementiert. Die restlichen Implementierungen enthalten lediglich den Kopf der jeweiligen Klassen mit dem Kommentar „//TODO: Add more tests for GraphCreation methods“.

Abdeckung durch Unittests in %: 474LOC insgesamt. 37LOC abgedeckt. 42/4,74 = 8,860 ≈9%

Generierte Testfälle: 0

Richtige Testfälle: 0

Hone:

Syntax: Es wurde nichts generiert.

Semantische Richtigkeit: Es wurde nichts generiert.

Abdeckung durch Unittests in %: 249LOC insgesamt. LOC abgedeckt. 0/2,49 = 0,00 ≈ 0%

Generierte Testfälle: 0

Richtige Testfälle: 0

Hybrid-Images:

Syntax: Es wurden nur Methodenköpfe implementiert. Daraufhin folgt lediglich der Befehl „pass“.

Semantische Richtigkeit: Es wurden nur Methodenköpfe implementiert. Daraufhin folgt lediglich der Befehl „pass“.

Abdeckung durch Unittests in %: 144LOC insgesamt. 0LOC abgedeckt. 0/1,44 = 0 ≈ 0%

Generierte Testfälle: 0

Richtige Testfälle: 0

Login-Registration:

Syntax: Es wurden die Standards eingehalten. Zum Beispiel daran erkennbar, dass keine Wildcard Imports vorhanden sind. Es tritt jedoch bei dem Import von „axios“ ein Syntaxfehler auf.

Semantische Richtigkeit: Es wurde nur ein Test für die „HomePage“ Vue Komponente und allen damit verbundenen JS Dateien generiert. Dabei nutzen die Tests „axios“. Der Import musste jedoch angepasst werden, um nicht auf einen Syntaxfehler zu stoßen. Selbst nach dieser Anpassung schlägt der Test fehl. Somit muss mehr Aufwand betrieben werden, um den Test lauffähig zu bekommen.

Abdeckung durch Unittests in %: 706LOC insgesamt. 589LOC abgedeckt. 589/7,06 = 83,42 ≈ 83%

Generierte Testfälle: 0

Richtige Testfälle: 0

Readtime:

Syntax: Gemäß des PEP8 Standards.

Semantische Richtigkeit: Es wurden nur Tests für die „api.py“ generiert. Die generierten Tests beinhalten dabei die gleichen Fehler wie bei allen Modellen. Nach den gleichen Anpassungen sind die Tests lauffähig.

Abdeckung durch Unittests in %: 284LOC insgesamt. 284LOC abgedeckt. 55/2,84 = 19,366 ≈ 19%

Generierte Testfälle: 0

Richtige Testfälle: 0

Codestral-22B-v0.1:

Actor-Relationship-Game:

Syntax: Entspricht dem Oracle-Standard. Imports und Umbrüche richtig.

Semantische Richtigkeit: Für die TMDB-API laufen alle generierten Tests nach der Anpassung des „Assertion“ Imports durch. Es wurden keine weiteren Testfälle generiert.

Abdeckung durch Unittests in %: 474LOC insgesamt. 37LOC abgedeckt. 42/4,74 = 8,860 ≈9%

Generierte Testfälle: 10

Richtige Testfälle: 4

Hone:

Syntax: Es fehlen Kommentare. Ansonsten entsprechend der Standards.

Semantische Richtigkeit: Es wurden nur Tests für die „json_utils.py“ Datei generiert. Diese Tests funktionieren. Jedoch mussten einige Anpassungen vorgenommen werden, da die Tests in einer Klasse eingebettet sind. Dies ist bei der Referenzimplementierung nicht der Fall. Ferner mussten die Imports und Dateipfade angepasst werden.

Abdeckung durch Unittests in %: 249LOC insgesamt. LOC abgedeckt. 19/2,49 = 7,63 ≈ 8%

Generierte Testfälle: 1

Richtige Testfälle: 1

Hybrid Images:

Syntax: Es wurde kein Code generiert.

Semantische Richtigkeit: Es wurde kein Code generiert.

Abdeckung durch Unittests in %: 144LOC insgesamt. 0LOC abgedeckt. 0/1,44 = 0 ≈ 0%

Generierte Testfälle: 7

Richtige Testfälle: 7

Login-Registration:

Syntax: Es wurden die Standards eingehalten.

Semantische Richtigkeit: Es wurden alle nötigen Tests implementiert. Auch die Implementierung der Tests ähnelt sehr stark der vorgegebenen Referenz. Allerdings kommt es bei dem Ausführen der jeweiligen „.spec.js“ Dateien mit „Jest“ zu Fehlern. Da diese nicht klar nachvollziehbar sind, muss Aufwand investiert werden, um diese Tests lauffähig zu bekommen. Somit liegt das gleiche Ergebnis wie bei dem Modell „Qwen2-72B“ vor.

Abdeckung durch Unittests in %: 706LOC insgesamt. 706LOC abgedeckt. 706/7,06 = 100 ≈ 100%

Generierte Testfälle: 10

Richtige Testfälle: 9

Readtime:

Syntax: Es wurde kein Code generiert.

Semantische Richtigkeit: Es wurde kein Code generiert.

Abdeckung durch Unittests in %: 284LOC insgesamt. 284LOC abgedeckt. 0/2,84 = 0,00 ≈ 0%

Generierte Testfälle: 10

Richtige Testfälle: 10

Llama3.1-70B:

Actor-Relationship-Game:

Syntax: Entspricht dem Oracle-Standard. Imports sind richtig. Es wurde sogar das Freilassen der Zeile vor dem Beginn eines einzeiligen Kommentars eingehalten.

Semantische Richtigkeit: Für die TMDB-API laufen zwei der vier generierten Tests durch. Jedoch ist bei den anderen Zwei aufgrund von gewollt invaliden Inputs auch das Scheitern der Tests gewollt. Für die beiden erfolgreichen Tests ist auffällig, dass anstelle eines „assertEquals“ lediglich ein „assertNotNull“ Aufruf genutzt wird, obwohl mit „expectedResponse“ eine passende Variable erstellt, jedoch nicht genutzt wird. Die Tests für die Klasse „Actor“ laufen allesamt durch. Gleiches gilt auch für die Movie Klasse. Bei den Tests für die „ActorGraph“ Klasse wird mit „containsMovie“ erneut eine Methode vorausgesetzt, welche nicht existiert. Entsprechend schlagen zwei der fünf Tests dieser Klasse fehl. Jedoch fehlt auch eine Methode, welche die Verbindung zwischen zwei Schauspielern prüft. Bei der Klasse „ActorGraphUtilTest“ wird eine derart große Menge nicht implementierter Methoden vorausgesetzt, dass die Tests nicht lauffähig sind. Auch bei den Tests für die „GameplayInterface“ Klasse, werden Methoden vorausgesetzt die nicht existieren. Somit ist nur ein Test erfolgreich. Selbiges gilt für die „GraphCreationTest“ Klasse.

Abdeckung durch Unittests in %: 474LOC insgesamt. 185LOC abgedeckt. 185/4,74 = 39,029 ≈39%

Generierte Testfälle: 10

Richtige Testfälle: 10

Hone:

Syntax: Es fehlen Kommentare. Ansonsten entspricht es dem Oracle-Standard.

Semantische Richtigkeit: Es wurden für alle relevanten Dateien Tests generiert. Erneut mussten dann jedoch Anpassungen an den Dateien vorgenommen werden, da alle Tests in Klassen eingebettet wurden. Mit der Anpassung der Imports und Dateipfade sind dann jedoch die Tests lauffähig.

Abdeckung durch Unittests in %: 249LOC insgesamt. LOC abgedeckt. 249/2,49 = 100 ≈ 100%

Generierte Testfälle: 10

Richtige Testfälle: 10

Hybrid-Images:

Syntax: Es fehlen einige Indentations. Ansonsten gemäß den Standards.

Semantische Richtigkeit: Die Tests werden zwar richtig implementiert. Jedoch schlagen die Tests aufgrund der falsch gewählten Beispiele fehl. Sobald also richtige Numpy-Arrays genutzt werden, funktionieren alle generierten Unittests. Somit liegt das gleiche Ergebnis wie bereits bei dem Modell Qwen2-72B vor.

Abdeckung durch Unittests in %: 144LOC insgesamt. 144LOC abgedeckt. 144/1,44 = 100,00 ≈ 100%

Generierte Testfälle: 10

Richtige Testfälle: 10

Login-Registration:

Syntax: Es wurden die Standards eingehalten.

Semantische Richtigkeit: Es wurden alle nötigen Tests implementiert. Bei den Tests für die „RegisterPage“ Vue Komponente laufen sechs der neun Tests ohne Anpassungen. Es musste lediglich das Package „vuelidate“ installiert werden. Jedoch wird hier nur getestet, ob die notwendigen DOM-Elemente, basierend auf der Vue Komponente, vorhanden sind. Gleiches gilt für die Tests der „LoginPage“ Vue Komponente. Hier laufen vier der sieben Tests ohne Anpassungen. Jedoch wird auch hier nur geprüft, ob die richtigen DOM-Elemente basierend auf die „LoginPage“ Vue Komponente vorhanden sind. Anders hingegen sieht es bei den Tests zu der „HomePage“ Vue Komponente aus. In diesem Fall muss eine Anpassung an einer Funktion vorgenommen werden, um den Test ausführen zu können. Letzteres bezieht sich auf die Verwendung des Schlüsselworts „await“ in einer Methode, welche nicht durch das Schlüsselwort „async“ als asynchrone Methode markiert wurde. Nach dem Hinzufügen des „async“ Schlüsselworts, schlagen dann jedoch alle Tests fehl. Es sind somit mehr Anpassungen an den dortigen Tests vorzunehmen.

Abdeckung durch Unittests in %: 706LOC insgesamt. 706LOC abgedeckt. 706/7,06 = 100 ≈ 100%

Generierte Testfälle: 10

Richtige Testfälle: 8

Readtime:

Syntax: Es wurde kein Code generiert.

Semantische Richtigkeit: Es wurde kein Code generiert.

Abdeckung durch Unittests in %: 284LOC insgesamt. 284LOC abgedeckt. 0/2,84 = 0,00 ≈ 0%

Generierte Testfälle: 10

Richtige Testfälle: 10