

Promptstratégiák hatása a nagy nyelvi modellek programkód-tesztelési képességére

The Impact of Prompting Strategies on the Test Case Generation Capabilities of Large Language Models

Dr. HORNYÁK Olivér

Miskolci Egyetem, 3515 Miskolc-Egyetemváros, Informatikai Intézet
oliver.hornyak@uni-miskolc.hu

Abstract

This paper investigates the effect of different prompting strategies on the quality of program tests generated by large language models. In the experiment, test cases were generated for JavaScript programming tasks using three prompts with different levels of detail. The evaluation was based on fault detection rate, executability, size, and efficiency of the generated test suites. The results show that, for the examined tasks, more detailed prompts produced larger test suites but did not improve fault-detection capability, while the executability of the generated test suites was lower than with the basic prompt.

Keywords: large language models, prompt engineering, test case generation, software testing, test design techniques

Kivonat

A cikk különböző promptstratégiák hatását vizsgálja a nagy nyelvi modellek által generált programtesztek minőségére. A kísérletben JavaScript-programozási feladathoz három, eltérő részletezettségű prompt segítségével készültek tesztesetek. Az értékelés a hibafeltérési arány, a tesztkészletek végrehajthatósága, mérete és hatékonysága alapján történt. Az eredmények azt mutatják, hogy a vizsgált feladatokon a részletesebb promptok több tesztesetet eredményeztek, azonban nem növelték a hibafeltérési képességet, és a generált tesztkészletek végrehajthatósága is alacsonyabb volt az egyszerű alap-prompthoz képest.

Kulcsszavak: nagy nyelvi modellek, prompttervezés, teszteset-generálás, szoftvertesztelés, teszttervezési technikák

1. BEVEZETÉS

A nagy nyelvi modellek egyre szélesebb körben jelennek meg a szoftverfejlesztési folyamat különböző szakaszaiban, így a programkód előállítás, dokumentálása, hibakeresése és tesztelése során is. A tesztesetek automatikus generálása különösen ígéretes alkalmazási terület, mivel megfelelően megfogalmazott utasítások alapján a modellek képesek lehetnek a program működését ellenőrző tesztek előállítására. A generált tesztek minősége azonban nemcsak a használt modelltől és a vizsgált programkódtól, hanem jelentős mértékben a modell számára megfogalmazott prompt tartalmától és részletezettségétől is függhet.

A teszteset-generálás során lényeges kérdés, hogy elegendő-e egy általános, rövid utasítás, vagy jobb eredmény érhető el akkor, ha a prompt konkrét tesztelési szempontokat és módszereket is meghatároz. A hagyományos szoftvertesztelésben alkalmazott technikák, például az ekvivalenciaosztály-particionálás, a határérték-elemzés és a negatív tesztelés szisztematikus megközelítést biztosítanak a tesztesetek kialakításához. Ezek explicit megadása a promptban várhatóan befolyásolja a nagy nyelvi modell által előállított tesztkészlet szerkezetét, lefedettségét és hibafeltérő képességét.

Jelen munka célja különböző promptstratégiák hatásának vizsgálata nagy nyelvi modellek programozási feladatokhoz generált teszteseteinek minőségére. A vizsgálat során ugyanazon programozási feladatokhoz eltérő részletezettségű és eltérő mértékben módszertanvezérelt promptok alkalmazásával generált tesztkészletek kerülnek összehasonlításra. Az értékelés középpontjában a hibafeltérési arány, a kódlefedettség, a generált tesztesetek száma és a tesztek hatékonysága áll. A vizsgálat arra keresi a választ, hogy a részletesebb, klasszikus teszttervezési módszereket is előíró promptok valóban jobb minőségű és hatékonyabb tesztkészleteket eredményeznek-e az általános tesztgenerálási utasításokhoz képest.

2. NAGY NYELVI MODELLEK ALKALMAZÁSA TESZTESZET-GENERÁLÁSRA

2.1. LLM-alapú teszteset-generálás

A nagy nyelvi modellek szoftverfejlesztési alkalmazásai közül az automatikus teszteset-generálás az egyik intenzíven kutatott területté vált. Wang és munkatársai 102, nagy nyelvi modellek szoftvertesztelési alkalmazásával foglalkozó publikáció áttekintése alapján a tesztesetek előállítását a terület egyik meghatározó feladatának azonosították [9]. Az LLM-alapú megközelítés egyik előnye a hagyományos automatikus teszt-generálási módszerekkel szemben, hogy a modell a programkód mellett természetes nyelvű specifikációkat, dokumentációt és egyéb kontextuális információkat is fel tud használni a tesztek előállításához.

Schäfer et. al. munkatársai TestPilot rendszere nagy nyelvi modell segítségével generál automatikusan JavaScript egységteszteket. A 25 npm csomag 1684 API-függvényén végzett vizsgálatban a generált tesztek medián utasításlefedettsége 70,2%, áglefedettsége pedig 52,8% volt [7]. A szerzők azt is kimutatták, hogy a promptban rendelkezésre bocsátott információk, például a vizsgált függvény implementációja, szignatúrája és használati példái egyaránt hozzájárulnak a generált tesztek eredményességéhez. A megközelítés további sajátossága, hogy a sikertelen tesztek hibaüzeneteinek visszacsatolásával a modell a korábban generált teszteseteket javítani is képes.

A kódlefedettség ugyanakkor önmagában nem feltétlenül jellemzi megfelelően egy tesztkészlet hibafeltáró képességét. Moradi Dakhel et. al. ezért a mutációs tesztelést kapcsolták össze az LLM-alapú tesztgenerálással. MuTAP eljárásukban a túlélő mutánsokról szóló információ visszakerül a következő promptba, ezzel irányítva a modellt további, hibafeltárára alkalmas tesztesetek létrehozására [3]. Eredményeik arra utalnak, hogy az iteratív visszacsatolás jelentősen növelheti a generált teszteset hibafeltáró képességét.

A nagy nyelvi modellek által generált teszteset azonban nem tekinthető automatikusan helyesnek. Yuan és munkatársai a ChatGPT által generált egységteszteket helyesség, lefedettség, olvashatóság és használhatóság szempontjából vizsgálták [10]. A generált tesztek fordítási hibákat, futásidejű hibákat és különösen hibás ellenőrző feltételeket is azonosítottak. Az általuk javasolt ChatTester ezért iteratív javítási lépéseket alkalmaz a kezdetben generált teszteset minőségének növelésére. Hasonló gyakorlati megközelítést alkalmaz az ipari környezetben vizsgált TestGen-LLM is, ahol a generált teszteset automatikus validációs lépések és fejlesztői ellenőrzés követik [1]. Ezek az eredmények azt mutatják, hogy az LLM-alapú tesztgenerálás ígéretes, de a generált tesztkészlet minőségét a modell mellett a rendelkezésre álló kontextus, a visszacsatolás és a modellnek adott utasítások is jelentősen befolyásolják.

2.2. Promptstratégiák a szoftvertesztelésben

A teszteset-generálás során a prompt nem csupán a feladat természetes nyelvű megfogalmazása, hanem a modell számára rendelkezésre álló tesztelési kontextus és módszertani útmutatás hordozója is. Ennek megfelelően a prompt szerkezete és részletezettsége közvetlenül befolyásolhatja, hogy a modell milyen bemeneti tartományokat, végrehajtási útvonalakat és speciális eseteket vesz figyelembe.

Ryan et. al. SymPrompt módszere a program szerkezetéhez igazodó, többlépcsős promptolást alkalmaz [6]. Ahelyett, hogy a modell egyetlen általános utasítás alapján generálná a teljes tesztkészletet, a tesztgenerálási feladatot több, a program végrehajtási útvonalaihoz kapcsolódó részfeladatra bontják. A vizsgálat eredményei szerint a kód szerkezetét figyelembe vevő prompting a hagyományos alap-promptokhoz képest jelentősen növelheti a lefedettséget, ami alátámasztja, hogy a prompt felépítése önmagában is lényeges tényezője az automatikus tesztgenerálásnak [6].

Ouédraogo et. al. nagymintás empirikus vizsgálata négy nagy nyelvi modell tesztgenerálási képességét és különböző promptstratégiák hatását elemezte [4]. A több mint 216 ezer generált tesztesetre kiterjedő vizsgálat azt mutatta, hogy a promptolási stratégia hatással van a generált teszteset végrehajthatóságára, lefedettségére és általános minőségére. A szerzők eredményei egyúttal arra is rámutatnak, hogy az összetettebb gondolkodási és irányítási lépéseket tartalmazó promptok sem szüntetik meg teljesen a hibás vagy nem végrehajtható teszteset problémáját [4].

A klasszikus teszttervezési módszerek közvetlenül is beépíthetők a promptba. Rodríguez és munkatársai olyan promptot alkalmaztak, amely a forráskód és a követelmények mellett explicit módon előírta az ekvivalenciaosztályok és a határértékek figyelembevételét [5]. Vizsgálatuk megerősítette, hogy a generált teszteset eredményessége nem kizárólag az alkalmazott nyelvi modelltől függ, hanem a prompt kialakításától és a követelmények pontosságától is. Ugyanakkor a szerzők az emberi felülvizsgálat szükségességét is hangsúlyozták [5].

Az eddigi kutatások tehát egyrészt általános prompt-engineering technikák, másrészt egyes teszttervezési módszerek alkalmazhatóságát vizsgálták. Kevés eredmény áll azonban rendelkezésre arról, hogyan változik ugyanazon programozási feladatok esetében a generált tesztkészlet minősége, ha az általános tesztgenerálási utasítást fokozatosan egészítjük ki strukturált tesztelési szempontokkal, majd explicit klasszikus teszttervezési módszerekkel. Jelen kutatás ezt a kérdést vizsgálja azonos nagy nyelvi modell és azonos programozási feladatok alkalmazása mellett, elsősorban a hibafeltérési képesség, a kódlefedettség és a generált tesztek hatékonysága alapján.

3. KÍSÉRLETI MÓDSZERTAN

A vizsgálat célja annak meghatározása, hogy a promptban megadott tesztelési útmutatás részletezettsége és módszertani tartalma milyen hatással van a nagy nyelvi modell által generált tesztesetek minőségére. A kísérlet során ugyanazon programozási feladatokhoz három különböző promptstratégia alkalmazásával készülnek tesztkészletek. A vizsgálatban az alkalmazott nyelvi modell, a programkód, a specifikáció és a futtatási környezet változatlan, így független változóként kizárólag a promptstratégia jelenik meg.

3.1. Programozási feladatok és hibamodellek

A vizsgálati készlet tíz, JavaScript nyelven megvalósított, kis méretű programozási feladatból állt. A feladatok kiválasztásánál szempont volt, hogy eltérő tesztelési problémákat reprezentáljanak, ugyanakkor a programkód mérete és összetettsége lehetővé tegye az egyes hibák és az azok felismeréséhez szükséges tesztesetek egyértelmű meghatározását. A vizsgált programok többek között szállítási költség számítását, kedvezmény meghatározását, életkor szerinti kategorizálást, jelszó-ellenőrzést, osztályzat meghatározását, szököv-ellenőrzést és háromszög-osztályozást valósítottak meg.

Minden feladathoz egy helyes referencia-implementáció, valamint három, egyetlen szándékosan beépített hibát tartalmazó mutáns készült. A vizsgálati készlet így összesen 30 hibás programváltozatot tartalmazott. A mutációk között határértékhibák, hibás relációs operátorok, hiányzó vagy helytelen feltételágak, valamint speciális bemenetek nem megfelelő kezelése szerepelt. A mutánsokat a kísérlet megkezdése előtt célzott tesztesetekkel validáltuk, és mind a 30 esetben igazoltuk, hogy viselkedésük legalább egy bemenet esetében eltér a referencia-implementációtól.

A nagy nyelvi modell minden esetben a helyes referencia-implementáció forráskódját és annak természetes nyelvű specifikációját kapta bemenetként. A generált tesztesetek futtatása egységes Jest környezetben történt. A tesztesetek generálásához az OpenAI GPT-5.6 Sol (gpt-5.6-sol) modelljét használtuk a Responses API-n keresztül, reasoning.effort=none beállítással. A futtatások egymástól függetlenül, korábbi beszélgetési kontextus átadása nélkül történtek.

3.2. Vizsgált promptstratégiák

A kísérlet három, fokozatosan növekvő módszertani iránymutatást tartalmazó promptstratégiát hasonlít össze.

A P1 alap-prompt kizárólag a tesztesetek elkészítését kéri a modelltől:

„Készíts Jest teszteseteket az alábbi JavaScript-függvényhez a megadott specifikáció alapján.”

Ez a stratégia referenciaértékként szolgál, és nem tartalmaz külön teszttervezési útmutatást.

A P2 strukturált prompt már meghatározza a vizsgálandó tesztesetek általános kategóriáit. A modell azt az utasítást kapja, hogy a normál működés ellenőrzése mellett pozitív és negatív eseteket, határhelyzeteket és speciális bemeneteket is vizsgáljon, valamint törekedjen a program feltételes végrehajtási útvonalainak lefedésére. A prompt ugyanakkor nem nevezi meg az alkalmazandó formális teszttervezési technikákat.

A P3 módszertanvezérelt prompt explicit módon előírja hagyományos teszttervezési módszerek alkalmazását:

„A tesztesetek megtervezése során alkalmazz ekvivalenciaosztály-particionálást, határérték-elemzést és negatív tesztelést. Először azonosítsd a releváns bemeneti osztályokat és határértékeket, majd ezek alapján készíts Jest teszteseteket.”

A három stratégia így az általános tesztgenerálási utasítástól a strukturált megközelítésen keresztül az explicit tesztelési módszertan alkalmazásáig terjed. A promptokban kizárólag a módszertani útmutatás változik; a programkód és a specifikáció minden esetben azonos.

3.3. A kísérlet végrehajtása és kiértékelése

A nagy nyelvi modellek válaszainak nemdeterminisztikus jellegéből adódóan minden program és promptstratégia kombinációját öt alkalommal futtatjuk. A teljes kísérlet így

$$10 \text{ program} \times 3 \text{ promptstratégia} \times 5 \text{ ismétlés} = 150$$

független tesztgenerálási futásból áll. Az egyes futtatások között az előző beszélgetésből vagy eredményből származó kontextus nem kerül átadásra a modellnek.

A generált tesztkészletek minőségét négy szempont szerint értékeljük. Elsődleges mutatóként a hibafeltérési arányt alkalmazzuk:

$$\text{FDR} = \text{feltárt hibák száma} / \text{beépített hibák száma}.$$

Egy hibát akkor tekintünk feltártnak, ha legalább egy generált tesztet olyan módon bukik meg a hibás implementáción, hogy ugyanaz a teszt a javított referencia-implementáción sikeresen végrehajtható.

A tesztkészletek szerkezeti hatékonyságának vizsgálatához az utasítás- és áglefedettséget is meghatározzuk. Ezek mellett rögzítésre kerül a generált és sikeresen végrehajtható tesztesetek száma. Annak vizsgálátára, hogy a jobb hibafeltérési eredmény nem kizárólag nagyobb számú tesztet generálásából következik-e, a teszthatékonyságot a következő mutatóval jellemezzük:

$$\text{TE} = \text{feltárt hibák száma} / \text{generált tesztesetek száma}.$$

Az öt ismétlés eredményeit promptstratégiánként átlag és szórás segítségével értékeljük. Az összehasonlítás során elsősorban azt vizsgáljuk, hogy a strukturált, illetve a klasszikus teszttervezési technikákat explicit módon előíró promptok növelik-e a hibafeltérési arányt és a kódlefedettséget, illetve hogy az esetleges javulás együtt jár-e a generált tesztesetek számának jelentős növekedésével.

4. EREDMÉNYEK ÉS ÉRTÉKELÉS

4.1. A promptstratégiák összehasonlítása

A promptstratégiák összehasonlítása előtt a vizsgálatához létrehozott programozási feladatok és hibaváltozatok validálására került sor. A tíz JavaScript-feladathoz feladatonként három, egyetlen szándékosan beépített hibát tartalmazó mutáns készült, így a vizsgálati készlet összesen 30 hibás programváltozatot tartalmazott. A mutációk elsősorban határértékhibákat, hibás relációs operátorokat, hiányzó feltételeket, valamint speciális bemenetek helytelen kezelését reprezentálták.

A mutánsok alkalmasságát a referencia-implementációkhoz viszonyítva célzott tesztesetekkel ellenőriztük. A validálás célja annak kizárása volt, hogy a vizsgálatban ekvivalens, vagyis a helyes implementációtól megfigyelhető viselkedésében nem különböző mutáns szerepeljen. Mind a 30 hibás programváltozathoz azonosítható volt legalább egy olyan bemenet, amelynél a mutáns viselkedése eltért a referencia-implementációtól. A validáció során így mind a 30 mutáns feltárható volt, ami 100%-os referencia-hibafeltérési aránynak felel meg.

A további vizsgálatban a nagy nyelvi modell a helyes referencia-implementációt és annak természetes nyelvű specifikációját kapja bemenetként. A generált tesztkészleteket először a referencia-implementáción futtatjuk, és csak a hibamentesen végrehajtható tesztkészleteket értékeljük tovább. Ezt követően ugyanazok a tesztek futnak a három egyhibás mutánsra. Egy mutáns akkor tekinthető feltártnak, ha legalább egy olyan tesztet bukik meg rajta, amely a referencia-implementáción sikeresen végrehajtható. Ez az eljárás lehetővé teszi, hogy a különböző promptstratégiák hibafeltérő képessége egyértelműen összehasonlítható legyen.

A validált mutánskészlet alkalmazásával biztosítható, hogy a később mért különbségek ne a hibamodellek eltérő jellegéből, hanem elsősorban a generált tesztkészletek minőségéből adódjanak. Ezáltal a három promptstratégia összehasonlítása egységes és reprodukálható feltételek mellett végezhető el.

A vizsgált promptstratégiák összesített eredményei

1. táblázat

Mutató	P1	P2	P3
Érvényes tesztkészletek	92%	68%	78%
Hibafeltérési arány	100%	100%	100%
Tesztesetek átlagos száma	21,93 ±4,02	35,47 ±7,34	36,28 ±12,00
Teszthatékonyság	0,142±0,029	0,089±0,022	0,095±0,039

A promptstratégiák között elsősorban a generált tesztkészletek végrehajthatóságában és méretében mutatkozott különbség. A legnagyobb arányban a P1 alap-prompt eredményezett a referencia-implementáción hibamentesen végrehajtható tesztkészletet (92%), míg P2 esetében ez 68%, P3 esetében 78% volt.

A végrehajtható tesztkészletek hibafeltáró képessége ugyanakkor mindhárom stratégiánál azonosnak bizonyult: valamennyi érvényes tesztkészlet mindhárom, az adott feladathoz tartozó mutánszt feltárta. A kód-lefedettség szintén minden stratégia esetében nagyon magas volt.

A részletesebb promptok lényegesen több tesztet eredményeztek. A P1 átlagosan 21,93, míg P2 és P3 35,47, illetve 36,28 tesztet generált. Mivel ez nem járt együtt további hibafeltérési javulással, a legnagyobb teszthatékonyságot az egyszerű P1 stratégia érte el. Az eredmények alapján a vizsgált, kis méretű programozási feladatok esetében a részletesebb tesztelési útmutatás nem eredményezett jobb hibafeltérést, ugyanakkor nagyobb tesztkészletekkel és alacsonyabb végrehajthatósággal járt együtt.

5. ÖSSZEFOGLALÁS

A kutatás különböző részletezettségű promptstratégiák hatását vizsgálta nagy nyelvi modell által generált tesztkészletek minőségére. A vizsgált feladatokon a végrehajtható tesztkészletek hibafeltérési aránya mindhárom promptstratégia esetében 100% volt, így a részletesebb módszertani útmutatás nem eredményezett további javulást a hibafeltérásban. Ugyanakkor a strukturált és módszertanvezérelt promptok lényegesen több tesztet generáltak, miközben a referencia-implementáción hibamentesen végrehajtható tesztkészletek aránya csökkent. A vizsgált feladatokon ezért az egyszerű alap-prompt bizonyult a leghatékonyabbnak.

A vizsgálat egyik korlátja a feladatok kis mérete és a hibamodellek korlátozott összetettsége. A valamennyi érvényes tesztkészletnél elért teljes hibafeltérás arra is utal, hogy a kialakított mutánskészlet nem különítette el kellő mértékben a promptstratégiák hibafeltéró képességét. További vizsgálatokban ezért összetettebb programok, nehezebben feltérható mutációk és több nagy nyelvi modell alkalmazása indokolt.

IRODALMI HIVATKOZÁSOK

- [1] Alshahwan N., Chheda J., Finogenova A., Gokkaya B., Harman M., Harper I., Marginean A., Sengupta S., Wang E.: *Automated Unit Test Improvement using Large Language Models at Meta*. Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering. ACM, 2024, 185–196.
- [2] Lemieux C., Inala J. P., Lahiri S. K., Sen S.: *CodaMOSA: Escaping Coverage Plateaus in Test Generation with Pre-trained Large Language Models*. 45th IEEE/ACM International Conference on Software Engineering, (ICSE). IEEE, 2023, 919–931.
- [3] Moradi Dakhel A., Nikanjam A., Majdinasab V., Khomh F., Desmarais M. C.: *Effective test generation using pre-trained Large Language Models and mutation testing*. Information and Software Technology. Elsevier, 2024, 171, 107468.
- [4] Ouédraogo W. C., Kaboré A. K., Li Y., Tian H., Koyuncu A., Klein J., Lo D., Bissyandé T. F.: *Prompt engineering in LLMs for automated unit test generation: A large-scale study*. Empirical Software Engineering, Springer Nature, 2026, 31(4), 1–58.
- [5] Rodríguez M., Rossi G., Fernandez A.: *Evaluating Large Language Models for the Generation of Unit Tests with Equivalence Partitions and Boundary Values*. In: Cloud Computing, Big Data and Emerging Topics, 13th Conference JCC-BD&ET 2025. Springer, 2025, 46–60.
- [6] Ryan G., Jain S., Shang M., Wang S., Ma X., Ramanathan M. K., Ray B.: *Code-Aware Prompting: A Study of Coverage-Guided Test Generation in Regression Setting using LLM*. Proceedings of the ACM on Software Engineering. ACM, 2024, 1(FSE), 951–971.
- [7] Schäfer M., Nadi S., Eghbali A., Tip F.: *An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation*. IEEE Transactions on Software Engineering. IEEE, 2024, 50(1), 85–105.
- [8] Tufano M., Drain D., Svyatkovskiy A., Deng S. K., Sundaresan N.: *Unit Test Case Generation with Transformers and Focal Context*. arXiv preprint, 2020.
- [9] Wang J., Huang Y., Chen C., Liu Z., Wang S., Wang Q.: *Software Testing With Large Language Models: Survey, Landscape, and Vision*. IEEE Transactions on Software Engineering. IEEE, 2024, 50(4), 911–936.
- [10] Yuan Z., Liu M., Ding S., Wang K., Chen Y., Peng X., Lou Y.: *Evaluating and Improving ChatGPT for Unit Test Generation*. Proceedings of the ACM on Software Engineering. ACM, 2024, 1(FSE), 1703–1726.