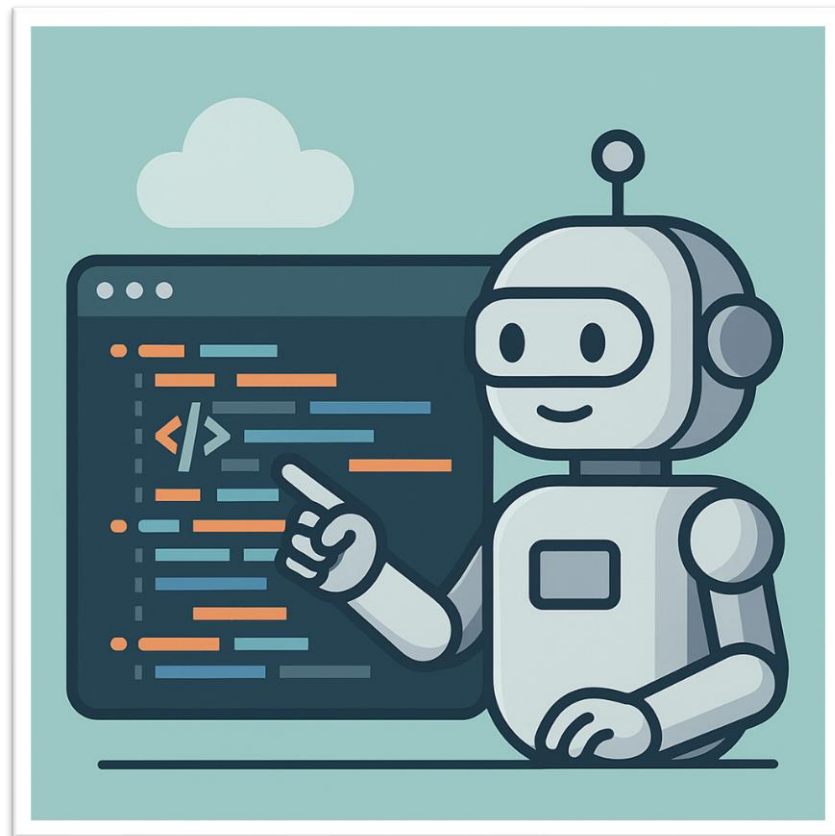


3 JULI 2025



ONDERZOEK NAAR GEBRUIK EN TOEKOMST VAN LLM'S BINNEN NEDERLANDSE IT-ORGANISATIES

SAMENVATTING

Dit onderzoek gaat over hoe Nederlandse IT-organisaties Large Language Models (LLM's) inzetten bij het onderhouden van software. LLM's zijn taalmodellen die getraind zijn op gigantisch veel menselijke taal en deze kunnen interpreteren en genereren. Hierdoor zijn ze steeds aantrekkelijker voor software engineers, omdat ze goed kunnen ondersteunen bij hun taken. Daardoor is er steeds meer vraag naar hoe ze dit het beste kunnen inzetten om hun werkprocessen te optimaliseren. Ook is er vraag vanuit het onderwijs naar welke technieken studenten moeten leren, zodat zij goed voorbereid zijn op de praktijk. Daarom wordt in dit verslag onderzocht hoe deze technieken al worden ingezet, welke voordelen en belemmeringen ontwikkelaars hierbij ervaren, en hoe ze verwachten dit in de toekomst te doen. Om het onderzoek af te bakenen, is er gericht op taken die vallen onder softwareonderhoud.

De hoofdvraag die daarbij geformuleerd is, luidt: **“Wat zijn (mogelijke) toepassingen van Large Language Models (LLM's), relevant voor softwareontwikkelaars in Nederland, met specifieke aandacht voor taken die vallen onder het onderhouden van software?”**. Daarbij zijn deelvragen opgesteld die ingaan op welke onderhoudstaken met LLM's worden ondersteund, welke voordelen en risico's men ervaart, en wat internationale voorbeelden laten zien over de mogelijkheden van LLM's. Deze informatie is verkregen door een enquête uit te zetten onder software engineers in Nederland en door literatuuronderzoek te doen voor de praktijkvoorbeelden.

De resultaten tonen aan dat 71% van de respondenten al gebruikmaakt van LLM's bij onderhoud, met als belangrijkste toepassingen: het debuggen, refactoren en uitleggen van code, en het schrijven van documentatie. Respondenten ervaren vooral tijdsbesparing en efficiëntie als voordelen. Tegelijkertijd zijn ze nog terughoudend door zorgen over privacy, de juistheid van gegenereerde code en beveiligingsrisico's.

Opvallend is dat ontwikkelaars die LLM's vaker gebruiken, deze ook nuttiger vinden. Dit wijst op een leereffect. Daarnaast gaan grote internationale techbedrijven al een stap verder. Zij zetten LLM's in voor automatische bugfixing, performanceoptimalisatie en refactoring van code. Deze voorbeelden laten zien dat het gebruik van LLM's verder kan gaan dan enkel ondersteuning, en zelfs richting automatisering van onderhoud kan groeien.

Op basis van de bevindingen doet dit verslag aanbevelingen voor zowel onderwijsinstellingen als bedrijven. Voor beide partijen is het belangrijk om bewust om te gaan met de nadelen van LLM's. Let op welke data je deelt met de partijen achter de LLM's, kijk kritisch naar de output en controleer deze met betrouwbare bronnen. Op scholen moet ruimte komen om LLM's in te zetten bij opdrachten, zodat studenten kunnen oefenen met deze techniek. IT-bedrijven zouden vooral bij elkaar moeten kijken en toepassingen uitproberen, om deze vervolgens te integreren in hun werkprocessen.

BEGRIPPENLIJST

Begrip	Definitie
LLM	Large Language Model – een AI-model dat getraind is op grote hoeveelheden tekst om natuurlijke taal te interpreteren en te genereren.
Prompt	De invoer die aan een LLM wordt gegeven om een antwoord of output te genereren.
Softwareonderhoud	Het aanpassen, verbeteren of herstellen van bestaande software om bugs te verhelpen, prestaties te verbeteren of veiligheid te waarborgen.
Debuggen	Het opsporen en oplossen van fouten in softwarecode.
Refactoren	Het herstructureren van bestaande code zonder de functionaliteit te veranderen, om de leesbaarheid en onderhoudbaarheid te verbeteren.
Performance verbeteren	Het verbeteren van de snelheid, efficiëntie of schaalbaarheid van software.
Context	De informatie die aan een LLM meegegeven wordt om een goed antwoord te krijgen.

INHOUDSOPGAVE

Samenvatting.....	
Begrippenlijst.....	
1 Inleiding	4
1.1 Context en aanleiding	4
1.2 Doel rapport	5
1.3 Leeswijzer	5
2 Onderzoeksopzet.....	6
2.1 Doelstelling, hoofd- en deelvragen	6
2.2 Methode van onderzoek	7
3 Resultaten.....	8
3.1 Deelvraag 1 - <i>Welke toepassingen van LLM's worden er al ingezet voor het onderhouden van software binnen Nederland?</i>	8
3.1.1 Resultaten enquête - Toepassingen van LLM's bij softwareonderhoud	10
3.1.2 Resultaten enquête - Huidig gebruik van LLM's bij software onderhoud.....	11
3.2 Deelvraag 2 - <i>Wat zijn de redenen om wel of niet een LLM in te zetten binnen software onderhoud?</i>	13
3.2.1 Resultaten enquête - Voor- en nadelen van gebruik van LLM's	14
3.2.2 Resultaten enquête - Ervaren nut van gebruik van LLM's.....	15
3.3 Deelvraag 3 - <i>Wat zijn nog mogelijke toepassingen van LLM's voor het onderhouden van software?</i>	17
3.3.1 Resultaten enquête - Verwachte toekomstige inzet van LLM's	18
3.3.2 Resultaten bronnenonderzoek – Voorbeelden uit de praktijk.....	19
4 Conclusie	24
5 Advies	26
6 Reflectie.....	28
Verantwoording.....	31
Bronnenlijst	32
Bijlagen.....	33
Bijlage A – Enquête opzet.....	33
Bijlage B – Resultaten open vragen enquête	38
Bijlage C – Chat uitwerking thematische analyse ChatGPT.....	45
Bijlage D – Uitleg Thema's Thematische analyses	64
Bijlage E – Diagrammen vergelijking ervaren nut en toekomst versus huidig gebruik	67

1 INLEIDING

In de inleiding van het rapport wordt verteld waar het onderzoeksrapport over zal gaan. Als eerste wordt toegelicht waar de vraag naar het onderzoek vandaan komt, welk probleem er met dit onderzoek onderzocht wordt, wat er met dit onderzoek bereikt wil worden en hoe dit onderzoeksrapport gelezen kan worden.

1.1 CONTEXT EN AANLEIDING

De opkomst van Large Language Models (LLM's) zoals ChatGPT, GitHub Copilot en DeepSeek heeft grote impact op softwareontwikkeling. Deze modellen kunnen ingezet worden voor taken zoals het genereren van code, het schrijven van documentatie en het opsporen van fouten (Fan et al., 2023). Bedrijven zijn daarom steeds vaker geïnteresseerd in de mogelijkheden van AI om hun ontwikkelproces efficiënter te maken en kosten te besparen.

Toch brengt de inzet van LLM's ook uitdagingen met zich mee. Zo is het vaak onduidelijk wat er met ingevoerde data gebeurt, wat vragen oproept over privacy en veiligheid. Daarnaast kunnen LLM's zogenoemde hallucinaties vertonen: situaties waarin het model foutieve, maar overtuigend klinkende informatie genereert.

Hoewel al veel onderzoek is gedaan naar het gebruik van LLM's bij softwareontwikkeling – zoals bij het schrijven van code – is er nog relatief weinig bekend over hun inzet bij het onderhouden van software (Jin et al., 2024). Dat is opvallend, want uit eerder onderzoek blijkt dat software engineers het grootste deel van hun tijd juist aan onderhoud besteden (Soh et al., 2013). Er is dus behoefte aan meer inzicht in hoe LLM's in dit deel van het ontwikkelproces kunnen worden toegepast.

Daarnaast is het nog onduidelijk in hoeverre softwarebedrijven in Nederland deze technieken al inzetten of overwegen te gebruiken. Ook is niet goed in beeld waarom bedrijven juist wel of niet voor deze technologie kiezen. Tot nu toe richtte onderzoek zich vooral op de technische mogelijkheden van LLM's, terwijl er nog weinig bekend is over de praktische toepassing in bedrijfsprocessen.

Dit inzicht is ook van belang voor het onderwijs. Scholen in Noord-Nederland willen weten welke technieken relevant zijn voor de regio, zodat ze hun curriculum hierop kunnen afstemmen. Vooral bij grote overheidsinstanties zoals CJIB, DUO en RDW – waar veel studenten later gaan werken – is het van belang dat er aansluiting is tussen onderwijs en praktijk.

Om het onderzoek af te bakenen, is ervoor gekozen om te focussen op het gebruik van LLM's bij softwareonderhoud. In tegenstelling tot softwareontwikkeling is hier nog weinig onderzoek naar gedaan. De specifieke onderhoudstaken die in dit onderzoek centraal staan zijn: herschrijven van code, foutopsporing, verbeteren van prestaties en documentatie.

Door de impact van LLM's op deze onderhoudstaken te analyseren, ontstaat er beter inzicht in waar kansen liggen om werkprocessen te verbeteren. Dit helpt zowel bedrijven bij het efficiënter inrichten van hun processen als het onderwijs bij het beter voorbereiden van studenten op de praktijk.

1.2 DOEL RAPPORT

Het doel van dit rapport is om inzicht te geven in hoe IT-organisaties in Nederland Large Language Models (LLM's) inzetten ter ondersteuning en verbetering van hun softwareontwikkelprocessen. De nadruk ligt daarbij op onderhoudstaken binnen softwareontwikkeling.

Daarnaast wordt onderzocht waarom IT-organisaties in Nederland ervoor kiezen om wel of geen gebruik te maken van LLM's. Daarbij komen de overwegingen, voordelen en nadelen aan bod die bij die keuze een rol spelen.

Tot slot richt het rapport zich ook op de toekomst: voor welke onderhoudstaken zouden LLM's mogelijk ingezet kunnen worden en welke kansen biedt dit voor organisaties?

1.3 LEESWIJZER

In dit onderzoeksverslag wordt onderzoek gedaan naar hoe IT-organisaties in Nederland LLM's inzetten bij het onderhouden van software en wat de toekomstverwachtingen hierbij zijn.

In Hoofdstuk 1 wordt het verslag ingeleid, wordt verteld wat het probleem is, waarom dit een probleem is en wat we met dit onderzoeksverslag willen bereiken.

Om gestructureerd dit probleem op te lossen wordt er in Hoofdstuk 2 een onderzoeksopzet beschreven. Hierin is een hoofdvraag geformuleerd met drie deelvragen en de methoden om deze deelvragen te beantwoorden worden gegeven.

Daarna zijn de resultaten die uit het onderzoek zijn gekomen uitgewerkt in Hoofdstuk 3. Per deelvraag wordt de methode uitgebreid beschreven. Vervolgens worden de verzamelde gegevens uit de enquête en bronnenonderzoek overzichtelijk weergegeven in diagrammen met daarbij een analyse met de bevindingen die uit deze data zijn gekomen.

Zodra alle deelvragen zijn beantwoord kan de conclusie gegeven worden in Hoofdstuk 4. Van tevoren wordt nog kort even het onderzoek beschreven en de eerdere bevindingen. Daarna wordt er op basis van deze resultaten antwoord gegeven op de hoofdvraag.

Tot slot volgt er in Hoofdstuk 5 een advies aan zowel softwarebedrijven als de opleiding hbo-ICT met de inzichten die verkregen zijn in dit onderzoek.

Achter in het verslag zijn de bijlagen opgenomen, waaronder de enquête opzet, ruwe antwoorden van de open vragen van de enquête, verantwoording van de thematische analyse en extra toelichtingen bij de thema's gevonden in de analyse.

2 ONDERZOEKSOPZET

Dit hoofdstuk beschrijft de onderzoeksopzet. Het begint met de doelstelling en de daaruit volgende hoofdvraag. Om deze hoofdvraag te kunnen beantwoorden, zijn meerdere deelvragen opgesteld. Vervolgens wordt voor elke deelvraag de gekozen onderzoeksmethode toegelicht.

2.1 DOELSTELLING, HOOFD- EN DEELVRAGEN

Het doel van dit onderzoek is om te bepalen welke (mogelijke) toepassingen van Large Language Models (LLM's), relevant zijn voor softwareontwikkelaars in Nederland, met specifieke aandacht voor taken die vallen onder het onderhouden van software.

Daarbij is de volgende hoofdvraag opgesteld:

Wat zijn (mogelijke) toepassingen van Large Language Models (LLM's), relevant voor softwareontwikkelaars in Nederland, met specifieke aandacht voor taken die vallen onder het onderhouden van software?

Om deze hoofdvraag te beantwoorden zijn een aantal deelvragen geformuleerd die moeten helpen de hoofdvraag te beantwoorden:

Deelvraag 1: *Welke toepassingen van LLM's worden er al ingezet voor het onderhouden van software?*

- *Doel hiervan is het identificeren van toepassingen van LLM's voor het onderhouden van software. Hierdoor kan er een beeld geschetst worden van wat er al gedaan wordt.*

Deelvraag 2: *Wat zijn de redenen om wel of niet een LLM in te zetten binnen software onderhoud?*

- *Het doel van deze deelvraag is dieper in te gaan op wat de redenen zijn om wel of niet een LLM in te zetten voor bepaalde taken, hierdoor kan er gekeken worden op welke vlakken er kansen liggen en wat de risico's zijn.*

Deelvraag 3: *Wat zijn nog mogelijke toepassingen van LLM's voor het onderhouden van software voor softwareontwikkelaars in Nederland?*

- *Doel van de laatste deelvraag is om voorbeelden uit de praktijk te geven, zodat dit als inspiratie aan softwareontwikkelaars in Nederland kan worden weergegeven. Ook worden toekomstbeelden van ontwikkelaars in Nederland in beeld gebracht zodat er kan worden bepaald of er ook acceptatie is.*

2.2 METHODE VAN ONDERZOEK

Deelvraag 1 – Welke toepassingen van LLM's worden er al ingezet voor het onderhouden van software binnen Nederland?

Om er achter te komen hoe softwareontwikkelaars LLM's al toepassen in Nederland zal er een enquête worden afgenomen aan deze groep. In deze enquête wordt er gevraagd hoe ze dit op het moment al gebruiken voor softwareonderhoud en hoe vaak ze dit doen.

De antwoorden zullen worden geanalyseerd door middel van een thematische analyse zodat de meest voorkomende toepassingen naar voren zullen komen. De vraag hoe vaak ze LLM's gebruiken voor softwareonderhoud zal worden gevisualiseerd in een diagram en op basis daarvan zullen analyses worden gemaakt.

Deelvraag 2 – Wat zijn de redenen om wel of niet een LLM in te zetten binnen software onderhoud?

De redenen om een LLM in te zetten zullen ook gevraagd worden in de enquête van deelvraag 1. Om deze redenen op te splitsen zal er eerst gevraagd worden wat de nadelen zijn en daarna de voordelen. Ook wordt er in de enquête gevraagd hoe nuttig zij het gebruik van LLM's vinden bin onderhoudstaken.

De antwoorden voor de redenen zullen weer geanalyseerd worden door middel van thematische analyse, en de vraag hoe nuttig zij het gebruik vinden zal weer worden gevisualiseerd in een diagram en daarna geanalyseerd.

Om te kijken of er ook verschillen zitten tussen hoe nuttig software engineers het gebruik van LLM ervaren bij verschillend gebruik, worden deze twee data met elkaar vergeleken en gevisualiseerd in een diagram, zodat hier ook uitspraken over kan worden gedaan.

Deelvraag 3 – Wat zijn nog mogelijke toepassingen van LLM's voor het onderhouden van software voor softwareontwikkelaars in Nederland?

Als eerste worden de toekomstverwachting van softwareontwikkelaars in beeld gebracht door nog een vraag in de enquête te stellen hierover. Daarin wordt gevraagd of ze verwachten vaker LLM's te gebruiken voor een aantal onderhoudstaken. De resultaten hiervan zullen weer worden weergegeven in een diagram en deze zal ook net zoals bij de vorige deelvraag worden vergeleken met het huidige gebruik.

Om inspiratie te geven aan ontwikkelaars in Nederland te geven hoe andere bedrijven al LLM's hebben toegepast. Zal er door middel van literatuuronderzoek op het internet gezocht worden naar wetenschappelijke artikelen, blogs van ontwikkelaars en andere voorbeelden naar praktijkvoorbeelden. Hier zullen dan naar verwachting 5 toepassingen uit komen en daarvan wordt kort samengevat wat ze gedaan hebben, waarom ze het gedaan hebben en wat het heeft opgeleverd.

3 RESULTATEN

In dit hoofdstuk worden de resultaten van het onderzoek besproken opgesplitst per deelvraag. In elke deelvraag wordt er eerst een korte introductie gegeven met de daarbij gebruikte methodes om de resultaten te verkrijgen. Daarna worden de resultaten weergegeven met daarbij diagrammen die de resultaten overzichtelijk weergeven. Ook worden de resultaten geanalyseerd zodat deze later kunnen worden gebruikt om de hoofdvraag te beantwoorden.

3.1 DEELVRAAG 1 - WELKE TOEPASSINGEN VAN LLM'S WORDEN ER AL INGEZET VOOR HET ONDERHOUDEN VAN SOFTWARE BINNEN NEDERLAND?

Om een uitspraak te kunnen doen over waar nog kansen liggen voor het gebruik van LLM's bij onderhoudstaken door softwareontwikkelaars, moet eerst gekeken worden naar de huidige situatie. Hierdoor kunnen verschillen met de rest van de wereld worden vastgesteld en kan worden bepaald waar voor softwareontwikkelaars in Nederland nog mogelijkheden liggen.

Voor het beantwoorden van deze deelvraag is gebruikgemaakt van een enquête. Deze methode is gekozen om direct informatie te verzamelen bij softwareontwikkelaars uit de praktijk. De aanpak bestond uit drie stappen: het opstellen van de enquête, het verzamelen van de antwoorden en het verwerken van de resultaten.

1. Enquête ontwerp

De vragenlijst (te vinden in Bijlage A) is zo opgezet dat er een aantal gesloten en open vragen gesteld worden, zodat we kwantitatieve data hebben die we makkelijker kunnen analyseren en vergelijken, en kwalitatieve data zodat we wat dieper op de antwoorden in kunnen gaan. De vragen zijn geïnspireerd op de vragenlijst uit Tona et al. (2024). In deze paper worden vragen gesteld over het gebruik van LLM's binnen softwareontwikkeling in het algemeen, met focus op het ontwikkelen van software.

De enquête is gebruikt bij meerdere deelvragen. De vragen die specifiek zijn gesteld om deze deelvraag te beantwoorden, zijn als volgt:

1. **Top of mind:** Als eerste is gevraagd: "Bij welke onderhoudstaken gebruikt u al LLM's en kunt u daar in het algemeen kort iets over vertellen?" Deze vraag is bewust gesteld zonder vooraf voorbeelden van onderhoudstaken te geven, om zo ongefilterde reacties uit te lokken.
2. **Likertschaal:** Vervolgens is gevraagd: "Hoe vaak maakt u al gebruik van LLM's bij de volgende taken?" De taken (zoals code uitleggen, debuggen en documenteren) zijn gebaseerd op Jin et al. (2024). In dat paper worden onderhoudstaken benoemd die veel potentie hebben voor de inzet van LLM's. Respondenten konden hierbij kiezen uit vijf antwoordopties, van *nooit* tot *altijd*.
3. **Open vraag:** Tot slot is gevraagd: "Op welke wijze maakt u precies gebruik van LLM's bij deze taken?" Het doel hiervan nu de respondent meer is geprikkeld om nog een keer op de toepassingen in te gaan, hopende op meer voorbeelden en details.

2. Dataverzameling en Respondenten

De data zijn op twee manieren verzameld bij software engineers in Nederland:

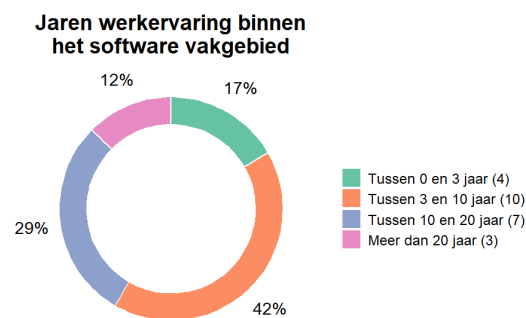
- **E-maillijst NHL Stenden:** Er is een uitnodiging verstuurd naar 140 softwarebedrijven in Noord-Nederland. Dit leverde 4 respondenten op.
- **Platform Prolific:** Via dit online onderzoeksplatform zijn respondenten ingekocht. De steekproef is gefilterd op "*Werksector: Informatietechnologie*" en "*Huidig woonland: Nederland*". Uit een beschikbare pool van ongeveer 150 personen zijn 20 respondenten geworven (Prolific | Easily Collect High-quality Data From Real People, z.d.).

In totaal heeft de enquête een dataset van 24 respondenten opgeleverd.

Om een goed beeld te geven van deze groep, is aan het begin van de enquête gevraagd naar hun werkervaring en de grootte van de organisatie waarvoor ze werken. In Figuur 1 is de verdeling van het aantal jaren werkervaring van de respondenten te zien. Figuur 2 toont de verdeling van de organisatiegrootte, gemeten in het aantal werkzame software engineers.

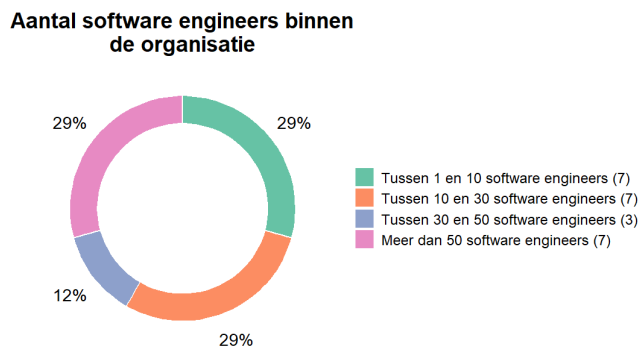
Figuur 1

Jaren werkervaring binnen het software vakgebied



Figuur 2

Aantal software engineers binnen de organisatie



3. Dataverwerking en -analyse

De drie vragen zijn op twee verschillende manieren geanalyseerd. Omdat de eerste en tweede vraag beide gingen over toepassingen van LLM's, zijn de antwoorden samengevoegd en in één keer geanalyseerd.

1. **Analyse van toepassingen van LLM's** Voor de samengevoegde antwoorden van de twee open vragen is een thematische analyse uitgevoerd met behulp van het LLM ChatGPT (GPT-4o, 2025). Dit is in twee stappen gedaan:
 1. **Antwoorden standaardiseren:** Eerst is een prompt ingevoerd met alle antwoorden, met het verzoek deze te standaardiseren en te vertalen naar het Nederlands. Hierdoor bestaat elk antwoord uit vergelijkbare soort woorden en is er meer consistentie in de formuleringen. Dit maakt de thematische analyse in de volgende stap betrouwbaarder.
 2. **Thematische analyse:** Vervolgens is een prompt gegeven om een thematische analyse uit te voeren. Daarbij zijn de genoemde toepassingen gegroepeerd in thema's, zodat duidelijk wordt hoe vaak elk thema voorkomt.

2. **Analyse van het huidige gebruik van LLM's:** De resultaten van de Likertschaal vraag zijn verwerkt in een gestapeld staafdiagram. In deze vraag konden respondenten aangeven hoe vaak zij LLM's momenteel gebruiken, met vijf antwoordopties van nooit tot altijd. In het diagram zijn de verschillen tussen de antwoorden per taak goed zichtbaar.

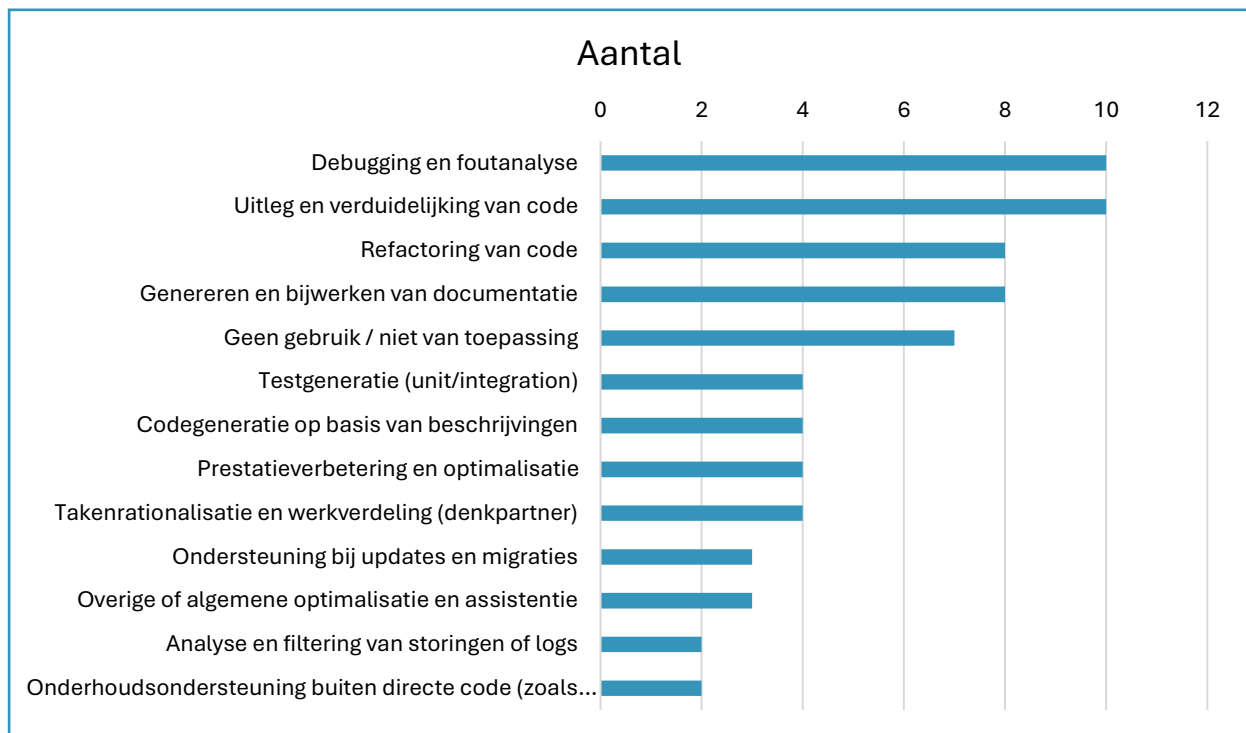
3.1.1 RESULTATEN ENQUÊTE - TOEPASSINGEN VAN LLM'S BIJ SOFTWAREONDERHOUD

Uit de open antwoorden (Bijlage B) is via thematische analyse naar voren gekomen voor welke onderhoudstaken LLM's al worden gebruikt. Deze resultaten zijn weergegeven in Figuur 3. De antwoorden zijn eerst gestandaardiseerd en daarna geanalyseerd met hulp van ChatGPT (GPT-4o). De uitwerking hiervan is te vinden in Bijlage C.

In de grafiek is te zien welke thema's er uit de antwoorden zijn gehaald, met bij elke staaf hoe vaak dat thema genoemd werd. Eén antwoord kon meerdere thema's bevatten als een respondent verschillende toepassingen noemde. In Bijlage D is per thema een korte uitleg te vinden over de gevonden toepassingen.

Figuur 3

Staafdiagram - Thema's toepassingen van LLM's bij softwareonderhoud



Uit de analyse blijkt dat debuggen en uitleggen van code het vaakst genoemd worden, namelijk allebei 10 keer. Daarna volgen refactoreren van code en documentatie genereren of bijwerken, die elk 8 keer genoemd zijn.

Deze vier toepassingen komen ook voor in de Likertschaal vraag. Het kan zijn dat respondenten hierdoor wat gestuurd zijn in hun antwoord. Tegelijkertijd zijn deze taken juist gekozen omdat ze veel potentie hebben voor inzet van LLM's, dus het is ook logisch dat ze vaker genoemd worden.

Bij 7 van de 24 respondenten kwam geen thema naar voren dat echt met onderhoud te maken heeft. Dat betekent dat 17 van de 24 respondenten op dit moment al op een of andere manier gebruiken van LLM's bij softwareonderhoud. Dit geeft aan dat al veel respondenten op een manier gebruikmaken van LLM's bij het onderhouden van software.

Ook worden LLM's al op veel verschillende manieren ingezet en kan het voor veel aspecten binnen het onderhouden van software worden gebruikt. Dit kan te maken hebben doordat veel taken die vallen onder het onderhouden van software te maken hebben met tekst.

3.1.2 RESULTATEN ENQUÊTE - HUIDIG GEBRUIK VAN LLM'S BIJ SOFTWARE ONDERHOUD

In Figuur 4 zijn de resultaten van de vraag over hoe vaak software engineers LLM's gebruiken bij de verschillende onderhoudstaken te zien in een gestapeld staafdiagram. Elke kleur in de staaf geeft een antwoord aan dat kon worden gekozen in de Likertschaal. De grootte van de verschillende kleuren geeft de verhouding aan van het aantal keren dat het antwoord is gegeven.

In het diagram is te zien dat voor het uitleggen en debuggen van code al door de helft van de respondenten vaak een LLM wordt gebruikt, en refactoren van code gaat ook die kant op. Bij het documenteren van code wordt door minder respondenten vaak een LLM ingezet, maar de groep die altijd een LLM bij deze taak inzet is wel groot. De groep die LLM's vaak inzet om performance te verbeteren is nog kleiner, maar de groep die dat soms doet, is wel het grootst.

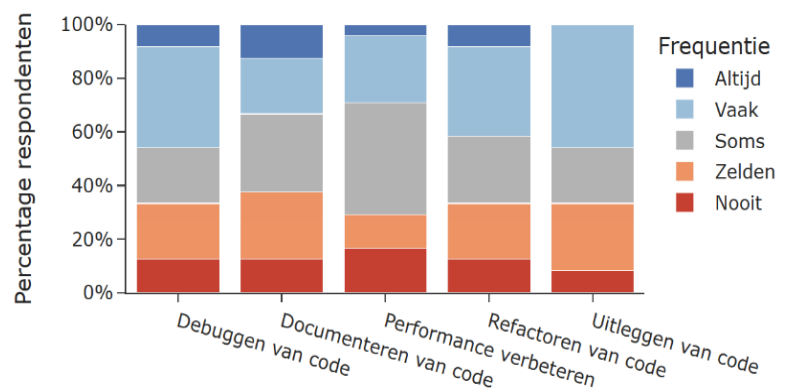
Bij elke taak is de inzet nog vrij verspreid. Wat hierbij opvalt, is dat er voor elke taak nog een groep is die LLM's nooit of zelden inzet bij hun taken. Het zou kunnen dat ze deze taken nooit hoeven uit te voeren en dus ook geen LLM kunnen gebruiken, dat ze het nut er niet van inzien, of niet weten hoe ze ermee moeten beginnen. Dit kan in de volgende deelvraag worden beantwoord.

Om de verschillen in gebruik te analyseren op basis van werkervaring en organisatiegrootte, is per respondent het gemiddelde genomen van de antwoorden op de verschillende taken. Doordat de respondenten nu worden verdeeld over de vier keuzes bij elke analyse, wordt het aantal respondenten per antwoord wel heel klein (kleinste geval 3). Hierdoor zijn de volgende analyses niet heel betrouwbaar, maar ze kunnen toch enige toegevoegde waarde hebben.

Figuur 4

Gestapelde staafdiagram - Huidig gebruik van LLM's bij softwareonderhoud

Huidig gebruik van LLM's bij softwareonderhoud (in %)



In Figuur 5 is te zien dat respondenten met 0 tot 3 jaar werkervaring het vaakst gebruikmaken van LLM's, waarbij 50% van deze respondenten heeft geantwoord met vaak. Het percentage daalt daarna met elke groep werkervaring, totdat het bij respondenten met 20+ jaar werkervaring weer lager ligt. Dit kan komen doordat minder ervaren ontwikkelaars sneller geneigd zijn om nieuwe tools te proberen.

Ook is te zien dat bij de groep respondenten met 20+ jaar ervaring het vaakst zelden en nooit als antwoord werd gegeven. Dit kan komen doordat deze groep een vaste werkwijze heeft waaraan ze niet willen afwijken, of omdat ze minder vertrouwen hebben in LLM's bij onderhoudstaken.

In Figuur 6 is te zien hoe vaak LLM's worden gebruikt in verschillende organisatiegroottes. Voor bedrijven met 1 tot 10 werknemers zijn de antwoorden het verdeeldst. Ook is in deze groep de enige respondent te vinden die nooit een LLM gebruikt. Dit kan komen door persoonlijke voorkeur, of doordat de ontwikkelingen rondom LLM's in kleinere teams minder snel worden opgepikt.

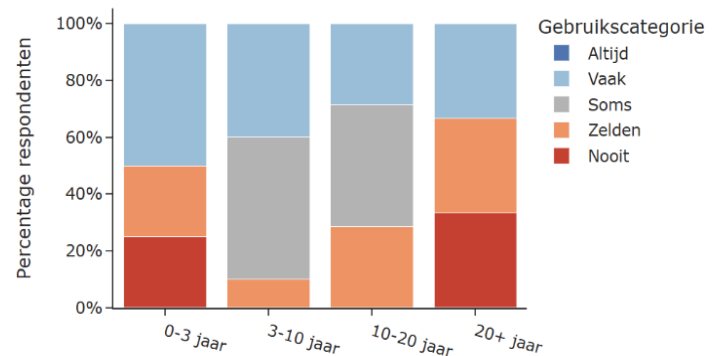
In bedrijven met 10 tot 30 ontwikkelaars wordt LLM-gebruik het vaakst genoemd bij onderhoudstaken. Daarna neemt het gebruik geleidelijk af in de groep 30 – 50, en wordt het heel klein bij bedrijven met 50+ medewerkers. Dit kan komen doordat grotere bedrijven zich meer moeten houden aan centrale richtlijnen, en deze minder makkelijk kunnen aanpassen door hun grootte.

Ook valt op dat in de drie grootste organisatiegroepen geen enkele respondent 'nooit' als gemiddelde antwoord heeft gegeven. Dit betekent dat het gebruik van LLM's in deze groepen in elk geval op enige manier wordt toegepast, en dat het gebruik ervan voorzichtig wordt geaccepteerd in grotere organisaties.

Figuur 5

Gestapelde staafdiagram - Huidig gebruik van LLM's per werkervaring

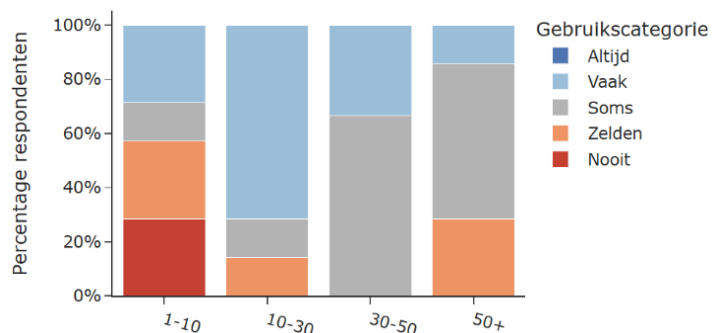
Gemiddeld huidig gebruik van LLM-taken per werkervaring



Figuur 6

Gestapelde staafdiagram - Huidig gebruik van LLM's per organisatiegrootte

Gemiddeld huidig gebruik van LLM-taken per organisatiegrootte



3.2 DEELVRAAG 2 - WAT ZIJN DE REDENEN OM WEL OF NIET EEN LLM IN TE ZETTEN BINNEN SOFTWARE ONDERHOUD?

Om te bepalen wat softwareontwikkelaars aanspoort of juist tegenhoudt om een LLM te gebruiken, zijn we benieuwd naar hun redenen daarvoor. Op basis daarvan kan er gekeken worden of het wel of niet een goed idee is om een LLM in zetten, en er kan achterhaald worden aan welke aspecten van LLM's er nog gewerkt moet worden.

Daarom hebben we in de enquête een aantal vragen opgenomen die deze redenen moeten achterhalen. Eerst wordt gevraagd naar de voor- en nadelen, en vervolgens of ontwikkelaars LLM's nuttig vinden bij het uitvoeren van hun taken.

1. Enquête: voor- en nadelen van het gebruik van LLM's bij softwareonderhoud

Aan de enquête uit deelvraag 1 zijn twee open vragen toegevoegd:

“Wat vindt u de grootste nadelen of risico's van het gebruik van LLM's bij onderhoudstaken?”

“Wat vindt u de grootste voordelen van het gebruik van LLM's bij onderhoudstaken?”

De antwoorden die respondenten hebben gegeven, zijn geanalyseerd met behulp van een thematische analyse, op dezelfde wijze als bij de open vragen van deelvraag 1. Eerst zijn de antwoorden gestandaardiseerd met behulp van ChatGPT (GPT-4o), en daarna zijn de terugkerende thema's geïdentificeerd met een vervolgprompt binnen hetzelfde model. De verwerking hiervan is terug te vinden in Bijlage C.

2. Enquête: ervaren nut van het gebruik van LLM's

Dat een LLM veel wordt gebruikt, betekent nog niet automatisch dat het ook als nuttig wordt ervaren. Daarom is in de enquête van deelvraag 1 ook deze vraag gesteld voor dezelfde taken als de Likertschaal vraag in deelvraag 1:

“Hoe nuttig zijn volgens u LLM's bij de volgende taken?”

Respondenten konden hierbij kiezen uit vijf antwoordopties, van helemaal niet nuttig tot zeer nuttig. De resultaten zijn weergegeven in een divergerend gestapeld staafdiagram, waarin duidelijk zichtbaar is in welke richting de antwoorden neigen.

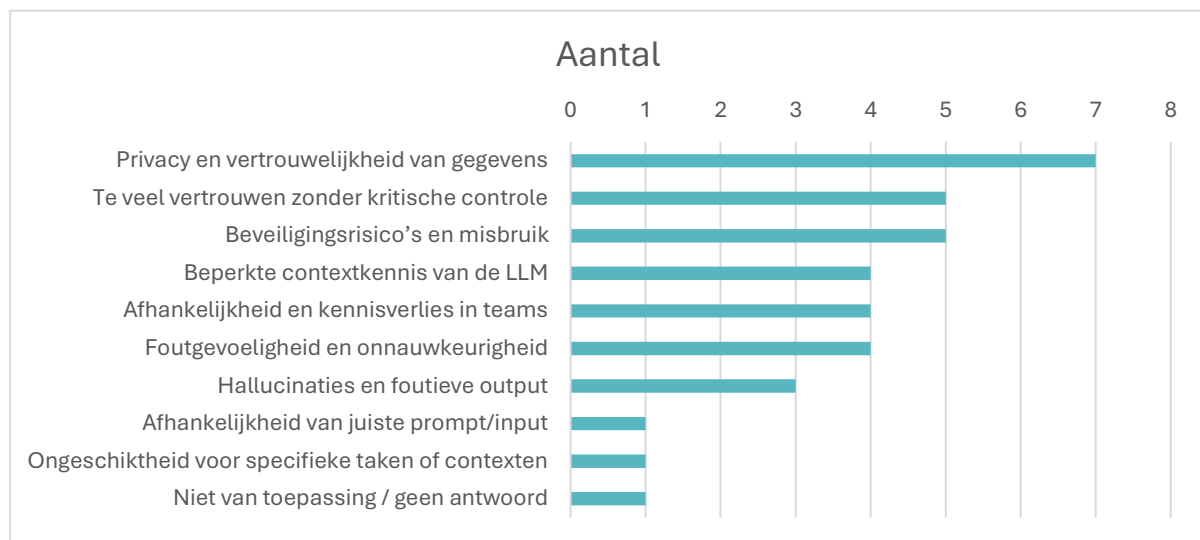
Daarnaast is er een vergelijking gemaakt tussen het gemiddelde huidige gebruik en het gemiddelde ervaren nut. Deze vergelijking is uitgewerkt in een bollendiagram, waarin goed te zien is of er een verband is tussen hoe vaak LLM's worden gebruikt en hoe nuttig dit gebruik ervaren wordt.

3.2.1 RESULTATEN ENQUÊTE - VOOR- EN NADELEN VAN GEBRUIK VAN LLM'S

Nadat de open antwoorden (te vinden in Bijlage B) op de vraag over de nadelen van het gebruik van LLM's bij softwareonderhoud zijn gestandaardiseerd en thematisch geanalyseerd, zijn de thema's geteld en verwerkt in Figuur 7. In dit figuur is per thema te zien hoe vaak het in de antwoorden is genoemd. Een korte uitleg per thema is opgenomen in Bijlage D.

Figuur 7

Staafdiagram - Thema's nadelen van gebruik van LLM's bij softwareonderhoud



In het diagram is te zien dat software engineers nog de grootste zorgen hebben bij de privacy en de vertrouwelijkheid van gegevens (zeven keer genoemd). Dit komt doordat LLM's vaak online moeten worden gebruikt aangeboden door een bedrijf. Om ze te kunnen gebruiken, moet je code of informatie delen met die bedrijven, en het is dan onduidelijk wat er precies met die data gebeurt.

Ook *te veel vertrouwen zonder kritische controle* (vijf keer genoemd) wordt vaak genoemd als nadeel. Daarmee wordt bedoeld dat de output van een LLM er overtuigend uitziet, maar niet altijd klopt. Dit komt doordat LLM's zijn getraind op hoe geloofwaardig een antwoord klinkt, niet per se of dit ook juist is.

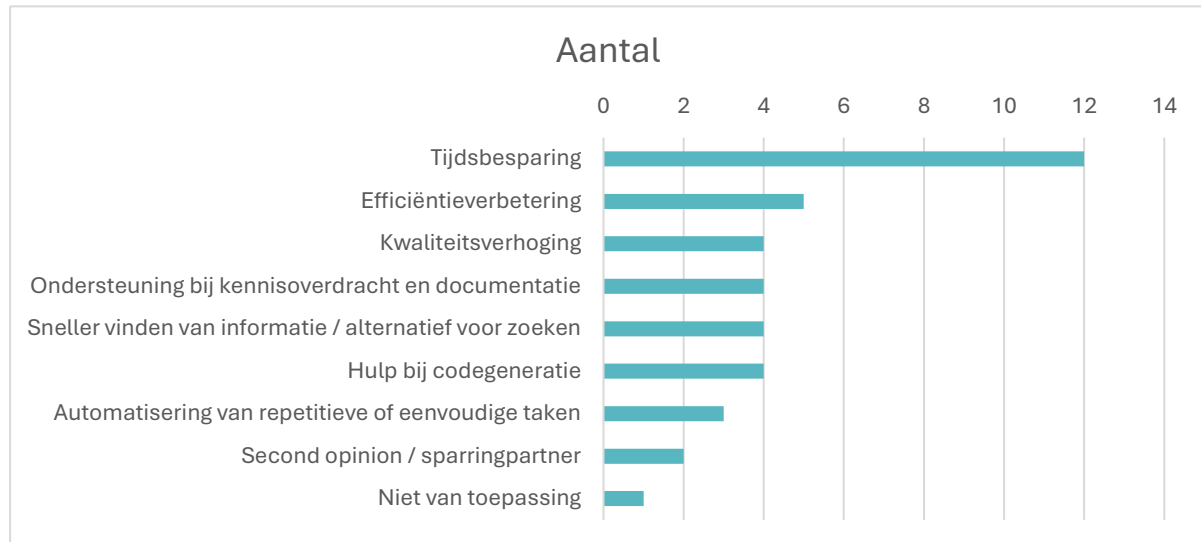
Daarnaast worden ook beveiligingsrisico's en mogelijk misbruik vaak genoemd (ook vijf keer). LLM's kunnen bijvoorbeeld code teruggeven waarin de beveiliging, zoals ontbrekende inputvalidatie of encryptie, niet op orde is. Daarom is het belangrijk dat de output van een LLM altijd goed wordt gecontroleerd.

Wat opvalt, is dat veel van deze nadelen te maken hebben met vertrouwen en controle. Respondenten vinden LLM's wel handig, maar willen ze nog niet zomaar gebruiken zonder alles goed na te kijken, vooral niet bij gevoelige of complexe onderhoudstaken.

Ook zijn de open antwoorden (te zien in Bijlage B) op de vraag over de voordelen van het gebruik van LLM's bij softwareonderhoud geanalyseerd. Dit is gedaan door de antwoorden eerst te standaardiseren en daarna thematisch te analyseren. De resultaten hiervan zijn te zien in Figuur 8. In Bijlage D is ook weer per thema een korte uitleg te vinden.

Figuur 8

Staafdiagram - Thema's voordelen van gebruik van LLM's bij softwareonderhoud



Uit de analyse blijkt dat tijdsbesparing veruit het vaakst genoemd wordt als voordeel (twaalf keer). Respondenten geven aan dat ze door LLM's veel sneller tot oplossingen komen, minder tijd kwijt zijn aan het zoeken of typen van code, en daardoor taken sneller kunnen afronden.

Efficiëntieverbetering (vijf keer genoemd) wordt ook als belangrijk voordeel gezien. Hiermee wordt bedoeld dat LLM's het werkproces soepeler laten verlopen, bijvoorbeeld door automatisch suggesties te doen of stappen te versnellen. Hierdoor kunnen ontwikkelaars zich richten op andere taken.

Kwaliteitsverhoging (vier keer genoemd) is het derde veelvoorkomende voordeel. Sommige respondenten geven aan dat de output van LLM's helpt om fouten te voorkomen of te verbeteren. Ook noemen ze dat LLM's kunnen wijzen op betere of nettere manieren om code te schrijven.

Uit deze voordelen blijkt dat LLM's bij veel verschillende aspecten van softwareonderhoud een waardevolle toevoeging kunnen zijn.

3.2.2 RESULTATEN ENQUÊTE - ERVAREN NUT VAN GEBRUIK VAN LLM'S

In Figuur 9 zijn de resultaten van de vragen over het nut van LLM's gebruik bij de verschillende taken te zien. Het antwoord *neutraal* is steeds in het midden van het diagram geplaatst en de andere antwoorden daaronder en boven. Hierdoor zijn de verschillen goed te zien tussen de verschillende taken.

In het algemeen zijn ontwikkelaars zeer positief over het nut van LLM's, vooral bij het uitleggen en debuggen van code vindt een grote groep dit nuttig of zeer nuttig. Bij het refactoreren en

documenteren van code vinden ontwikkelaars ook heel nuttig maar is de beoordeling net iets lager verdeelt.

Bij het performance verbeteren zijn de meningen het meest verdeelt en is de groep die neutraal en niet nuttig ingevuld heeft het grootst, toch is de groep die zeer nuttig heeft ingevuld nog steeds heel groot. Mogelijk komt dit door dat performance verbeteren moeilijker is wat een LLM mogelijk minder goed aankan, en meer kennis van de ontwikkelaar is vereist.

Er zijn heel weinig respondenten die niet of helemaal niet nuttig beantwoord hebben. Dit kan er op wijzen dat een LLM bijna nooit als helemaal waardeloos gezien wordt, ook niet bij taken waarbij het minder sterk is.

Om te bekijken of het gebruik van LLM's effect heeft op het ervaren nut van LLM's bij software onderhoud, zijn deze twee gemiddelden tegenover elkaar gezet in een bollendiagram (Figuur 10). Op de X-as is het gemiddelde huidige gebruik te zien van alle bevroagde taken in deelvraag 1, en op de Y-as het gemiddelde van al deze taken die hiervoor zijn besproken. De grootte van de bol geeft aan hoe vaak de combinatie van antwoorden voorkomt.

In het diagram is ook te zien wat het lineaire verband is tussen het ervaren nut en het huidige gebruik door middel van de rode lijn. Hieraan is te zien dat er een positief verband is tussen deze twee. Wanneer een ontwikkelaar dus vaker LLM's gebruikt bij onderhoudstaken gaat hij deze nuttiger vinden.

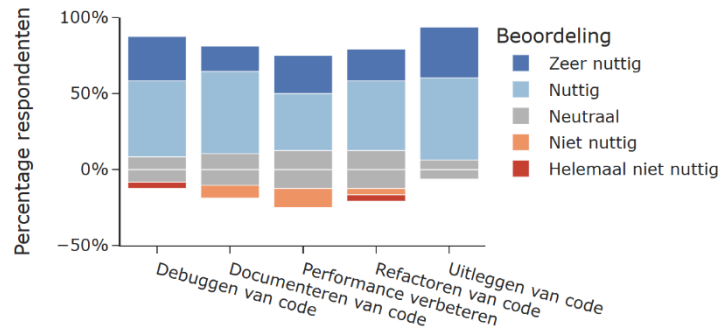
Bij de combinatie nuttig en vaak is de bol het grootst, hieraan is te zien dat mensen die vaak LLM's gebruiken het ook nuttig vinden.

Sommige bollen zoals bij zelden en soms liggen boven de trendlijn. Hier liggen nog kansen voor meer gebruik van LLM's in de toekomst.

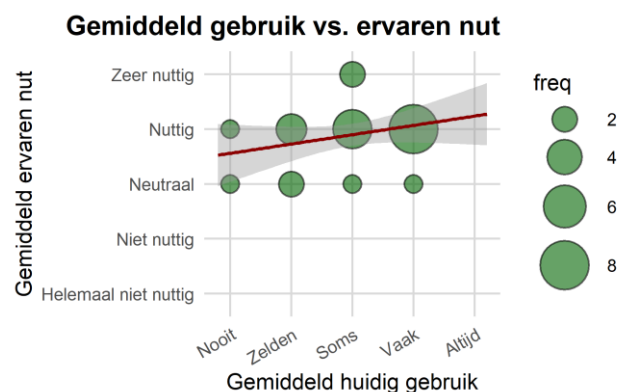
In bijlage D zijn ook het ervaren nut versus het huidige gebruik te zien per taak. Bij de taken debuggen en refactoren van code is het verband heel goed te zien. Vooral bij deze taken zijn ontwikkelaars in het begin heel erg aan het twijfelen over het nut, maar zien er na vaker gebruik steeds meer voordelen in.

Figuur 9
Divergerend gestapeld staafdiagram ervaren nut van gebruik van LLM's bij softwareonderhoud

Ervaren nut van LLM's bij softwareonderhoud (in %)



Figuur 10
Bollendiagram - Vergelijking huidig gebruik met ervaren nut



3.3 DEELVRAAG 3 - WAT ZIJN NOG MOGELIJKE TOEPASSINGEN VAN LLM'S VOOR HET ONDERHOUDEN VAN SOFTWARE?

Door te onderzoeken hoe bedrijven in Nederland LLM's momenteel inzetten, kunnen ze onderling van elkaar leren en kunnen er aanbevelingen worden gedaan. Daarnaast biedt inzicht in de verwachtingen van softwareontwikkelaars informatie over de acceptatie van LLM's en waar nog kansen liggen voor toekomstig gebruik. Om ook inspiratie te halen uit hoe grote internationale bedrijven hiermee omgaan, is er gekeken naar praktijkvoorbeelden.

Om deze deelvraag te beantwoorden zijn twee methoden gebruikt. Allereerst is onderzocht wat de toekomstverwachting is van Nederlandse softwareontwikkelaars, aan de hand van een aantal vragen in de enquête. Daarnaast is er een bronnenonderzoek gedaan naar hoe internationale bedrijven LLM's inzetten bij onderhoudstaken.

1. Enquête: toekomstverwachting van ontwikkelaars

Aan de enquête uit deelvraag 1 zijn vragen toegevoegd waarin per onderhoudstaak is gevraagd:

“In hoeverre bent u van plan in de toekomst LLM's in te gaan zetten voor onderhoudstaken?”

Respondenten konden hierbij kiezen uit vijf antwoordopties, van helemaal niet tot veel meer. De antwoorden zijn weergegeven in een divergerend gestapeld staafdiagram, waarin goed zichtbaar is naar welke kant de antwoorden neigen – dus of er vooral positieve of juist terughoudende verwachtingen zijn per taak.

Omdat deze toekomstverwachting is vergeleken met het huidige gebruik, zijn beide met elkaar in relatie gebracht. Deze vergelijking is weergegeven in een bollendiagram, waarin op de X-as het gemiddelde huidige gebruik staat en op de Y-as de verwachte toekomstige inzet. De grootte van elke bol geeft aan hoe vaak een bepaalde combinatie van antwoorden voorkomt.

2. Bronnenonderzoek: voorbeelden uit de praktijk

Naast de enquête is er een bronnenonderzoek uitgevoerd naar hoe grote internationale softwarebedrijven LLM's al inzetten voor onderhoudstaken. Per taak die ook in de enquête is behandeld, is gezocht naar praktijkvoorbeelden waarbij het gebruik van LLM's duidelijke voordelen opleverde.

Er zijn vijf voorbeelden verzameld. Per voorbeeld is onderzocht:

- Waarom de LLM is ingezet
- Hoe deze is toegepast
- Wat het heeft opgeleverd

Deze voorbeelden zijn samengevat om inzicht te geven in toepassingen die ook relevant of inspirerend kunnen zijn voor bedrijven in Nederland.

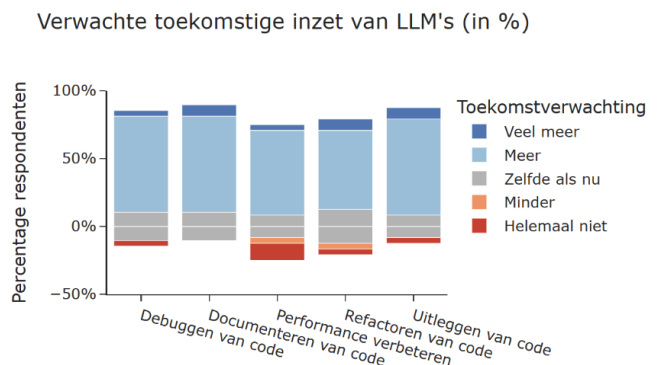
3.3.1 RESULTATEN ENQUÊTE - VERWACHTE TOEKOMSTIGE INZET VAN LLM'S

In Figuur 11 zijn de resultaten te zien van de vraag aan ontwikkelaars in welke mate ze verwachten LLM's in de toekomst in te zetten. Op de X-as staan de onderhoudstaken die in de enquête zijn bevraagd, en op de Y-as het percentage van de gegeven antwoorden. Het antwoord "zelfde als nu" is in het midden van het diagram geplaatst; de overige antwoorden zijn daarboven of daaronder weergegeven.

Uit het diagram blijkt dat bij elke onderhoudstaak het grootste aandeel antwoorden aangeeft dat het gebruik van LLM's in de toekomst zal toenemen. Dit wijst erop dat er over het algemeen positieve verwachtingen zijn over het toekomstige gebruik van LLM's bij deze taken. Vooral bij het documenteren van code, debuggen en het uitleggen van code zijn de verwachtingen hoog.

Bij het verbeteren van performance is het aandeel antwoorden met "helemaal niet" relatief groot. Er zijn dus enkele respondenten die sceptischer zijn over het toekomstig gebruik van LLM's voor deze taak. Ook bij het refactoren van code valt op dat de groep "zelfde als nu" wat groter is dan bij de andere taken. Dit kan erop wijzen dat LLM's voor deze taak al relatief veel worden ingezet en dat er minder groei wordt verwacht.

Figuur 11
Divergerend gestapeld staafdiagram - verwachte toekomstige inzet van LLM's



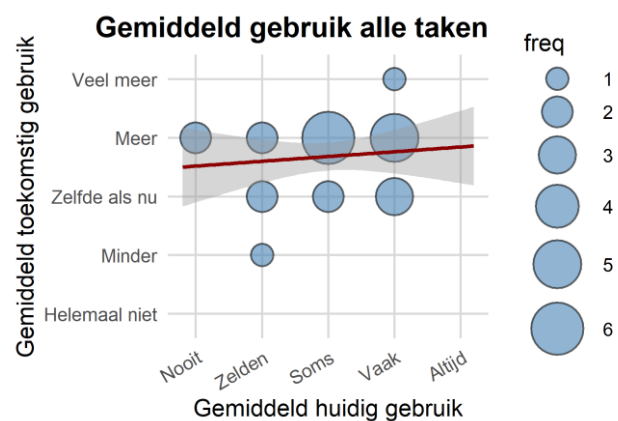
In Figuur 12 is het bollendiagram te zien met de vergelijking tussen het huidige gemiddelde gebruik en het verwachte toekomstige gemiddelde gebruik. Ook is de lineaire regressie weergegeven met behulp van de rode lijn.

Uit het diagram blijkt een lichte positieve relatie tussen het huidige gebruik en het verwachte toekomstige gebruik. Dit betekent dat respondenten die nu al regelmatig LLM's gebruiken, verwachten ze in de toekomst nog vaker te gebruiken dan respondenten die dat momenteel niet doen. Dit kan erop wijzen dat de toegevoegde waarde van LLM's in eerste instantie minder duidelijk is, maar steeds zichtbaarder wordt naarmate men er meer mee werkt.

Daarnaast liggen de gemiddelden op alle gebruiksniveaus hoger dan het huidige gebruik. Dit geeft aan dat vrijwel alle respondenten verwachten LLM's in de toekomst vaker in te zetten bij onderhoudstaken. Bij ontwikkelaars die LLM's nu al vaak gebruiken, is de verwachting zelfs dat het gebruik nog verder zal toenemen. Dit suggereert dat er nog steeds veel groeipotentieel is.

In Bijlage E is de vergelijking per taak terug te vinden. Wat daarbij opvalt, is dat bij het refactoren van code en het verbeteren van performance een sterke positieve relatie zichtbaar is tussen het huidige en toekomstige gebruik. Dit zijn taken waarvoor sommige ontwikkelaars nu nog sceptisch zijn, maar waar waarschijnlijk wel veel potentie ligt.

Figuur 12
Bollendiagram - vergelijking huidig gebruik met toekomstig gebruik



3.3.2 RESULTATEN BRONNENONDERZOEK – VOORBEELDEN UIT DE PRAKTIJK

Hierna worden vijf praktijkvoorbeelden gegeven van de inzet van LLM's. Bij elk voorbeeld wordt eerst toegelicht waarvoor het is gebruikt, vervolgens hoe het is toegepast, en tot slot wat het bedrijf heeft opgeleverd.

3.3.2.1 AIRBNB – GROOTSCHALIGE MIGRATIE VAN TESTFRAMEWORK MET LLM'S

Airbnb heeft LLM's ingezet om ongeveer 3500 React-componenttestbestanden te migreren van de Enzyme-testbibliotheek naar React Testing Library (RTL). In eerste instantie werd geschat dat deze migratie handmatig zo'n anderhalf jaar zou duren, maar door slim gebruik te maken van LLM's wisten ze het proces in slechts zes weken af te ronden.

Er werd een geautomatiseerde pipeline opgezet die elk testbestand stap voor stap verwerkte.

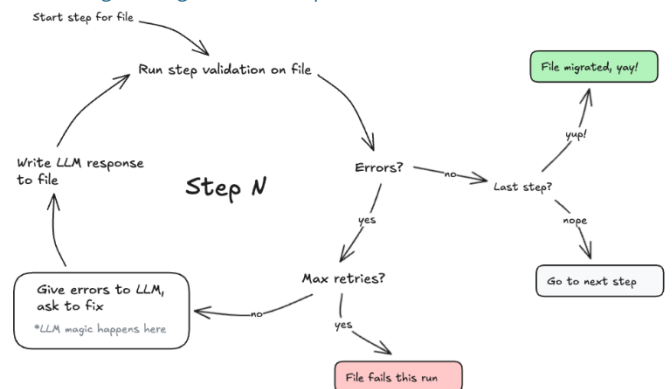
1. In de eerste stap werden Enzyme-tests herschreven naar RTL-tests met behulp van een LLM.
2. Vervolgens werd gecontroleerd of de aangepaste tests nog correct draaiden. Als dat niet het geval was, werd geprobeerd de fout automatisch te herstellen met behulp van een LLM.
3. Tot slot werd gecontroleerd of de stijl en code-opmaak voldeden aan de standaarden van Airbnb. Ook dit werd automatisch aangepast, opnieuw met inzet van een LLM

Als een stap niet succesvol werd uitgevoerd, trad een herhalingslus in werking. Hierbij werd de mislukte stap opnieuw geprobeerd, met als extra invoer de foutmelding uit de vorige poging. Dit proces werd herhaald tot een vooraf ingestelde limiet was bereikt. (Een voorbeeld hiervan is te zien in Figuur 13.)

Wanneer herhalen niet voldoende bleek, werd de prompt uitgebreid met extra context, zoals broncode, gerelateerde tests en migratierichtlijnen, om het model beter te laten begrijpen wat er nodig was.

Figuur 13

Herhalingslus migratie test componenten Airbnb



Door deze aanpak werd in een bulktest al 75% van de bestanden binnen vier uur succesvol gemigreerd. Voor de resterende bestanden werd een andere strategie toegepast: foutmeldingen werden geclusterd om patronen te herkennen, waarna de prompts werden aangepast om deze specifieke problemen beter te verhelpen. Dit iteratieve proces werd vier dagen herhaald, totdat 97% van alle bestanden automatisch gemigreerd kon worden. De laatste 3% is handmatig overgezet, omdat de benodigde aanpassingen voor het model te complex werden.

Dankzij deze aanpak heeft Airbnb aanzienlijk tijd en kosten bespaard vergeleken met handmatige migratie.

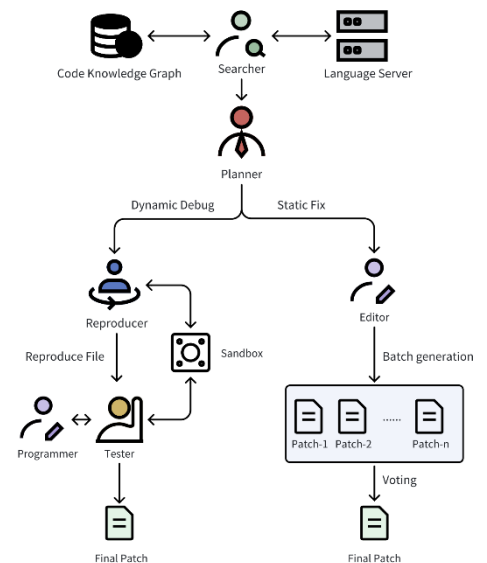
(AirBnB: Large-Scale Test Framework Migration Using LLMS - ZenML LLMOPs Database, z.d.)

3.3.2.2 MARSHCODE AGENT – AUTOMATISCH BUGFIXEN MET LLM AGENTS

ByteDance – het moederbedrijf van TikTok – heeft een tool ontwikkeld genaamd MarsCode Agent, waarmee bugs in bestaande code automatisch kunnen worden opgespoord en opgelost. De tool werkt met meerdere AI-agents die aan elkaar gekoppeld zijn, waarbij elke agent een eigen, specifieke taak uitvoert. Door deze taakverdeling blijft het werk van elke agent overzichtelijk en gericht, waardoor de kans op fouten kleiner wordt. Omdat bugs in veel verschillende vormen kunnen voorkomen, maakt deze opzet het mogelijk om dynamisch te bepalen welke agent het geschiktst is om een bepaald type bug op te lossen.

Het systeem bestaat uit een aantal rollen, die hieronder worden toegelicht (en ook zijn weergegeven in Figuur 14):

Figuur 14
Schematische weergave MarsCode



- **Searcher** – De bug komt binnen bij de Searcher, die met behulp van tools zoals een *code knowledge graph* en het *language server protocol* de bestanden opspoot die gerelateerd zijn aan het probleem.
- **Planner** – Deze agent bepaalt op basis van de complexiteit van de bug of deze dynamisch (via realtime-analyse) of statisch (via directe codeanalyse) moet worden opgelost.
- **Reproducer** – Als de bug dynamisch wordt opgelost, probeert de Reproducer het probleem na te bootsen in een sandbox omgeving, zodat het getest kan worden.
- **Tester** – De Tester voert code uit van zowel de Reproducer als de Programmer, en controleert of de bug is opgelost. Als er nog fouten optreden, worden de foutmeldingen teruggestuurd naar de Programmer. Als alles werkt, wordt de oplossing geaccepteerd.
- **Programmer** – Deze agent past de code aan op basis van de oorspronkelijke probleemomschrijving en de foutmeldingen van de Tester. Daarna stuurt hij een nieuwe oplossing terug. Dit proces herhaalt zich totdat het probleem is opgelost.
- **Editor** – Voor simpelere fouten, zoals veelvoorkomende patronen of syntaxisfouten, wordt de Editor ingezet. Deze genereert meerdere mogelijke oplossingen en kiest de beste optie op basis van een stelsysteem.

MarsCode Agent is getest op een uitdagende benchmarkdataset (*SWE-bench Lite*) die 300 realistische bugs bevat, vergelijkbaar met fouten die softwareontwikkelaars in de praktijk tegenkomen. Van deze 300 bugs wist MarsCode Agent er 34% volledig zelfstandig op te lossen – een aanzienlijke verbetering vergeleken met het gebruik van een enkel model zoals Claude 2 (4,8%) of GPT-4 (1,7%). Dit laat zien dat MarsCode Agent nog niet alles kan oplossen, maar wel een grote stap vooruit is vergeleken met eerdere systemen, en veelbelovend is voor de toekomst van automatisch bug oplossen. (Liu et al., 2024)

3.3.2.3 MERCADO LIBRE – DOCUMENTATIE UITLEGGEN EN GENEREREN

Mercado Libre heeft onderzocht op welke manieren zij LLM's kunnen inzetten om hun werkprocessen te verbeteren. Daaruit zijn drie concrete toepassingen voortgekomen. Mercado Libre is het grootste e-commerce en fintechbedrijf van Latijns-Amerika en wordt vaak omschreven als “de Amazon van Zuid-Amerika”.

De eerste toepassing is een soort interne zoekmachine waarmee ontwikkelaars vragen kunnen stellen over het technische systeem van het bedrijf. Dit is vooral handig voor ontwikkelaars die nog weinig ervaring hebben met de interne systemen. Zij kunnen bijvoorbeeld vragen stellen over waar bepaalde functionaliteit of data zich bevindt. In eerste instantie werkte het systeem goed, maar al snel bleek dat er flinke gaten zaten in de documentatie. Daardoor gaf de LLM soms foutieve of verzonden (gehallucineerde) antwoorden.

Om dat probleem aan te pakken, is een tweede tool ontwikkeld die automatisch documentatie kan genereren, zodat de bestaande systemen beter gedocumenteerd raken. Uit onderzoek bleek dat van de duizenden tabellen een groot deel slecht of helemaal niet was vastgelegd. Informatie werd vaak informeel gedeeld of men dacht dat de tabellen voor zich spraken. Het handmatig bijwerken van deze documentatie was te tijdrovend en kostbaar, dus werd een LLM ingezet.

De LLM kreeg prompts als: “Je bent een expert in documentatie. Schrijf documentatie voor deze tabel op basis van de volgende elementen.” Daarbij werd bestaande informatie toegevoegd over de tabellen en de bijbehorende technische documentatie. Dit leverde documentatie op die door 90% van de stakeholders werd goedgekeurd. De overige 10% vond dat de output te weinig structuur had of dat technische termen ontbraken. Op basis van deze feedback werd verder geïtereerd om ook deze groep tevreden te stellen.

De derde en laatste toepassing richt zich op het begrijpen en structureren van natuurlijke taal. Een voorbeeld hiervan is het herkennen van hoeveel een product bevat (bijvoorbeeld “2000 etiketten verdeeld over 25 vellen”) of het vertalen van een zin als “volgende donderdag” naar een concrete datum. Deze technologie werd toegepast in hun boekingssysteem voor interne dataspecialisten (Data Doctors). Door gebruik te maken van function calling kon het systeem automatisch informatie zoals namen, datums of hoeveelheden uit tekst halen en in een vaste structuur verwerken. Hierdoor hoefden gebruikers geen ingewikkelde formulieren meer in te vullen.

Deze drie voorbeelden laten goed zien dat LLM's veel kunnen, maar alleen als er duidelijke context, structuur en voorbereiding wordt meegegeven. Mercado Libre benadrukt dat je niet altijd het zwaarste model hoeft te gebruiken – vaak is een lichter model voldoende, zolang je de taak goed definieert en het model gericht aanstuurt. (Puerto, 2024)

3.3.2.4 UBER DRAGONCRAWL, MENSELIJKE TESTER

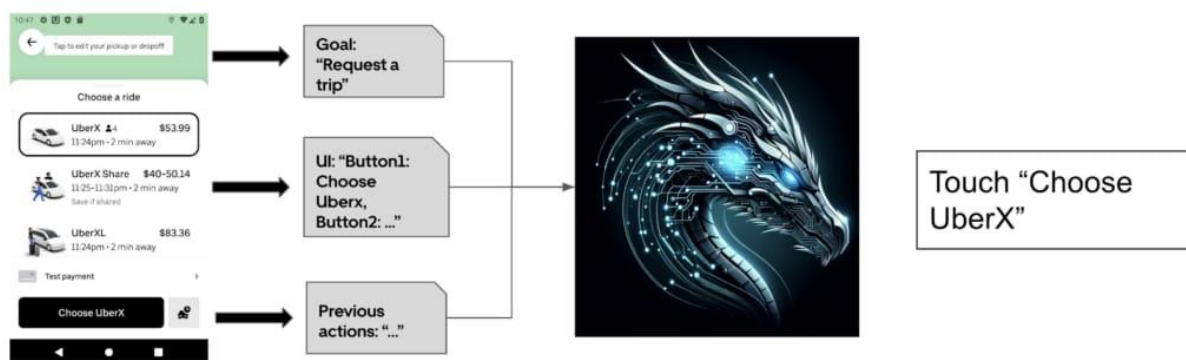
Uber heeft een tool ontwikkeld genaamd DragonCrawl, waarmee ze hun mobiele app automatisch end-to-end kunnen testen op basis van tekstuele doelen.

Uber liep tegen het probleem aan dat het erg lastig was om mobiele apps automatisch te testen. Er waren meer dan 3000 testcases opgesteld die in de oude situatie grotendeels handmatig uitgevoerd moesten worden. Een oplossing daarvoor is het schrijven van automatische testcases, maar zodra de app wordt aangepast – bijvoorbeeld door nieuwe pop-ups of een veranderde flow – moeten al die testcases weer worden aangepast. Hierdoor waren developers zo'n 30 tot 40% van hun tijd kwijt aan testen en onderhoud.

Om dit op te lossen heeft Uber DragonCrawl gemaakt (Figuur 15). Deze tool werkt met een LLM die zelf keuzes maakt op basis van wat er op het scherm staat en wat het doel van de test is. In plaats van een vast script te volgen, kijkt de tool naar de tekst op het scherm, begrijpt wat er moet gebeuren, en voert zelf de juiste acties uit – net als een echte gebruiker. Dit maakt DragonCrawl veel flexibeler dan traditionele testoplossingen en beter geschikt voor veranderingen in de app.

Figuur 15

Globaal overzicht werking DragonCrawl



Voor DragonCrawl gebruikt Uber het model MPNet, dat is getraind met technieken zoals masking en permuting. Hierdoor begrijpt het model niet alleen losse woorden, maar ook beter de structuur van zinnen, wat helpt om de juiste acties te kiezen op basis van de context op het scherm.

DragonCrawl wordt momenteel ingezet om dagelijks de kernfunctionaliteiten van Uber te testen, en heeft inmiddels al veel tijd en geld bespaard. Tot nu toe hebben ze de tool nog niet hoeven updaten of onderhouden. (Marcano, 2024)

3.3.2.5 GOOGLE – AUTOMATISCHE CODEOPTIMALISATIE MET LLM'S

Google heeft een systeem ontwikkeld dat bestaande code automatisch kan analyseren en detecteren waar optimalisaties mogelijk zijn. Daarna genereert het systeem een verbeterde versie van de code, die naar developers wordt gestuurd zodat ze deze kunnen controleren. Ook wordt er gemeten wat het effect van de aanpassing is op de prestaties van de software.

De reden dat Google hiermee is begonnen, heeft te maken met het einde van Moore's Law: de groei in rekenkracht van hardware neemt af. Daardoor wordt het steeds belangrijker om optimalisaties in de software zelf te vinden. Omdat Google gigantische datacenters heeft en codebases met miljarden regels code, kan dat veel winst opleveren. Maar handmatig zoeken naar optimalisaties is te traag en kost te veel tijd. LLM's zijn juist goed in het herkennen van patronen in tekst, en daarom zagen ze hier een slimme oplossing.

Het systeem heet ECO (Efficient Code Optimizer) en werkt in vier stappen:

1. **Patronen vinden**

Eerst worden miljoenen oude codewijzigingen bij Google geanalyseerd om patronen te vinden van code die in het verleden is aangepast voor betere prestaties. Deze zogenoemde *anti-patterns* worden opgeslagen in een database.

2. **Vergelijken met huidige code**

ECO zoekt in de huidige codebase naar vergelijkbare stukken code waar dezelfde soort optimalisatie zou kunnen helpen.

3. **Verbeteringen maken met een LLM**

Vervolgens wordt er met behulp van een LLM, die is getraind op die patronen, automatisch een verbetering gegenereerd.

4. **Controleren en testen**

De verbeterde code wordt automatisch getest. Als er fouten optreden, probeert het systeem de oplossing een paar keer aan te passen. Lukt het dan nog niet, dan wordt de wijziging afgebroken. Als het wel goed gaat, wordt de patch doorgestuurd naar een ontwikkelaar voor een laatste check.

Dankzij dit systeem zijn er al meer dan 25.000 regels code aangepast in ruim 6400 commits, met een succespercentage van 99,5%. Daarnaast bespaart Google hiermee elk kwartaal meer dan 500.000 CPU-core-uren, wat veel tijd en energie bespaard. (Lin et al., 2025)

4 CONCLUSIE

Door de kracht van LLM's, die veel voordelen hebben bij het ontwikkelen van software, is er steeds meer vraag naar inzicht in hoe deze modellen invloed hebben op het werk van softwareontwikkelaars. Er is niet alleen interesse vanuit bedrijven in hoe ze deze technieken kunnen inzetten, maar ook vanuit scholen die willen weten wat ze studenten het beste kunnen leren.

Daarom is er vanuit het ISA-Lab van NHL Stenden een onderzoek gestart naar wat de (mogelijke) toepassingen zijn van Large Language Models (LLM's) die relevant zijn voor softwareontwikkelaars in Nederland. Daarbij is specifiek gekeken naar taken die te maken hebben met het onderhouden van software.

De hoofdvraag luidt: *Wat zijn (mogelijke) toepassingen van Large Language Models (LLM's), relevant voor softwareontwikkelaars in Nederland, met specifieke aandacht voor taken die vallen onder het onderhouden van software?*

Om deze vraag te beantwoorden zijn drie deelvragen opgesteld.

Als eerste is onderzocht welke toepassingen van LLM's er nu al worden gebruikt voor softwareonderhoud door het uitvoeren van een enquête. Hieruit blijkt dat 71% van de ondervraagde softwareontwikkelaars op een bepaalde manier al LLM's inzet bij onderhoudstaken. Vooral bij debuggen en het uitleggen van code worden LLM's vaak gebruikt; 50% van de respondenten geeft aan dit vaak te doen.

Daarna is er gekeken naar de redenen om wel of niet een LLM te gebruiken bij onderhoud. In de antwoorden kwamen een aantal zorgen naar voren, zoals het risico dat gevoelige data lekt, het klakkeloos overnemen van output die goed lijkt, maar niet klopt, en mogelijke fouten in gegenereerde code die voor beveiligingsproblemen kunnen zorgen. Tegelijkertijd werd tijdsbesparing veruit het vaakst genoemd als voordeel; dit gaven de meeste respondenten aan. Ook bleek dat LLM's als nuttiger worden ervaren naarmate ze vaker gebruikt worden. Dat laat zien dat er groei mogelijk is.

Tot slot is er gekeken naar toekomstige toepassingen van LLM's. Veel softwareontwikkelaars verwachten in de toekomst vaker LLM's te gebruiken bij taken zoals het uitleggen, refactoren, debuggen en documenteren van code. In een bollendiagram is te zien dat bij sommige taken het verwachte gebruik duidelijk hoger ligt dan het huidige gebruik, zoals bij performance verbeteren en refactoren van code. Dit zijn taken waar dus nog kansen liggen.

Daarnaast zijn er ook voorbeelden opgezocht door middel van bronnenonderzoek hoe grote internationale bedrijven LLM's al gebruiken. Hieruit bleek dat ze vaak meerdere LLM's combineren of stappen automatiseren, bijvoorbeeld door een foutmelding automatisch mee te geven aan een volgende prompt. Dit levert veel tijdswinst op en biedt inspiratie voor hoe LLM's ook in Nederland slim kunnen worden ingezet.

Om met deze informatie de hoofdvraag te beantwoorden en daarmee het onderzoek af te ronden. LLM's worden al door een groot deel van de softwareontwikkelaars in Nederland

ingezet, ze worden momenteel vaak alleen nog ingezet ter ondersteuning bij taken zoals hulp bij debuggen en het uitleggen van code. Het grootste voordeel dat ze hierbij ervaren is tijdwinst.

Veel ontwikkelaars zeggen in de toekomst vaker LLM's in te zetten bij veel van hun taken. En daarbij ook dat des te meer ze de modellen gebruiken, ze het ook nuttiger gaan vinden. Dit geeft aan dat de verwachtingen voor de toekomst heel positief zijn en er waarschijnlijk nog veel gaat veranderen in de werkprocessen van het onderhouden van software. Vooral wanneer de zorgen over privacy, nauwkeurigheid en betrouwbaarheid beter worden aangepakt.

Ook is te zien dat een aantal grote internationale bedrijven LLM's al veel breder inzet en daarmee onderhoudstaken deels kunnen automatiseren. Deze praktijkvoorbeelden kunnen inspiratie bieden voor Nederlandse bedrijven zodat ze hier ook meer uit kunnen halen.

5 ADVIES

Op basis van de resultaten uit dit onderzoek kunnen een aantal aanbevelingen worden gedaan. In de inleiding is al aangegeven dat er zowel vanuit het onderwijs als vanuit IT-organisaties behoefte is aan duidelijkheid over hoe LLM's momenteel worden ingezet en hoe ze dat in de toekomst het beste kunnen aanpakken. Eerst volgen er aanbevelingen voor het onderwijs over hoe ze hiermee om kunnen gaan. Daarna komen de punten waar IT-bedrijven op moeten letten en wat zij het beste kunnen doen.

Advies aan het IT-onderwijs

- Leer studenten kritisch te zijn op de output die een LLM geeft en de feiten die worden gegeven ook te verifiëren met goede betrouwbare bronnen. Waarschijnlijk zullen programmeurs in de toekomst ook steeds vaker aan het code reviewen zijn en minder aan het code schrijven. Investeer in deze vaardigheden zodat onveilige code output van een LLM niet in productie code terechtkomt.
- Op het moment worden LLM's al op verschillende manieren ingezet bij softwareonderhoud. Geef studenten opdrachten waarbij er ruimte is om te oefenen met deze toepassingen. Ook zou een project waar een van de praktijkvoorbeelden wordt nagebootst een leerzame opdracht zijn, zoals de voorbeelden met het testen en migreren van code. In de opleiding van de studie zit ook een opdracht waarmee er door middel van een parser (programma dat data kan analyseren) informatie uit een database wordt gehaald, dit zou ook kunnen worden gedaan met behulp van een LLM.
- Wijs studenten op de risico's en ethische aspecten van LLM gebruik. Zet geen gevoelige informatie of code in een online model waarvan je het bedrijf dat er achter zit niet vertrouwd. Zo kan deze niet bij kwaadwillige bedrijven terechtkomen. Ook leren bepaalde LLM's van de input van de gebruiker en zou het kunnen voorkomen dat de informatie bij andere gebruikers terecht kan komen.
- Leer studenten hoe een LLM's werken en hoe deze zijn getraind. Zo kunnen studenten achterhalen waar de krachten zitten en voor welke toepassingen bepaalde LLM's het beste voor kunnen worden gebruikt. Hierdoor weten ze ook wat de redenen voor hallucinaties zijn en hoe ze dit kunnen verminderen.

Advies aan IT-bedrijven in Nederland

- Bedrijven zullen zich ook bewust moeten worden van hoe een LLM werkt, hoe je deze goed moet inzetten en wat daarbij de risico's zijn (zoals privacy en beveiligingsrisico's). Organiseer daarom trainingen waarin dit wordt uitgelegd en hoe je goede prompts schrijft. Maak ook bedrijfsvoorschriften waarin wordt uitgelegd welke data je met een LLM mag delen en hoe je met de output moet omgaan. Hierdoor wordt de veiligheid van informatie en code gewaarborgd.
- Kijk hoe andere ontwikkelaars LLM's gebruiken bij softwareonderhoud. Dit kan binnen het eigen team zijn, maar in dit onderzoek staan ook al veel voorbeelden en toepassingen genoemd. Begin klein, zodat de LLM betere output geeft en er goed te zien

is wat er uit komt. Zo kan er veilig ervaring opgedaan worden met de techniek en beter worden inschatten wat het oplevert. Uit dit onderzoek blijkt dat hoe vaker ontwikkelaars ermee werken, hoe nuttiger ze het uiteindelijk vinden, en dat het veel tijdswinst oplevert.

- Kijk naar de praktijkvoorbeelden van grotere bedrijven waarbij al grote winsten behaald zijn. Internationale bedrijven zetten LLM's al in om bijvoorbeeld bij automatisch te testen, documenteren, bug fixen, refactoren van code en performance te verbeteren. Laat je inspireren door deze toepassingen en onderzoek of dit ook bij jullie haalbaar is.

Kortom, LLM's bieden veel kansen, maar vragen wel om bewust en doordacht gebruik. Door te investeren in kennis en ervaring, kunnen zowel het onderwijs als IT-bedrijven goed voorbereid zijn op de toekomst van softwareonderhoud.

6 REFLECTIE

Door terug te kijken op het denken en handelen, kan er geleerd worden wat de volgende keer anders of beter moet. Wat daarbij vooral ook belangrijk is, is om te kijken wat er goed ging en waar dus mee doorgedaan moet worden. Voor dit onderzoeksverslag wordt gebruikgemaakt van de reflectiemethode STARR. Deze methode past hier goed bij, omdat het gericht is op het reflecteren op een specifieke situatie, taak of opdracht.

Stapsgewijs wordt in deze methode aandacht gegeven aan de context, wat er moest gebeuren, wat er is gedaan en wat het heeft opgeleverd. Aan het einde wordt de reflectie gegeven, waarin wordt beschreven welke sterke punten meegenomen kunnen worden naar een volgende situatie en wat er de volgende keer beter of anders kan worden gedaan.

Situatie

Voor de opleiding hbo-ICT moet in het afstudeerjaar de competentie onderzoekend vermogen worden aangetoond. Om deze competentie aan te tonen moet er een methodisch onderzoek worden uitgevoerd. Er zijn daarom een aantal projecten door school gestart waarin een aantal vraagstukken zitten die kunnen worden onderzocht.

Het project dat aan mij is aangeboden, is “gebruik en toekomst van LLM’s binnen Noordelijke IT-organisaties” en is in samenwerking met het ISA-Lab uitgevoerd. In het project is er enerzijds vraag vanuit het onderwijs naar welke technieken ze de studenten moeten leren, en anderzijds vanuit het werkveld naar informatie over welke technieken zij het beste kunnen inzetten. Er is daarom onderzoek gedaan om hier goede inzichten in te leveren.

Taak

Mijn taak daarbij was het onderzoek te ontwerpen, uit te voeren en te rapporteren op een methodische manier. Dit hield onder andere in: het formuleren van hoofd- en deelvragen, het verzamelen en analyseren van data door middel van een onderzoeksmethode, en daarop een conclusie trekken en advies geven.

Actie

Aan het begin van het onderzoek ben ik begonnen met uitzoeken op welke punten binnen het werk van software engineers er vraag is naar inzichten, zodat het onderzoek kon worden ingekaderd en een waardevolle toevoeging zou geven. Daar ben ik toen terechtgekomen bij het onderhouden van software.

Ik heb daarop een onderzoeksplan geschreven waarin het doel was de verschillen te zoeken tussen hoe software engineers binnen Noord-Nederland LLM’s inzetten en daarbuiten. Deze verschillen konden daarna als aanbeveling worden gegeven om de werkprocessen in Noord-Nederland te verbeteren en zo een beeld te kunnen geven van de toekomst.

Om een verschil te kunnen meten moest eerst de huidige situatie onderzocht worden. Ik heb er daarom voor gekozen om een enquête naar IT-organisaties in Noord-Nederland te sturen. Hierin heb ik vragen gesteld zoals welke toepassingen ze gebruiken bij het onderhouden van software.

De enquête is opgestuurd naar een groot aantal contacten van school, maar heeft helaas maar 4 respondenten opgeleverd. Dit was toen wel teleurstellend en daarom ben ik op zoek gegaan naar alternatieven zoals: interviews houden, reclame maken voor de enquête op LinkedIn of direct bij bedrijven afnemen.

Wouter kwam toen met een suggestie, namelijk Prolific: een onderzoeksplatform waar response kan worden ingekocht. Nadat ik deze optie verder heb onderzocht, hebben we besloten dit te doen en het was een groot succes. Na 6 uur hadden we al antwoord van de 20 ingekochte respondenten. Het onderzoeksgebied is in deze stap wel uitgebreid naar heel Nederland, doordat dit de dichtstbijzijnde filteroptie was qua regio.

Het analyseren van de open antwoorden was hierna de grootste uitdaging. Ik heb dit toen gedaan met een LLM, maar wist niet zeker of dit de juiste manier was. Ik heb toen Leo van der Sluis gevraagd om advies. Die gaf mij toen een aantal adviezen en informatie over thematische analyse met behulp van R. Deze analyse heb ik daarna ook geprobeerd, maar ik kwam erachter dat de antwoorden die uit de LLM kwamen veel beter waren dan die met behulp van R. Dit kwam doordat bij R de woorden die per thema worden bepaald ook echt in de antwoorden moeten voorkomen, en bij een LLM niet, doordat die de context veel beter begrijpt. Ik heb daarom ook de andere open vragen met behulp van een LLM geanalyseerd.

Daarna was het plan om te onderzoeken hoe bedrijven buiten Nederland LLM's toepassen bij softwareonderhoud door middel van bronnenonderzoek, zodat de verschillen konden worden bepaald. Dit bleek in de praktijk toch een stuk lastiger, doordat de groep moeilijk te generaliseren was — iedereen doet het weer anders en de groep is erg groot. Ik heb toen gezeten met Mathilde om dit probleem op te lossen en we hebben er toen voor gekozen om 5 praktijkvoorbeelden te geven ter inspiratie voor bedrijven, zodat toch een toekomstbeeld kon worden gegeven.

Er waren ook nog een aantal gesloten vragen in de enquête gesteld over huidig gebruik, nut van gebruik en verwacht toekomstig gebruik voor een aantal taken binnen softwareonderhoud. Om deze te analyseren is er goed nagedacht hoe deze data kon worden gevisualiseerd en in verhouding kon worden gebracht. Ik heb om dit te doen een aantal opties geprobeerd met behulp van R Studio. Dit was wel een stuk ingewikkelder dan met bijvoorbeeld Excel, maar in dat programma liep ik op een bepaald moment tegen de beperkingen van de functionaliteiten aan, waardoor ik op zoek ben gegaan naar alternatieven.

De rest van het verslag is verder zonder al te veel moeilijkheden gemaakt.

Resultaat

Uit het onderzoek is al een goed beeld gekomen van wat softwareontwikkelaars op dit moment doen met AI ter ondersteuning bij onderhoudstaken. Ook is daarbij aandacht geweest voor de redenen om wel of niet een LLM in te zetten en hoe nuttig dat dan is. De praktijkvoorbeelden van grote bedrijven buiten Nederland zijn ook goed gekozen en geven een goede indruk dat er nog veel potentie in de inzet van LLM's zit.

De thematische analyse die is uitgevoerd is heel uitgebreid en geeft al globaal weer voor welke toepassingen LLM's worden gebruikt. Dit geeft aanleiding voor vervolgonderzoeken om hier dieper op in te gaan.

De aandacht voor het visualiseren van de gesloten vragen heeft goed uitgepakt, doordat de diagrammen nu duidelijk en overzichtelijk zijn weergegeven. Hierdoor konden er ook uitspraken worden gedaan over de toekomst en het nut in vergelijking met het huidige gebruik. Het inzetten van Prolific om response op te halen is ook een hele goede stap geweest. Hierdoor zijn er toch nog een groot aantal goede antwoorden binnengekomen en waren er al goede trends te zien. Alleen was de response alsnog vrij klein om echt generalisaties te doen, vooral bij het gebruik per werkervaring en organisatiegrootte.

Uit het onderzoek zijn veel punten en methodes gekomen die kunnen worden ingezet in vervolgonderzoeken, waarin er dieper op een thema kan worden ingegaan met de technieken die in dit onderzoek zijn beschreven.

Reflectie

Aan het einde van het onderzoek kan ik zeggen dat ik heel tevreden ben met het resultaat qua onderzoeksresultaten en wat ik zelf heb geleerd. Ik heb veel geleerd over het afnemen van een enquête en hoe daarbij response kan worden geworven. Ook heb ik veel geleerd over het visualiseren van data en hoe ik deze in het werk stel om uitspraken te doen.

Waar ik in dit onderzoek vooral tegenaan liep was de onderzoeksopzet. Dit was lastig, doordat ik eerst nog geen goed beeld had van softwareonderhoud en hoe die toepassingen eruit konden zien. Hier had ik beter onderzoek naar kunnen doen of beter kunnen inkaderen door een aantal taken binnen softwareonderhoud uit te kiezen. Hierdoor liep het onderzoek wat stroef, omdat het vaak onduidelijk was wat er moest gebeuren. Hoe ik hier uiteindelijk toch uit ben gekomen, is door de beschikbare data te analyseren en hier mijn onderzoeksopzet op aan te passen. Dit is dus ook een leerpunt voor de toekomst: beter kijken welke data je verwacht te verkrijgen uit de deelvragen.

Uiteindelijk vind ik toch dat de vormgeving van het methodisch onderzoeksverslag heel goed is geslaagd, afgezien van dat deelvraag 3 nog iets beter geformuleerd had kunnen worden. Misschien had dit gesplitst kunnen worden in een deelvraag over toekomstverwachtingen van software engineers in Nederland en praktijkvoorbeelden buiten Nederland. De formulering klopt dan misschien niet helemaal, maar het zegt wel allemaal iets over het toekomstige gebruik van LLM's.

VERANTWOORDING

Bij het controleren en schrijven van de tekst van dit onderzoeksverslag is gebruikgemaakt van suggesties van het AI-taalmodel ChatGPT. Het taalmodel is bijvoorbeeld ingezet om te helpen bij het controleren van spelling, grammatica en de structuur van de tekst. De suggesties zijn altijd gecontroleerd op juistheid en waar nodig aangepast voordat ze zijn overgenomen.
(*ChatGPT (GPT-4o)*, 2025)

BRONNENLIJST

AirBnB: Large-Scale Test Framework Migration using LLMS - ZenML LLMOPs Database. (z.d.). Geraadpleegd op 24 juni 2025, van <https://www.zenml.io/llmops-database/large-scale-test-framework-migration-using-llms>

ChatGPT (GPT-4o). (2025). OpenAI. Geraadpleegd op 15 juni 2025, van <https://chatgpt.com/>

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., & Zhang, J. M. (2023, 5 oktober). *Large Language Models for Software Engineering: Survey and Open Problems.* arXiv.org. <https://arxiv.org/abs/2310.03533>

Ferino, S., Hoda, R., Grundy, J., & Treude, C. (2025, 10 maart). *Junior Software Developers' Perspectives on Adopting LLMs for Software Engineering: a Systematic Literature Review.* arXiv.org. <https://arxiv.org/abs/2503.07556>

Jin, H., Huang, L., Cai, H., Yan, J., Li, B., & Chen, H. (2024, 5 augustus). *From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future.* arXiv.org. <https://arxiv.org/abs/2408.02479>

Lin, H., Maas, M., Roquemoire, M., Hasanzadeh, A., Lewis, F., Simonson, Y., Yang, T., Yazdanbakhsh, A., Altinbükten, D., Papa, F., Edmonds, M. N., Patil, A., Schwarz, D., Chandra, S., Kennelly, C., Hashemi, M., & Ranganathan, P. (2025, 19 maart). *ECO: An LLM-Driven Efficient Code Optimizer for Warehouse Scale Computers.* arXiv.org. <https://arxiv.org/abs/2503.15669>

Liu, Y., Gao, P., Wang, X., Liu, J., Shi, Y., Zhang, Z., & Peng, C. (2024, 2 september). *MarsCode Agent: AI-Native Automated Bug Fixing.* arXiv.org. <https://arxiv.org/abs/2409.00899>

Marcano, J. (2024, 29 april). *DragonCrawl: Generative AI for High-quality Mobile Testing | Uber Blog.* Uber Blog. Geraadpleegd op 24 juni 2025, van <https://www.uber.com/en-NL/blog/generative-ai-for-high-quality-mobile-testing/>

Prolific | Easily collect high-quality data from real people. (z.d.). Prolific. <https://www.prolific.com/>

Puerto, J. E. C. (2024, 23 november). *Beyond the Hype: Real-World Lessons and Insights from Working with Large Language Models | by Juan Eduardo Coba Puerto | Mercado Libre Tech | Medium | Mercado Libre Tech.* *Medium.* <https://medium.com/mercadolibre-tech/beyond-the-hype-real-world-lessons-and-insights-from-working-with-large-language-models-6d637e39f8f8>

Soh, Z., Khomh, F., Gueheneuc, Y., & Antoniol, G. (2013). *Towards understanding how developers spend their effort during maintenance activities. 2013 20th Working Conference On Reverse Engineering (WCRE).* <https://doi.org/10.1109/wcre.2013.6671290>

Tona, C., Juárez-Ramírez, R., Jiménez, S., & Durán, M. (2024). *Exploring LLM Tools Through the Eyes of Industry Experts and Novice Programmers. 12th International Conference in Software Engineering Research And Innovation, 313–321.* <https://doi.org/10.1109/conisoft63288.2024.00048>

BIJLAGE A – ENQUÊTE OPZET

Geachte heer/mevrouw,

Wij vragen u om mee te doen aan een onderzoek. Meedoen is vrijwillig, maar dan hebben we wel uw schriftelijke toestemming nodig. Voordat u beslist of u wilt meedoen aan dit onderzoek, krijgt u uitleg over wat het onderzoek inhoudt. Lees deze informatie rustig door en vraag de onderzoeker uitleg als u vragen heeft

Informed consent

1. Titel onderzoek: *Current and future usage of AI in software maintenance within companies*

2. Doel van het onderzoek: Het bepalen welke (mogelijke) toepassingen van Large Language Models (LLM's) relevant zijn voor softwareontwikkelaars of dat gaan worden, met specifieke aandacht voor taken die vallen onder het onderhouden van software.

3. Aanleiding voor het onderzoek: De inzet van kunstmatige intelligentie binnen softwareontwikkeling neemt in rap tempo toe, en er is relatief weinig bekend over hoe LLM's worden toegepast bij onderhoudstaken in IT-organisaties.

4. Wat meedoen inhoudt en wat wordt er van u verwacht: U wordt gevraagd om een enquête in te vullen die ongeveer 10 minuten duurt. Deze enquête richt zich op uw ervaringen en overwegingen met betrekking tot het gebruik van LLM's in onderhoudstaken binnen softwareontwikkeling.

5. Mogelijke voor- en nadelen voor de respondent: Voordelen van deelname zijn dat uw input bijdraagt aan waardevolle inzichten over de toepassing van LLM's in softwareonderhoud. Nadelen kunnen zijn dat het invullen van de enquête tijd kost en dat u mogelijk vragen tegenkomt die u niet kunt of wilt beantwoorden.

6. Als u niet wilt meedoen of wilt stoppen met het onderzoek. U beslist zelf of u meedoet aan het onderzoek. Deelname is vrijwillig. Als u niet wilt deelnemen heeft dat geen nadelige gevolgen voor u. Als u wel meedoet, kunt u zich altijd bedenken en toch stoppen, ook tijdens het onderzoek. U hoeft niet te zeggen waarom u stopt. De gegevens die tot dat moment zijn verzameld, mogen geanonimiseerd (dus niet herleidbaar naar u als persoon) worden gebruikt voor het onderzoek.

7. Einde van het onderzoek.

Uw deelname aan het onderzoek stopt als u de vragenlijst heeft ingevuld. Het onderzoek is afgerond zodra we de gegevens geanalyseerd hebben en deze hebben gepubliceerd.

8. Gebruik en bewaren van uw gegevens

Voor dit onderzoek worden geen persoonsgegevens verzameld. De gegevens (data) worden tijdens het uitvoeren van het onderzoek als volgt opgeslagen, verwerkt en gedeeld: Tijdens dit

onderzoek wordt informatie verzameld en opgeslagen in de vorm van de antwoorden op de vragen die u invult. Deze informatie wordt uitsluitend verzameld en opgeslagen voor het doel van deze onderzoeksstudie. De opgeslagen gegevens zullen worden geanalyseerd en kunnen in geaggregeerde vorm worden gepubliceerd, zoals in een rapport wetenschappelijk paper. Hieronder vallen ook antwoorden op open vragen. Hieruit zal niet direct geciteerd worden, maar wel kan een algemene beschrijving worden gegeven van interessante opmerkingen die voorkomen in de gegeven antwoorden. Ten alle tijden wordt ervoor gezorgd dat er geen persoonlijk identificeerbare informatie wordt onthuld. Na afloop van het onderzoek worden de data als volgt bewaard: 10 jaar lang op de SURF Research Drive (een gesloten omgeving waar data op bewaard kunnen worden, alleen toegankelijk voor de onderzoekers) en waar nodig beschikbaar gesteld aan eventuele controles ten behoeve van de kwaliteit van het onderzoek, zoals aan een accreditatiepanel. De uitkomsten van het onderzoek zullen intern en extern worden gedeeld zoals eerder beschreven. De gegevens die worden gedeeld bevatten geen informatie die tot u te herleiden is. Ook in rapporten en publicaties over het onderzoek zijn de gegevens niet tot u te herleiden.

9. Vertrouwelijkheid van uw gegevens

Om uw privacy te waarborgen worden uw gegevens versleuteld. Binnen NHL Stenden Hogeschool is beperkt toegang tot deze gegevens. Deze personen handelen conform de richtlijnen wetenschappelijke integriteit.

10. Toegang tot uw gegevens voor controle

U heeft het recht om uw gegevens in te zien en u kunt volledig inzage krijgen in de wijze waarop uw gegevens worden verwerkt en bewaard. Om te kunnen beoordelen of het onderzoek op een betrouwbare wijze is uitgevoerd, kunnen leden van een visitatiecommissie inzage krijgen in de niet-versleutelde informatie.

11. Bewaartermijn gegevens

Uw gegevens worden minimaal 10 jaar, volgens de richtlijnen van de Nederlandse Organisatie voor Wetenschappelijk Onderzoek, bewaard door NHL Stenden Hogeschool.

12. Meer informatie over uw rechten bij verwerking van gegevens

Voor algemene informatie over uw rechten bij verwerking van uw persoonsgegevens kunt u de website van de Autoriteit Persoonsgegevens (<https://www.autoriteitpersoonsgegevens.nl/>) raadplegen. Het privacy statement van NHL Stenden Hogeschool vindt u via Privacy Statement NHL Stenden Hogeschool – NHL Stenden Privacy en Security (<https://privacy.nhlstenden.com/index.php/nl/privacy-statement-nhl-stenden-hogeschool/>)

13. Heeft u vragen?

Bij vragen kunt u contact opnemen met Wouter Brinksma (wouter.brinksma@nhlstenden.com).

14. Ondertekening toestemmingsverklaring

Na het delen van deze informatie, wordt u gevraagd te beslissen over deelname aan dit onderzoek. Door uw schriftelijke toestemming geeft u aan dat u de informatie heeft begrepen en instemt met deelname aan het onderzoek.

- Ik ben over het onderzoek geïnformeerd en heb de uitleg hierboven ontvangen, gelezen/bekeken en begrepen.
- Ik ben in de gelegenheid gesteld om vragen over het onderzoek te stellen.
- Ik heb de tijd gehad over mijn deelname aan het onderzoek te kunnen nadenken.
- Ik geef toestemming voor het gebruik van mijn gegevens die tijdens dit onderzoek worden verzameld, verwerkt en gedeeld met derde partijen ten behoeve van dit onderzoek, zoals beschreven in de informatie die ik gekregen heb.
- Ik heb mijn toestemming in volledige vrijheid gegeven. Niemand dwingt mij (zowel niet direct als indirect) om aan dit onderzoek mee te doen.
- Ik ben ervan verzekerd dat ik niet te identificeren ben in door dit onderzoek naar buiten gebrachte gegevens, rapporten, presentaties of artikelen. Mijn privacy en mijn persoonsgegevens zijn gewaarborgd als deelnemer/betrokkene van dit onderzoek.
- Ik begrijp dat ik op elk moment mijn deelname aan het onderzoek kan beëindigen en mijn toestemming kan intrekken op de wijze zoals beschreven in de informatie en ik hoef daar geen reden voor op te geven. De gegevens die tot dat moment zijn verzameld, mogen geanonimiseerd worden gebruikt voor het onderzoek.
- Ik begrijp dat de verzamelde gegevens minimaal 10 jaar, volgens de richtlijnen van de Nederlandse Organisatie voor Wetenschappelijk Onderzoek, door NHL Stenden Hogeschool worden bewaard.

Achtergrond informatie

In het eerste gedeelte van het onderzoek zijn we voornamelijk geïnteresseerd in hoe LLM's worden ingezet door softwareontwikkelaars bij onderhoudstaken. De focus ligt hierbij niet zozeer op *welke* LLM's worden gebruikt, maar op *hoe* ze worden toegepast en bij *welke taken*. We zijn dan ook benieuwd op welke manier u daar op dit moment al gebruik van maakt.

2. Hoe lang bent u al werkzaam binnen het software vakgebied?

- Tussen 0 en 3 jaar
- Tussen 3 en 10 jaar
- Tussen 10 en 20 jaar
- Meer dan 20 jaar

3. Hoeveel software engineers werken binnen uw organisatie?

- Tussen 1 en 10 software engineers
- Tussen 10 en 30 software engineers
- Tussen 30 en 50 software engineers
- Meer dan 50 software engineer

Deel 1 - Onderzoek naar toepassingen van LLM's bij onderhoudstaken

In het eerste gedeelte van het onderzoek zijn we voornamelijk geïnteresseerd in hoe LLM's worden ingezet door softwareontwikkelaars bij onderhoudstaken. De focus ligt hierbij niet zozeer op *welke* LLM's worden gebruikt, maar op *hoe* ze worden toegepast en bij *welke taken*. We zijn dan ook benieuwd op welke manier u daar op dit moment al gebruik van maakt.

4. *Top of mind* - Bij welke onderhoudstaken gebruikt u al LLM's en kunt u daar in het algemeen kort iets over vertellen?

(Gebruikt u nog geen LLM's bij onderhoudstaken, antwoord dan met "n.v.t.")

Open vraag

5. Hoe vaak maakt u al gebruik van LLM's bij de volgende taken?

Debuggen van code : Nooit – Zelden – Soms – Vaak – Altijd

Refactoren van code : Nooit – Zelden – Soms – Vaak – Altijd

Documenteren van code : Nooit – Zelden – Soms – Vaak – Altijd

Performance verbeteren : Nooit – Zelden – Soms – Vaak – Altijd

Uitleggen van code : Nooit – Zelden – Soms – Vaak – Altijd

6. Op welke wijze maakt u precies gebruik van LLM's bij deze taken?

(Gebruikt u nog geen LLM's bij onderhoudstaken, antwoord dan met "n.v.t.")

Open vraag

Deel 2 - Onderzoek naar de voor- en nadelen van het gebruik van LLM's bij onderhoudstaken

In het tweede gedeelte van de enquête zijn we geïnteresseerd in de voor- en nadelen van het gebruik van LLM's bij onderhoudstaken. We willen graag achterhalen wat voor u de redenen zijn om wel of juist niet een LLM in te zetten. Aan het eind van het onderzoek kunnen we op basis daarvan in kaart brengen waar nog kansen liggen voor toekomstige toepassingen.

7. Hoe nuttig zijn volgens u LLM's bij de volgende taken?

Debuggen van code : Helemaal niet nuttig – Niet nuttig – Neutraal – Nuttig – Zeer nuttig

Refactoren van code : Helemaal niet nuttig – Niet nuttig – Neutraal – Nuttig – Zeer nuttig

Documenteren van code : Helemaal niet nuttig – Niet nuttig – Neutraal – Nuttig – Zeer nuttig

Performance verbeteren : Helemaal niet nuttig – Niet nuttig – Neutraal – Nuttig – Zeer nuttig

Uitleggen van code : Helemaal niet nuttig – Niet nuttig – Neutraal – Nuttig – Zeer nuttig

8. Wat vindt u de grootste nadelen of risico's van het gebruik van LLM's bij onderhoudstaken?

Open vraag

9. Wat vindt u de grootste voordelen van het gebruik van LLM's bij onderhoudstaken?

Open vraag

10. In hoeverre bent u van plan in de toekomst LLM's in te gaan zetten voor onderhoudstaken?

Debuggen van code : Helemaal niet – Minder – Zelfde als nu – Meer – Veel meer

Refactoren van code : Helemaal niet – Minder – Zelfde als nu – Meer – Veel meer

Documenteren van code : Helemaal niet – Minder – Zelfde als nu – Meer – Veel meer

Performance verbeteren : Helemaal niet – Minder – Zelfde als nu – Meer – Veel meer

Uitleggen van code : Helemaal niet – Minder – Zelfde als nu – Meer – Veel meer

11. (Optioneel) Zijn er nog andere onderhoudstaken waarvoor LLM's gebruikt kunnen worden die we niet mee hebben genomen in deze vragenlijst?

Open vraag

BIJLAGE B – RESULTATEN OPEN VRAGEN ENQUÊTE

VRAAG 4

Respondent	Bij welke onderhoudstaken gebruikt u al LLM's en kunt u daar in het algemeen kort iets over vertellen? (Gebruikt u nog geen LLM's bij onderhoudstaken, antwoord dan met "n.v.t.")
1	Ik gebruik LLMs vooral voor code-refactoring, het schrijven van automatiseringsscripts en het genereren of bijwerken van documentatie. Ze helpen om repetitieve taken sneller uit te voeren en geven snelle suggesties bij het oplossen van problemen.
2	Migraties, json files genereren, checks waar code efficiënter zou kunnen. Wanneer het nodig is om veel code te schrijven maar de code vrij eenvoudig is dan is het makkelijk om daar een LLM voor te gebruiken.
3	We gebruiken software om onderhoudstaken uit te voeren, maar dat zijn geen LLM's. We gebruiken LLM's ter ondersteuning van developers om met name boilerplate code veel sneller te kunnen schrijven. Ook gebruiken we LLM's wel voor het analyseren van code, maar dat zou ik niet als onderhoud classificeren.
4	Vooraf bij het ontwikkelen en programmeren worden LLM's gebruikt als ondersteuning en hulp. Soms ook wel bij bugfixing en het schrijven van tests.
5	Het geven van commando's om software in een project te updaten. Bijvoorbeeld met behulp van package managers als pnpm.
6	n/a
7	Gegevensopruiming en opsporen van bugs
8	Ik gebruik LLM'S vooral voor het schrijven van code (python, sql). Ook gebruik ik LLM's om voor mij een stappenplan te genereren voor vrijwel alle projecten die ik uitvoer. ideeën over hoe iets het beste te implementeren is ook een van de dingen waar ik het voor gebruik (denk aan hoe het beste een melding te genereren zodat deze zsm bij de developer is bij een cruciale fout of iets dergelijks
9	We use LLM in refactoring of code, test case generation and to update simple code via natural language manipulation
10	Refactoring my code and ask for documentation
11	LLMs worden voornamelijk gebruikt voor repetitieve taken zoals gegevensgeneratie en -opvraging. Daarnaast worden LLMs geraadpleegd in situaties waarin de oplossing niet voor de hand ligt, om te brainstormen en alternatieven te verkennen
12	I used to debug my code and help me to write documentation
13	n.v.t.
14	n.v.t.
15	Ondersteuning bij storingsdiagnose Suggesties voor mogelijke oorzaken op basis van symptomen en historische data. Genereren en bijwerken van documentatie, automatisch opstellen of actualiseren van onderhoudshandleidingen en instructies.
16	n.v.t
17	n.v.t.

18	Ik heb LLM's vooral gebruikt voor het schrijven van documentatie en het verduidelijken van concepten die ik soms moeilijk vind.
19	Ik gebruik het vaak om code te laten genereren voor iets. Dan leg ik uit wat ik wil hebben en vraag ik naar de bijbehorende code. Verder gebruik ik het om code te refactoren. Dan plak ik de code in ChatGPT en vraag ik om een verbeterde variant.
20	Het schrijven van unit tests, of commit messages
21	Wij zetten o.a. Claude Code in (een agentic AI) om relevante onderdelen te vinden binnen een grotere codebase, en daarin kleine fixes te doen. Daarnaast chats met Claude om over problemen te sparren, bij kleinere projecten m.b.v. Repomix (source bundling tool).
22	Bij onderhoudstaken nog niet
23	n.v.t.
24	N.V.T

VRAAG 6

Respondent	Op welke wijze maakt u precies gebruik van LLM's bij deze taken? (Gebruikt u nog geen LLM's bij onderhoudstaken, antwoord dan met "n.v.t.")
1	Voor refactoring vraag ik om schonere of efficiëntere versies van bestaande code. Bij debugging leg ik foutmeldingen voor en vraag om mogelijke oorzaken. Voor documentatie laat ik code verklaren of samenvatten.
2	Wanneer het ontcijferen van code of functies mij wat lastig word dan kan het helpen om een LLM te laten kijken, deze heeft vaker een breder overzicht en zal mij naar de defecte code kunnen wijzen, of impliceren waar het mis gaat zodat ik gerichter kan zoeken.
3	De meeste development omgevingen hebben tegenwoordig koppelingen met LLM's ingebouwd. Op het moment dat een developer bezig is worden al suggesties gedaan voor het completeren of verbeteren van code. Soms laten we twee LLM's allebei een functie genereren en vergelijken we de resultaten door de andere LLM om suggesties voor verbeteringen te vragen.
4	Als ik tijdens het ontwikkelen een probleem tegenkom of een stukje code niet helemaal begrijp dan vraag ik LLM om uitleg. Vaak is dat duwtje genoeg om verder te kunnen. Als de code runt vraag ik vaak om een refactor. Als ik die persoonlijk goedkeur pas ik hem toe.
5	Via Cursor AI, een uitgebreide versie van VS Code laat ik de LLM's meekijken met de code en mij assisteren om die te verbeteren en fouten op te lossen.
6	n/a
7	Na handmatige controle worden LLM's toegepast voor extra filtering
8	Voor het debuggen copy paste ik de error melding in de LLM en volg ik zijn advies op. Met het maken van code geef ik een beschrijving van wat ik wil. Ik geef nooit bestaande code aan de LLM
9	We use LLM to debug code by providing the code snippet and the issue description asking to find the error. Similarly we use it to refactoring by sharing the code and stating our requirements. Explaining code is mainly when a developer picks up a code that was originally

	developed by someone else. Documentation is self explanatory, we just tell what are the key sections to be included in the documentation
10	Basically used LLM to help me in my current task to improve code
11	Ik zal een antwoord geven op elk van de volgende taken: <ol style="list-style-type: none"> 1. Wat betreft debugging, zijn LLMs vooral nuttig bij het lezen van extreem lange debuglogs en het filteren van relevante gegevens. 2. Wat betreft refactoring, worden LLMs soms geraadpleegd voor een efficiëntere aanpak, hoewel ze meestal strategieën voorstellen die minder efficiënt zijn. Echter, LLMs kunnen nuttig zijn bij het opsporen van overbodige delen van de code. 3. LLMs zijn bijzonder behulpzaam bij documentatie, aangezien ze de codebase kunnen scannen, afhankelijkheden kunnen identificeren en in korte tijd een goed gestructureerde documentatie kunnen creëren. 4. Wat betreft prestatieverbetering, zal het antwoord vergelijkbaar zijn met code-refactoring. 5. Wanneer code niet vanzelfsprekend is, kan een LLM inzicht bieden, omdat het de gehele codebase kan scannen en een breder perspectief kan bieden.
12	I am asking to refactor my code or debug the code
13	LLM's zijn bij ons onderdeel van onze codebase en geven suggesties wanneer we code invoeren.
14	n.v.t.
15	Analyseren van storingsmeldingen, automatisch interpreteren en classificeren van tekst in storingsrapporten of tickets
16	n.v.t.
17	We gebruiken LLM's als ondersteuning bij het schrijven van code. Daarnaast haal ik mijn code soms door een LLM om deze te optimaliseren. Na de optimalisatie gaat er nog wel een controle over de code om er zeker van te zijn dat deze correct is. We vertrouwen nooit blind op de LLM. Daarnaast gebruik ik met enige regelmaat een LLM (chatGPT/Claude) om reeds geschreven code uit te leggen.
18	Ik plak een algemene uitleg over wat de functie doet en vraag om documentatie. Ik vraag ook om begeleiding bij het implementeren van bepaalde functies die complexer zijn
19	Zie vorige antwoord. Ook plak ik wel eens code in ChatGPT en vraag ik wat deze code doet.
20	Bij code waar variabelen niet duidelijk zijn kunnen llms dit beter uitleggen
21	<ul style="list-style-type: none"> - Debuggen: met name voor known issues met externe libraries. - Refactoren: code completion, suggesties, feedback - Documenteren: voorzet schrijven volgens richtlijnen - Performance: code completion, suggesties, feedback - Uitleggen: vooral voor het vinden van relevante code of gebruiksaanwijzingen voor libraries (niet business logic)
22	De LLM wordt om suggesties gevraagd.
23	Wij maken (PLC) software voor machine besturingen. Onderhoud is hier niet van toepassing. Deze controllers draaien niet op Windows of vergelijkbare OS's. Als het juist functioneert, kan een machine hier 20 jaar op draaien. Remote access geschiedt via daarvoor geschikte Remote Access Routers (bijv. eWon of IXON). Update daarvan wordt (automatisch) verzorgd door de leverancier.

24	Ik gebruik hier zelf Gemini Pro voor. Mijn 2 collegas gebruiken hier de Github Co-pilot extensie van VS Code voor
----	---

VRAAG 8

Respondent	Wat vindt u de grootste nadelen of risico's van het gebruik van LLM's bij onderhoudstaken?
1	Grootste risico is privacy - je weet niet precies waar de code naartoe gaat of hoe die wordt opgeslagen, vooral bij cloudgebaseerde LLMs.
2	Bij het doorsturen van grote lappen code dan is er snel hallucinatie gevaar. Door de grote hoeveelheid aan code zal er soms in een verkeerde hoek gezocht kunnen worden waardoor je zelf ook op een verkeerde plek kan gaan zoeken.
3	Het is (te) eenvoudig om op het resultaat van een LLM af te gaan zonder nauwkeurige controle. Het grootste risico is dat de developers hun eigen code niet meer begrijpen en er dus ook niet over kunnen redeneren.
4	LLM is in mijn mening niet de meest veilige manier om te ontwikkelen, vooral authenticatie en permission based tasks laakt het nogal.
5	Introduceren van onopvallende fouten die je als mens niet snel zou maken. Het over het hoofd zien van belangrijke zaken of instinkers. De data veiligheid is niet duidelijk gewaarborgd.
6	theft of intellectual property (source code)
7	Grootste nadelen zijn, iets onterecht als fout aanmerken of fouten vergeten
8	Alles wat je erin zet is nu deel van de ai. Oppassen met wat je deelt
9	One is that due to lack of complete business context sometimes they produce flawed or completely incorrect output. Also it prevents lack and loss of knowledge in the team as more and more tasks are handled by LLM
10	May guide you in wrong direction while debugging, output of result maybe wrong
11	Het gedeeltelijk openbaar maken van de codebase, aangezien je je code deelt met de LLM, die mogelijk wordt getraind en later kan worden gebruikt om het systeem te hacken.
12	Sometimes hallucinas and provide wrong code
13	LLM's snappen natuurlijk niet altijd de bedoeling.
14	afhankelijkheid van goede invoer van data
15	Te veel vertrouwen op AI leidt soms tot verlies van praktische kennis en kritisch denkvermogen bij mijn medewerkers
16	Privacy
17	Ga er niet blind van uit dat alles wat de LLM aanpast ook correct is.
18	het grootste risico is waarschijnlijk de vertrouwelijkheid van informatie, ik heb geen garanties dat de informatie die ik deel veilig is
19	ChatGPT snapt niet altijd wat ik bedoel. Verder weet je niet wat er gebeurt met de informatie die je aan ChatGPT geeft.
20	Als het verkeerd is dan kan het je meer tijd kosten
21	Indien niet kritisch geëvalueerd: tijd verspild aan verkeerde zaken / richting, onnodig lange of complexe code, vulnerabilities.

	<p>Juniors laten zich te snel leiden of kunnen minder kritisch controleren.</p> <p>Kloof productiviteit junior / senior (die effectief is met AI) groeit hierdoor en kennis spreidt minder.</p>
22	Het blijkt lastig om de LLM te blijven wantrouwen. De voorstellen zien er op het eerste gezicht goed uit.
23	Ik kan dit niet beoordelen. We passen dit nu minimaal toe.
24	<p>Hallucinaties van de AI zijn voor mij wel echt een doodoener, wanneer je werkt aan een project waar eigenlijk op het internet ook niet veel van te vinden is. Zodra dit gebeurd zie ik bijna elke LLM op het moment zelf zaken verzinnen die er niet zijn.</p> <p>In de toekomst zal dit wellicht beter zijn maar momenteel vind ik ook niet dat een LLM veel helpt bij een monolith codebase. Er is teveel context wat hij dan mist tenzij je heel specifiek zelf een LLM traint op puur je eigen omgevingen.</p> <p>Persoonlijk zou ik een LLM niet daadwerkelijke code laten schrijven of refactoren. Dit is een process waar je zelf goed bij na moet denken</p>

VRAAG 9

Respondent	Wat vindt u de grootste voordelen van het gebruik van LLM's bij onderhoudstaken?
1	LLMs zien patronen of fouten die mensen makkelijk over het hoofd zien - dat bespaart tijd en verhoogt de kwaliteit van onderhoud.
2	Het tempo, wanneer het op de juiste manier word ingezet kan het zo veel tijd schelen. Dagen werk kan soms in enkele minuten uitgezocht worden.
3	Mits op de juiste manier ingezet kan het de snelheid van het development proces enorm verbeteren
4	Snelheid. Waar je vroeger Google moest gebruiken of uren op Stackoverflow moest doorspitten naar een antwoord is LLM heel rap en duidelijk. Ook kun je doorvragen nadat je een antwoord hebt gekregen.
5	Enorme tijdsbesparing.
6	Easy and fast learning. When in doubt just use LLM instead of checking hundreds of github
7	Snel en efficiënt toepassen van llms
8	Het is snel. Je kan het meeste ook vinden via google en stackoverflow, maar een LLM is vele malen sneller met een oplossing
9	It helps us become more efficient. We can focus our time on other more important things. Routine tasks that are less value adding like documentation can be delegated to LLM
10	Help me to coding faster
11	Het elimineren van handmatige en repetitieve taken.
12	Guide me to write my code faster
13	Het scheelt tijd om code op te zoeken als een LLM wel de juiste syntax geeft.
14	automatisering van documentatie

15	Ze kunnen helpen bij het vastleggen van technologische kennis en ervaringen van medewerkers, zodat nieuwe medewerkers sneller toegang hebben tot relevante informatie.
16	Efficiency
17	Tijdsbesparing
18	bespaart je veel tijd en creëert documenten van hoge kwaliteit die je meteen kunt gebruiken
19	Het heeft de potentie om je werk te versnellen. Je maakt minder fouten.
20	Kan als een soort 2nd opinion werken
21	Indien correct toegepast: meer kwaliteit in minder tijd. Tijdsbesparing door: code generation, minder in documentatie hoeven kijken. Kwaliteit door: bredere kennis en meer durf om een idee het LLM in te gooien en te kijken hoe het uitpakt (i.p.v. zelf schrijven met kostbare tijd).
22	De LLM heeft meer kennis dan 1 persoon, waardoor de kwaliteit omhoog kan gaan.
23	Ik kan dit niet beoordelen. We passen dit nu minimaal toe.
24	Vooraf bij het debuggen en uitleggen van een stuk code kan het heel handig zijn om hier uitgebreid een uitleg van te krijgen wat het dan is. Voor debuggen valt er nog wat voor te zeggen als het om vrij edge case zaken gaat dat een LLM dan weer niet handig is maar bij general errors kan je er wel wat mee.

VRAAG 11

Respondent	(Optioneel) Zijn er nog andere onderhoudstaken waarvoor LLM's gebruikt kunnen worden die we niet mee hebben genomen in deze vragenlijst?
1	LLMs kunnen ook helpen bij het brainstormen over hoe je bepaalde code het best kunt opzetten, nog voordat je begint met schrijven.
2	Nvt
3	Nvt
4	
5	Het valideren van code volgens standaarden of andere eisen. Het creëren van nieuwe programma's die er überhaupt nog niet waren, vooral handig voor code die niet meer te onderhouden valt vanwege verouderde afhankelijkheden.
6	
7	
8	It-ers zijn wellicht niet de meest sociale mensen. Ik gebruik het soms om een nette email voor me te maken aan de hand van wat steekwoorden
9	
10	
11	Testen
12	

13	LLM's zijn altijd nuttig om te raadplegen omdat ze vaak invalshoeken bieden waar je zelf niet altijd als eerst aan gedacht hebt.
14	
15	We maken ook gebruik van LLM's voor de perfectieve onderhoud van onze systemen
16	
17	
18	
19	
20	Unit tests
21	<p>Alles raakt wel, maar de vorm/workflow van onderhoud zie ik ook veranderen door AI.</p> <p>Dus: VCS (branching, commit messages), issue tracking, code review.</p> <p>In theorie kan AI sommige kleine issues en RFCs automatisch van een PR voorzien. En PRs reviewen.</p>
22	Het bijwerken van code naar aanleiding van updates in libraries
23	Ervaring ontbreekt om hier een gefundeerd antwoord op te kunnen geven. Wel aan het onderzoeken wat de mogelijkheden zijn.
24	

Je zei:

Standaardiseer de volgende antwoorden van een enquête over LLM-gebruik bij onderhoudstaken. We zijn in deze enquête geïnteresseerd in hoe gebruikers LLM's inzetten bij hun taken en wat hierbij de redenen zijn om ze wel of niet in te zetten. Gebruik korte consistente termen en verwijder irrelevante bijwoorden en herformuleer vage beschrijvingen duidelijk. Vertaal Engelse beschrijvingen naar het Nederlands. Deze antwoorden worden later weer gebruikt als input voor een analyse. De vraag die gesteld is luid: "Bij welke onderhoudstaken gebruikt u al LLM's en kunt u daar in het algemeen kort iets over vertellen?" Kun je de gestandaardiseerde antwoorden zonder opsommingstekens teruggeven (1, 2, 3) en aangeven wanneer niet van toepassing. 1 Ik gebruik LLMs vooral voor code-refactoring, het schrijven van automatiseringsscripts en het genereren of bijwerken van documentatie. Ze helpen om repetitieve taken sneller uit te voeren en geven snelle suggesties bij het oplossen van problemen. 2 "Migraties, json files genereren, checks waar code efficiënter zou kunnen. Wanneer het nodig is om veel code te schrijven maar de code vrij eenvoudig is dan is het makkelijk om daar een LLM voor te gebruiken." 3 We gebruiken software om onderhoudstaken uit te voeren, maar dat zijn geen LLM's. We gebruiken LLM's ter ondersteuning van developers om met name boilerplate code veel sneller te kunnen schrijven. Ook gebruiken we LLM's wel voor het analyseren van code, maar dat zou ik niet als onderhoud classificeren. 4 Vooral bij het ontwikkelen en programmeren worden LLM's gebruikt als ondersteuning en hulp. Soms ook wel bij bugfixing en het schrijven van tests. 5 Het geven van commando's om software in een project te updaten. Bijvoorbeeld met behulp van package managers als npnm. 6 n/a 7 Gegevensopruiming en opsporen van bugs 8 Ik gebruik LLM'S vooral voor het schrijven van code (python, sql). Ook gebruik ik LLM's om voor mij een stappenplan te genereren voor vrijwel alle projecten die ik uitvoer. ideeën over hoe iets het beste te implementeren is ook een van de dingen waar ik het voor gebruik (denk aan hoe het beste een melding te genereren zodat deze zsm bij de developer is bij een cruciale fout of iets dergelijks 9 We use LLM in refactoring of code, test case generation and to update simple code via natural language manipulation 10 Refactoring my code and ask for documentation 11 LLMs worden voornamelijk gebruikt voor repetitieve taken zoals gegevensgeneratie en -opvraging. Daarnaast worden LLMs geraadpleegd in situaties waarin de oplossing niet voor de hand ligt, om te brainstormen en alternatieven te verkennen 12 I used to debug my code and help me to write documentation 13 n.v.t. 14 n.v.t. 15 "Ondersteuning bij storingsdiagnose Suggesties voor mogelijke oorzaken op basis van symptomen en historische data. Genereren en bijwerken van documentatie, automatisch opstellen of actualiseren van onderhoudshandleidingen en instructies." 16 n.v.t 17 n.v.t. 18 Ik heb LLM's vooral gebruikt voor het schrijven van documentatie en het verduidelijken van concepten die ik soms moeilijk vind. 19 "Ik gebruik het vaak om code te laten genereren voor iets. Dan leg ik uit wat ik wil hebben en vraag ik naar de bijbehorende code. Verder gebruik ik het om code te refactoren. Dan plak ik de code in ChatGPT en vraag ik om een verbeterde variant." 20 Het schrijven van unit tests, of commit messages 21 Wij zetten o.a. Claude Code in (een agentic AI) om relevante onderdelen te vinden binnen een grotere codebase, en daarin kleine fixes te doen. Daarnaast chats met Claude om over problemen te sparren, bij kleinere projecten m.b.v. Repomix (source bundling tool). 22 Bij onderhoudstaken nog niet 23 n.v.t. 24 N.V.T

ChatGPT zei:

Hieronder vind je de gestandaardiseerde en opgeschoonde antwoorden op de vraag “Bij welke onderhoudstaken gebruikt u al LLM's en kunt u daar in het algemeen kort iets over vertellen?” Elk antwoord is herformuleerd in korte, consistente termen, met verwijdering van irrelevante bijwoorden en vertaling waar nodig.

1. Refactoring, schrijven van automatiseringsscripts en documentatie. LLM's helpen repetitieve taken versnellen en bieden snelle oplossingsuggesties.
2. Migraties, genereren van JSON-bestanden, efficiëntiecontroles. Inzet bij het schrijven van eenvoudige maar omvangrijke code.
3. LLM's worden gebruikt voor boilerplate code en code-analyse, maar code-analyse wordt niet als onderhoud gezien.
4. Ondersteuning bij programmeren, bugfixing en het schrijven van tests.
5. Gebruikt voor update-commando's in softwareprojecten, zoals met package managers (bijv. npm).
6. Niet van toepassing.
7. Gegevensopruiming en bugdetectie.
8. Schrijven van code (Python, SQL), genereren van stappenplannen, advies over implementatiekeuzes (bijv. meldingen bij fouten).
9. Refactoring, genereren van tests, eenvoudige code-updates via natuurlijke taal.
10. Refactoring en documentatie.
11. Gegevensgeneratie en -opvraging, brainstormen bij complexe problemen en het verkennen van alternatieven.
12. Debuggen en schrijven van documentatie.
13. Niet van toepassing.
14. Niet van toepassing.
15. Ondersteuning bij storingsdiagnose, oorzaakanalyse op basis van symptomen en data, genereren en bijwerken van documentatie en handleidingen.
16. Niet van toepassing.
17. Niet van toepassing.
18. Schrijven van documentatie en verduidelijken van moeilijke concepten.
19. Genereren en refactoreren van code op basis van beschrijvingen.
20. Schrijven van unit tests en commitberichten.
21. Vinden van relevante codeonderdelen en uitvoeren van kleine fixes met Claude Code. Sparren over problemen en kleine projecten ondersteunen via Repomix.
22. Nog geen inzet bij onderhoudstaken.
23. Niet van toepassing.

24. Niet van toepassing.

Je zei:

Standaardiseer de volgende antwoorden van een enquête over LLM-gebruik bij onderhoudstaken. We zijn in deze enquête geïnteresseerd in hoe gebruikers LLM's inzetten bij hun taken en wat hierbij de redenen zijn om ze wel of niet in te zetten. Gebruik korte consistente termen en verwijder irrelevante bijwoorden en herformuleer vage beschrijvingen duidelijk. Vertaal Engelse beschrijvingen naar het Nederlands. Deze antwoorden worden later weer gebruikt als input voor een analyse. De vraag die gesteld is luid: "Op welke wijze maakt u precies gebruik van LLM's bij deze taken?" Kun je de gestandaardiseerde antwoorden zonder opsommingstekens teruggeven (1, 2, 3) en aangeven wanneer niet van toepassing. 1 Voor refactoring vraag ik om schonere of efficiëntere versies van bestaande code. Bij debugging leg ik foutmeldingen voor en vraag om mogelijke oorzaken. Voor documentatie laat ik code verklaren of samenvatten. 2 Wanneer het ontcijferen van code of functies mij wat lastig word dan kan het helpen om een LLM te laten kijken, deze heeft vaker een breder overzicht en zal mij naar de defecte code kunnen wijzen, of impliceren waar het mis gaat zodat ik gerichter kan zoeken. 3 De meeste development omgevingen hebben tegenwoordig koppelingen met LLM's ingebouwd. Op het moment dat een developer bezig is worden al suggesties gedaan voor het completeren of verbeteren van code. Soms laten we twee LLM's allebei een functie genereren en vergelijken we de resultaten door de andere LLM om suggesties voor verbeteringen te vragen. 4 Als ik tijdens het ontwikkelen een probleem tegenkom of een stukje code niet helemaal begrijp dan vraag ik LLM om uitleg. Vaak is dat duwtje genoeg om verder te kunnen. Als de code runt vraag ik vaak om een refactor. Als ik die persoonlijk goedkeur pas ik hem toe. 5 Via Cursor AI, een uitgebreide versie van VS Code laat ik de LLM's meekijken met de code en mij assisteren om die te verbeteren en fouten op te lossen. 6 n/a 7 Na handmatige controle worden LLM's toegepast voor extra filtering 8 Voor het debuggen copy paste ik de error melding in de LLM en volg ik zijn advies op. Met het maken van code geef ik een beschrijving van wat ik wil. Ik geef nooit bestaande code aan de LLM 9 We use LLM to debug code by providing the code snippet and the issue description asking to find the error. Similarly we use it to refactoring by sharing the code and stating our requirements. Explaining code is mainly when a developer picks up a code that was originally developed by someone else. Documentation is self explanatory, we just tell what are the key sections to be included in the documentation 10 Basically used LLM to help me in my current task to improve code 11 "Ik zal een antwoord geven op elk van de volgende taken: 1. Wat betreft debugging, zijn LLMs vooral nuttig bij het lezen van extreem lange debuglogs en het filteren van relevante gegevens. 2. Wat betreft refactoring, worden LLMs soms geraadpleegd voor een efficiëntere aanpak, hoewel ze meestal strategieën voorstellen die minder efficiënt zijn. Echter, LLMs kunnen nuttig zijn bij het opsporen van overbodige delen van de code. 3. LLMs zijn bijzonder behulpzaam bij documentatie, aangezien ze de codebase kunnen scannen, afhankelijkheden kunnen identificeren en in korte tijd een goed gestructureerde documentatie kunnen creëren. 4. Wat betreft prestatieverbetering, zal het antwoord vergelijkbaar zijn met code-refactoring. 5. Wanneer code niet vanzelfsprekend is, kan een LLM inzicht bieden, omdat het de gehele codebase kan scannen en een breder perspectief kan bieden." 12 I am asking to refactor my code or debug the code 13 LLM's zijn bij ons onderdeel van onze codebase en geven suggesties wanneer we code invoeren. 14 n.v.t. 15 "Analyseren van storingsmeldingen, automatisch interpreteren en classificeren van tekst in storingsrapporten of tickets" 16 n.v.t. 17 We gebruiken LLM's als ondersteuning bij het schrijven van code. Daarnaast haal ik mijn code soms door een LLM om deze te optimaliseren. Na de optimalisatie gaat er nog wel een controle over de code om er zeker van te zijn dat deze correct is. We vertrouwen nooit blind op de LLM.

Daarnaast gebruik ik met enige regelmaat een LLM (chatGPT/Claude) om reeds geschreven code uit te leggen. 18 Ik plak een algemene uitleg over wat de functie doet en vraag om documentatie. Ik vraag ook om begeleiding bij het implementeren van bepaalde functies die complexer zijn 19 "Zie vorige antwoord. Ook plak ik wel eens code in ChatGPT en vraag ik wat deze code doet." 20 Bij code waar variabelen niet duidelijk zijn kunnen llms dit beter uitleggen 21 "- Debuggen: met name voor known issues met externe libraries. - Refactoren: code completion, suggesties, feedback - Documenteren: voorzet schrijven volgens richtlijnen - Performance: code completion, suggesties, feedback - Uitleggen: vooral voor het vinden van relevante code of gebruiksaanwijzingen voor libraries (niet business logic)" 22 De LLM wordt om suggesties gevraagd. 23 Wij maken (PLC) software voor machine besturingen. Onderhoud is hier niet van toepassing. Deze controllers draaien niet op Windows of vergelijkbare OS's. Als het juist functioneert, kan een machine hier 20 jaar op draaien. Remote access geschiedt via daarvoor geschikte Remote Access Routers (bijv. eWon of IXON). Update daarvan wordt (automatisch) verzorgd door de leverancier. 24 Ik gebruik hier zelf Gemini Pro voor. Mijn 2 collegas gebruiken hier de Github Co-pilot extensie van VS Code voor

ChatGPT zei:

Hieronder zijn de antwoorden op de vraag "Op welke wijze maakt u precies gebruik van LLM's bij deze taken?" gestandaardiseerd, opgeschoond, vertaald en geherformuleerd in korte, consistente termen. Vage omschrijvingen zijn verduidelijkt, irrelevante bijwoorden verwijderd en Engelstalige input vertaald.

1. Refactoring: vragen om schonere of efficiëntere code. Debugging: foutmeldingen uitleggen en oorzaken achterhalen. Documentatie: code laten verklaren of samenvatten.
2. Uitleg van onduidelijke functies en aanwijzen van mogelijke fouten voor gerichter zoeken.
3. LLM-integratie in IDE's geeft automatisch suggesties. Soms worden functies door meerdere LLM's gegenereerd en vergeleken voor verbeteringen.
4. Vragen om uitleg bij onduidelijke code of problemen. Daarna eventueel vragen om refactoring en toepassen bij akkoord.
5. Via Cursor AI laat men de LLM meekijken, verbeteren en helpen bij foutoplossing.
6. Niet van toepassing.
7. Na handmatige controle ingezet voor extra filtering.
8. Foutmeldingen worden ingevoerd en het advies gevolgd. Bij codegeneratie wordt alleen een beschrijving gegeven, geen bestaande code gedeeld.
9. Code en beschrijving van het probleem worden ingevoerd voor debugging en refactoring. Voor uitleg van andermans code en documentatie worden instructies meegegeven.
10. LLM wordt gebruikt om code tijdens lopende taken te verbeteren.
11. Debugging: filteren van relevante informatie uit lange logs. Refactoring: opsporen van overbodige code. Documentatie: scannen van codebase en genereren van structuur. Prestatie: vergelijkbaar met refactoring. Uitleg: bredere context bieden bij onduidelijke code.

12. Refactoring en debugging op basis van codeinvoer.
13. LLM geeft automatische suggesties bij code-invoer.
14. Niet van toepassing.
15. Automatische interpretatie en classificatie van tekst in storingsrapporten en tickets.
16. Niet van toepassing.
17. Ondersteuning bij schrijven en optimaliseren van code. Extra controle na optimalisatie. Uitleg van bestaande code.
18. Uitleg over functies geven en vragen om documentatie of begeleiding bij implementatie.
19. Code plakken en vragen wat deze doet.
20. Onduidelijke variabelen en code worden toegelicht.
21. Debugging bij bekende issues met externe libraries. Refactoring en performance via suggesties en feedback. Documentatie wordt voorgesteld volgens richtlijnen. Uitleg gericht op bibliotheken.
22. LLM wordt gevraagd om suggesties.
23. Niet van toepassing.
24. Persoonlijk gebruik van Gemini Pro; collega's gebruiken GitHub Copilot in VS Code.

Je zei:

Standaardiseer de volgende antwoorden van een enquête over LLM-gebruik bij onderhoudstaken. We zijn in deze enquête geïnteresseerd in hoe gebruikers LLM's inzetten bij hun taken en wat hierbij de redenen zijn om ze wel of niet in te zetten. Gebruik korte consistente termen en verwijder irrelevante bijwoorden en herformuleer vage beschrijven duidelijk. Vertaal Engelse beschrijven naar het Nederlands. Deze antwoorden worden later weer gebruikt als input voor een analyse. De vraag die gesteld is luid: "Wat vindt u de grootste nadelen of risico's van het gebruik van LLM's bij onderhoudstaken?" Kun je de gestandaardiseerde antwoorden zonder opsommingstekens teruggeven (1, 2, 3) en aangeven wanneer niet van toepassing. 1 Grootste risico is privacy - je weet niet precies waar de code naartoe gaat of hoe die wordt opgeslagen, vooral bij cloudgebaseerde LLMs. 2 Bij het doorsturen van grote lappen code dan is er snel hallucinatie gevaar. Door de grote hoeveelheid aan code zal er soms in een verkeerde hoek gezocht kunnen worden waardoor je zelf ook op een verkeerde plek kan gaan zoeken. 3 Het is (te) eenvoudig om op het resultaat van een LLM af te gaan zonder nauwkeurige controle. Het grootste risico is dat de developers hun eigen code niet meer begrijpen en er dus ook niet over kunnen redeneren. 4 LLM is in mijn mening niet de meest veilige manier om te ontwikkelen, vooral authenticatie en permission based tasks laakt het nogal. 5 Introduceren van onopvallende fouten die je als mens niet snel zou maken. Het over het hoofd zien van belangrijke zaken of instinkers. De data veiligheid is niet duidelijk gewaarborgd. 6 theft of intellectual property (source code) 7 Grootste nadelen zijn, iets onterecht als fout aanmerken of fouten vergeten 8 Alles wat je erin zet is nu deel van de ai. Oppassen met wat je deelt 9 One is that due to lack of complete business context sometimes they produce flawed or completely incorrect output. Also it prevents lack and loss of knowledge in the team as more and more tasks are handled by LLM 10 May guide you in wrong direction while debugging, output of result

maybe wrong 11 Het gedeeltelijk openbaar maken van de codebase, aangezien je je code deelt met de LLM, die mogelijk wordt getraind en later kan worden gebruikt om het systeem te hacken. 12 Sometimes hallucinas and provide wrong code 13 LLM's snappen natuurlijk niet altijd de bedoeling. 14 afhankelijkheid van goede invoer van data 15 Te veel vertrouwen op AI leidt soms tot verlies van praktische kennis en kritisch denkvermogen bij mijn medewerkers 16 Privacy 17 Ga er niet blind van uit dat alles wat de LLM aanpast ook correct is. 18 het grootste risico is waarschijnlijk de vertrouwelijkheid van informatie, ik heb geen garanties dat de informatie die ik deel veilig is 19 "ChatGPT snapt niet altijd wat ik bedoel. Verder weet je niet wat er gebeurt met de informatie die je aan ChatGPT geeft." 20 Als het verkeerd is dan kan het je meer tijd kosten 21 "Indien niet kritisch geëvalueerd: tijd verspild aan verkeerde zaken / richting, onnodig lange of complexe code, vulnerabilities. Juniors laten zich te snel leiden of kunnen minder kritisch controleren. Kloof productiviteit junior / senior (die effectief is met AI) groeit hierdoor en kennis spreidt minder. " 22 Het blijkt lastig om de LLM te blijven wantrouwen. De voorstellen zien er op het eerste gezicht goed uit. 23 Ik kan dit niet beoordelen. We passen dit nu minimaal toe. 24 "Hallucinaties van de AI zijn voor mij wel echt een dooddoener, wanneer je werkt aan een project waar eigenlijk op het internet ook niet veel van te vinden is. Zodra dit gebeurd zie ik bijna elke LLM op het moment zelf zaken verzinnen die er niet zijn. In de toekomst zal dit wellicht beter zijn maar momenteel vind ik ook niet dat een LLM veel helpt bij een monolith codebase. Er is teveel context wat hij dan mist tenzij je heel specifiek zelf een LLM traint op puur je eigen omgevingen. Persoonlijk zou ik een LLM niet daadwerkelijke code laten schrijven of refactoren. Dit is een process waar je zelf goed bij na moet denken "

ChatGPT zei:

Hieronder vind je de gestandaardiseerde en opgeschoonde antwoorden op de vraag: "Wat vindt u de grootste nadelen of risico's van het gebruik van LLM's bij onderhoudstaken?" Vage formuleringen zijn verduidelijkt, irrelevante bijwoorden verwijderd en Engelstalige antwoorden vertaald.

1. Onzekerheid over privacy en opslag van code bij cloudgebaseerde LLM's.
2. Risico op hallucinaties bij grote hoeveelheden code; leidt tot foutieve conclusies en zoekrichtingen.
3. Te veel vertrouwen op output zonder controle; risico dat developers hun eigen code niet meer begrijpen.
4. Onveiligheid bij taken rond authenticatie en rechtenbeheer.
5. Kans op subtiele fouten en het missen van belangrijke details; onduidelijkheid over dataveiligheid.
6. Diefstal van intellectueel eigendom (broncode).
7. Fouten onterecht markeren of bestaande fouten over het hoofd zien.
8. Alles wat gedeeld wordt, kan deel worden van de AI; risico op datalekken.
9. Output kan fout zijn door gebrek aan zakelijke context; kennisverlies in het team door overmatige afhankelijkheid.
10. Foutieve aansturing tijdens debugging; onbetrouwbare resultaten.

11. Code wordt gedeeltelijk openbaar via LLM; risico op misbruik of hergebruik voor aanvallen.
12. Hallucinaties en foutieve code als resultaat.
13. LLM begrijpt niet altijd de bedoeling van de gebruiker.
14. Sterke afhankelijkheid van juiste invoer.
15. Verlies van praktische kennis en kritisch denkvermogen bij medewerkers door overmatig gebruik.
16. Privacyrisico.
17. Veranderingen van de LLM zijn niet per definitie correct; altijd controleren.
18. Geen garantie op vertrouwelijkheid van gedeelde informatie.
19. LLM begrijpt de intentie niet altijd; onduidelijk wat er met gedeelde info gebeurt.
20. Foutieve output kan leiden tot tijdverlies.
21. Zonder kritische beoordeling: tijdverspilling, complexe code, beveiligingsproblemen. Grotere afhankelijkheid van AI vergroot kennisverschil tussen junior en senior developers.
22. Moeilijk om LLM-output blijvend kritisch te beoordelen omdat het overtuigend lijkt.
23. Niet van toepassing.
24. Hallucinaties maken LLM's onbetrouwbaar, vooral bij weinig beschikbare context (zoals monolithische codebases). LLM's missen projectkennis zonder specifieke training. Persoonlijk geen vertrouwen in automatisch refactoreren of codegeneratie.

Je zei:

Standaardiseer de volgende antwoorden van een enquête over LLM-gebruik bij onderhoudstaken. We zijn in deze enquête geïnteresseerd in hoe gebruikers LLM's inzetten bij hun taken en wat hierbij de redenen zijn om ze wel of niet in te zetten. Gebruik korte consistente termen en verwijder irrelevante bijwoorden en herformuleer vage beschrijvingen duidelijk. Vertaal Engelse beschrijvingen naar het Nederlands. Deze antwoorden worden later weer gebruikt als input voor een analyse. De vraag die gesteld is luid: "Wat vindt u de grootste voordelen van het gebruik van LLM's bij onderhoudstaken?" Kun je de gestandaardiseerde antwoorden zonder opsommingstekens teruggeven (1, 2, 3) en aangeven wanneer niet van toepassing. 1 LLMs zien patronen of fouten die mensen makkelijk over het hoofd zien - dat bespaart tijd en verhoogt de kwaliteit van onderhoud. 2 Het tempo, wanneer het op de juiste manier wordt ingezet kan het zo veel tijd schelen. Dagen werk kan soms in enkele minuten uitgezocht worden. 3 Mits op de juiste manier ingezet kan het de snelheid van het development proces enorm verbeteren 4 Snelheid. Waar je vroeger Google moest gebruiken of uren op Stackoverflow moest doorspitten naar een antwoord is LLM heel rap en duidelijk. Ook kun je doorvragen nadat je een antwoord hebt gekregen. 5 Enorme tijdsbesparing. 6 Easy and fast learning. When in doubt just use LLM instead of checking hundreds of github 7 Snel en efficiënt toepassen van llmks 8 Het is snel. Je kan het meeste ook vinden via google en stackoverflow, maar een LLM is vele malen sneller met een oplossing 9 It helps us become more efficient. We can focus our time on other more important things.

Routine tasks that are less value adding like documentation can be delegated to LLM 10 Help me to coding faster 11 Het elimineren van handmatige en repetitieve taken. 12 Guide me to write my code faster 13 Het scheelt tijd om code op te zoeken als een LLM wel de juiste syntax geeft. 14 automatisering van documentatie 15 Ze kunnen helpen bij het vastleggen van technologische kennis en ervaringen van medewerkers, zodat nieuwe medewerkers sneller toegang hebben tot relevante informatie. 16 Efficiency 17 Tijdsbesparing 18 bespaart je veel tijd en creëert documenten van hoge kwaliteit die je meteen kunt gebruiken 19 "Het heeft de potentie om je werk te versnellen. Je maakt minder fouten." 20 Kan als een soort 2nd opinion werken 21 "Indien correct toegepast: meer kwaliteit in minder tijd. Tijdsbesparing door: code generation, minder in documentatie hoeven kijken. Kwaliteit door: bredere kennis en meer durf om een idee het LLM in te gooien en te kijken hoe het uitpakt (i.p.v. zelf schrijven met kostbare tijd)." 22 De LLM heeft meer kennis dan 1 persoon, waardoor de kwaliteit omhoog kan gaan. 23 Ik kan dit niet beoordelen. We passen dit nu minimaal toe. 24 Vooral bij het debuggen en uitleggen van een stuk code kan het heel handig zijn om hier uitgebreid een uitleg van te krijgen wat het dan is. Voor debuggen valt er nog wat voor te zeggen als het om vrij edge case zaken gaat dat een LLM dan weer niet handig is maar bij general errors kan je er wel wat mee.

ChatGPT zei:

Hieronder zijn de antwoorden op de vraag “Wat vindt u de grootste voordelen van het gebruik van LLM's bij onderhoudstaken?” gestandaardiseerd, vereenvoudigd en vertaald waar nodig. De formuleringen zijn kort, duidelijk en consistent gemaakt voor latere analyse.

1. LLM's herkennen patronen en fouten die mensen missen, wat tijd bespaart en de kwaliteit verhoogt.
2. Grote tijdsbesparing bij juist gebruik; dagen werk kan in minuten worden gedaan.
3. Verhoogt de snelheid van het ontwikkelproces aanzienlijk.
4. Snelle antwoorden zonder langdurig zoeken; mogelijkheid tot doorvragen.
5. Enorme tijdsbesparing.
6. Snelle en eenvoudige kennisopbouw; geen lange zoektochten op GitHub nodig.
7. Snelle en efficiënte inzet van LLM's.
8. Sneller dan zoeken via Google of Stack Overflow.
9. Verhoogt efficiëntie; routinetaken zoals documentatie kunnen worden gedelegeerd.
10. Versnelt het schrijven van code.
11. Vermindert handmatig en repetitief werk.
12. Helpt sneller code te schrijven.
13. Bespaart tijd door direct correcte syntax te geven.
14. Automatisering van documentatie.
15. Legt technologische kennis vast; ondersteunt onboarding van nieuwe medewerkers.
16. Efficiëntie.

17. Tijdsbesparing.
18. Bespaart tijd en levert direct bruikbare documentatie van hoge kwaliteit.
19. Versnelt werk en vermindert fouten.
20. Werkt als second opinion.
21. Meer kwaliteit in minder tijd door codegeneratie, minder zoekwerk en bredere kennis.
22. Meer kennis dan één persoon; kan de kwaliteit verhogen.
23. Niet van toepassing.
24. Handig bij uitleg en debugging van algemene fouten; minder geschikt voor edge cases.

Je zei:

Standaardiseer de volgende antwoorden van een enquête over LLM-gebruik bij onderhoudstaken. We zijn in deze enquête geïnteresseerd in hoe gebruikers LLM's inzetten bij hun taken en wat hierbij de redenen zijn om ze wel of niet in te zetten. Gebruik korte consistente termen en verwijder irrelevante bijwoorden en herformuleer vage beschrijvingen duidelijk. Vertaal Engelse beschrijvingen naar het Nederlands. Deze antwoorden worden later weer gebruikt als input voor een analyse. De vraag die gesteld is luid: "(Optioneel) Zijn er nog andere onderhoudstaken waarvoor LLM's gebruikt kunnen worden die we niet mee hebben genomen in deze vragenlijst?" Kun je de gestandaardiseerde antwoorden zonder opsommingstekens teruggeven (1, 2, 3) en aangeven wanneer niet van toepassing. 1 LLMs kunnen ook helpen bij het brainstormen over hoe je bepaalde code het best kunt opzetten, nog voordat je begint met schrijven. 2 Nvt 3 Nvt 4 5 Het valideren van code volgens standaarden of andere eisen. Het creëren van nieuwe programma's die er überhaupt nog niet waren, vooral handig voor code die niet meer te onderhouden valt vanwege verouderde afhankelijkheden. 6 7 8 It-ers zijn wellicht niet de meest sociale mensen. Ik gebruik het soms om een nette email voor me te maken aan de hand van wat steekwoorden 9 10 11 Testen 12 13 LLM's zijn altijd nuttig om te raadplegen omdat ze vaak invalshoeken bieden waar je zelf niet altijd als eerst aan gedacht hebt. 14 15 We maken ook gebruik van LLM's voor de prefectieve onderhoud van onze systemen 16 17 18 19 20 Unit tests 21 "Alles raakt wel, maar de vorm/workflow van onderhoud zie ik ook veranderen door AI. Dus: VCS (branching, commit messages), issue tracking, code review. In theorie kan AI sommige kleine issues en RFCs automatisch van een PR voorzien. En PRs reviewen." 22 Het bijwerken van code naar aanleiding van updates in libraries 23 Ervaring ontbreekt om hier een gefundeerd antwoord op te kunnen geven. Wel aan het onderzoeken wat de mogelijkheden zijn. 24

ChatGPT zei:

Hieronder vind je de gestandaardiseerde antwoorden op de optionele vraag: "Zijn er nog andere onderhoudstaken waarvoor LLM's gebruikt kunnen worden die we niet mee hebben genomen in deze vragenlijst?" Irrelevante delen zijn verwijderd, vage beschrijvingen verduidelijkt en de stijl is geharmoniseerd voor latere analyse.

1. Brainstormen over de beste aanpak van code vóór het schrijven.
2. Niet van toepassing.
3. Niet van toepassing.

4. Niet van toepassing.
5. Valideren van code op standaarden of vereisten, herschrijven van niet-onderhoudbare legacycode.
6. Niet van toepassing.
7. Niet van toepassing.
8. Opstellen van nette e-mails op basis van steekwoorden.
9. Niet van toepassing.
10. Niet van toepassing.
11. Ondersteuning bij testen.
12. Niet van toepassing.
13. Bieden van alternatieve invalshoeken die men zelf niet meteen ziet.
14. Niet van toepassing.
15. Ondersteuning bij preventief onderhoud.
16. Niet van toepassing.
17. Niet van toepassing.
18. Niet van toepassing.
19. Niet van toepassing.
20. Genereren van unit tests.
21. Ondersteuning bij VCS-workflows, zoals branching, commit messages, issue tracking en code reviews. Kleine issues of RFC's kunnen automatisch worden verwerkt tot pull requests.
22. Bijwerken van code bij updates in libraries.
23. Niet van toepassing; nog in onderzoek.
24. Niet van toepassing.

Je zij:

Kun je een thematische analyse doen over de volgende antwoorden. Deze antwoorden zijn gegeven in een enquête die gaat over het achterhalen hoe software engineers LLM tools inzetten bij hun werk. Hierbij wordt er in dit onderzoek verder verdiept in hoe ze dit doen bij het onderhouden van software. De vragen die we in deze analyse behandelen zijn: “Bij welke onderhoudstaken gebruikt u al LLM's en kunt u daar in het algemeen kort iets over vertellen?” en “Op welke wijze maakt u precies gebruik van LLM's bij deze taken?” de antwoorden hiervan zijn aan elkaar geplakt omdat het beide over taken gaat. Tussen deze 2 vragen is een likert gesteld waarbij er per taak (bvb refactoreren, debuggen documenteren van code) is gevraagd hoe vaak ze gebruik maken van LLM's bij deze taken om de respondent al een paar voorbeeldjes te geven. De thema's die worden gekozen moeten dus vooral toepassingen van LLM's en taken zijn die

een software engineer moet doen bij het onderhouden van software. Kun je bij elk thema ook aangeven het ID van de antwoorden waarin dit thema voorkomt. 1 Refactoring, schrijven van automatiseringsscripts en documentatie. LLM's helpen repetitieve taken versnellen en bieden snelle oplossingsuggesties. Refactoring: vragen om schonere of efficiëntere code. Debugging: foutmeldingen uitleggen en oorzaken achterhalen. Documentatie: code laten verklaren of samenvatten.

2 Migraties, genereren van JSON-bestanden, efficiëntiecontroles. Inzet bij het schrijven van eenvoudige maar omvangrijke code. Uitleg van onduidelijke functies en aanwijzen van mogelijke fouten voor gericht zoeken.

3 LLM's worden gebruikt voor boilerplate code en code-analyse, maar code-analyse wordt niet als onderhoud gezien. LLM-integratie in IDE's geeft automatisch suggesties. Soms worden functies door meerdere LLM's gegenereerd en vergeleken voor verbeteringen.

4 Ondersteuning bij programmeren, bugfixing en het schrijven van tests. Vragen om uitleg bij onduidelijke code of problemen. Daarna eventueel vragen om refactoring en toepassen bij akkoord.

5 Gebruikt voor update-commando's in softwareprojecten, zoals met package managers (bijv. npm). Via Cursor AI laat men de LLM meekijken, verbeteren en helpen bij foutoplossing.

6 Niet van toepassing. Niet van toepassing.

7 Gegevensopruiming en bugdetectie. Na handmatige controle ingezet voor extra filtering.

8 Schrijven van code (Python, SQL), genereren van stappenplannen, advies over implementatiekeuzes (bijv. meldingen bij fouten). Foutmeldingen worden ingevoerd en het advies gevolgd. Bij codegeneratie wordt alleen een beschrijving gegeven, geen bestaande code gedeeld.

9 Refactoring, genereren van tests, eenvoudige code-updates via natuurlijke taal. Code en beschrijving van het probleem worden ingevoerd voor debugging en refactoring. Voor uitleg van andermans code en documentatie worden instructies meegegeven.

10 Refactoring en documentatie. LLM wordt gebruikt om code tijdens lopende taken te verbeteren.

11 Gegevensgeneratie en -opvraging, brainstormen bij complexe problemen en het verkennen van alternatieven. Debugging: filteren van relevante informatie uit lange logs. Refactoring: opsporen van overbodige code. Documentatie: scannen van codebase en genereren van structuur. Prestatie: vergelijkbaar met refactoring. Uitleg: bredere context bieden bij onduidelijke code.

12 Debuggen en schrijven van documentatie. Refactoring en debugging op basis van codeinvoer.

13 Niet van toepassing. LLM geeft automatische suggesties bij code-invoer.

- 14 Niet van toepassing. Niet van toepassing.
- 15 Ondersteuning bij storingsdiagnose, oorzaakanalyse op basis van symptomen en data, genereren en bijwerken van documentatie en handleidingen. Automatische interpretatie en classificatie van tekst in storingsrapporten en tickets.
- 16 Niet van toepassing. Niet van toepassing.
- 17 Niet van toepassing. Ondersteuning bij schrijven en optimaliseren van code. Extra controle na optimalisatie. Uitleg van bestaande code.
- 18 Schrijven van documentatie en verduidelijken van moeilijke concepten. Uitleg over functies geven en vragen om documentatie of begeleiding bij implementatie.
- 19 Genereren en refactoren van code op basis van beschrijvingen. Code plakken en vragen wat deze doet.
- 20 Schrijven van unit tests en commitberichten. Onduidelijke variabelen en code worden toegelicht.
- 21 Vinden van relevante codeonderdelen en uitvoeren van kleine fixes met Claude Code. Sparren over problemen en kleine projecten ondersteunen via Repomix. Debugging bij bekende issues met externe libraries. Refactoring en performance via suggesties en feedback. Documentatie wordt voorgesteld volgens richtlijnen. Uitleg gericht op bibliotheken.
- 22 Nog geen inzet bij onderhoudstaken. LLM wordt gevraagd om suggesties.
- 23 Niet van toepassing. Niet van toepassing.
- 24 Niet van toepassing. Persoonlijk gebruik van Gemini Pro; collega's gebruiken GitHub Copilot in VS Code.

ChatGPT zei:

Hieronder volgt een thematische analyse van de gegeven antwoorden op basis van de gecombineerde vragen "Bij welke onderhoudstaken gebruikt u al LLM's?" en "Op welke wijze maakt u precies gebruik van LLM's bij deze taken?" De thema's zijn afgeleid uit terugkerende gebruikstoepassingen van LLM's binnen softwareonderhoud en worden gekoppeld aan de ID's van respondenten die deze thema's noemden.

1. Refactoring van code

Beschrijving: LLM's worden gebruikt om bestaande code te herschrijven voor leesbaarheid, efficiëntie of eenvoud, vaak op verzoek van de developer.

Respondent-ID's: 1, 4, 9, 10, 11, 12, 19, 21

2. Debugging en foutanalyse

Beschrijving: LLM's worden ingezet om foutmeldingen te interpreteren, logs te analyseren of oorzaken van bugs te achterhalen.

Respondent-ID's: 1, 4, 5, 8, 9, 11, 12, 15, 20, 21

3. Genereren en bijwerken van documentatie

Beschrijving: LLM's worden gebruikt om technische documentatie te maken of bij te werken op basis van code of uitleg.

Respondent-ID's: 1, 9, 10, 11, 12, 15, 18, 21

4. Uitleg en verduidelijking van code

Beschrijving: LLM's worden geraadpleegd om de werking van bestaande code of concepten te verduidelijken, vooral bij onbekende of complexe code.

Respondent-ID's: 2, 4, 8, 9, 11, 17, 18, 19, 20, 21

5. Testgeneratie (unit/integration)

Beschrijving: LLM's worden ingezet voor het genereren van tests, zoals unit tests of edge-case tests.

Respondent-ID's: 4, 9, 20, 21

6. Codegeneratie op basis van beschrijvingen

Beschrijving: LLM's worden gevraagd om nieuwe code te genereren op basis van een natuurlijke taalbeschrijving of specificatie.

Respondent-ID's: 2, 8, 9, 19

7. Prestatieverbetering en optimalisatie

Beschrijving: LLM's geven suggesties om code sneller of efficiënter te maken, vaak in combinatie met refactoring.

Respondent-ID's: 1, 11, 17, 21

8. Analyse en filtering van storingsmeldingen of logs

Beschrijving: LLM's worden gebruikt voor het analyseren van storingsmeldingen, debuglogs en foutrapportages.

Respondent-ID's: 11, 15

9. Ondersteuning bij updates en migraties

Beschrijving: LLM's ondersteunen bij het updaten van dependencies, libraries of scripts, en bij migratietaken.

Respondent-ID's: 2, 5, 22

10. Takenrationalisatie en werkverdeling (denkpartner)

Beschrijving: LLM's worden gebruikt om meerdere oplossingen te vergelijken of als sparringpartner bij het ontwerpen van codeoplossingen.

Respondent-ID's: 3, 11, 13, 21

11. Onderhoudsondersteuning buiten directe code (zoals commit messages of communicatie)

Beschrijving: LLM's worden gebruikt voor commitberichten, technische e-mails of communicatie rond onderhoud.

Respondent-ID's: 20, 8

12. Overige of algemene optimalisatie en assistentie

Beschrijving: Algemene hulp of versnelling bij onderhoudsgerelateerde ontwikkeltaken, zonder specifieke categorisatie.

Respondent-ID's: 6, 16, 24 (vaag of niet specifiek benoemd)

13. Geen gebruik / niet van toepassing

Beschrijving: De respondent geeft aan (nog) geen gebruik te maken van LLM's voor onderhoud.

Respondent-ID's: 6, 13, 14, 16, 22, 23, 24

Wil je dat ik deze thematische analyse ook omzet naar een tabelvorm voor overzichtelijkheid of verder verwerk in een verslagvorm?

Je zei:

Kun je een thematische analyse doen over de volgende antwoorden. Deze antwoorden komen van een enquête die gaat over het achterhalen hoe software engineers LLM tools inzetten bij hun werk en wat de redenen daarbij zijn om wel of niet een LLM in te zetten. Bij deze analyse gaat het om antwoorden van de vraag "Wat vindt u de grootste nadelen of risico's van het gebruik van LLM's bij onderhoudstaken?". We willen dus weten wat de grootste nadelen of risico's zijn. Kun je bij elk thema ook aangeven het ID van de antwoorden waarin dit thema voorkomt. 1

Onzekerheid over privacy en opslag van code bij cloudgebaseerde LLM's.

2 Risico op hallucinaties bij grote hoeveelheden code; leidt tot foutieve conclusies en zoekrichtingen.

3 Te veel vertrouwen op output zonder controle; risico dat developers hun eigen code niet meer begrijpen.

4 Onveiligheid bij taken rond authenticatie en rechtenbeheer.

5 Kans op subtiele fouten en het missen van belangrijke details; onduidelijkheid over dataveiligheid.

6 Diefstal van intellectueel eigendom (broncode).

7 Fouten onterecht markeren of bestaande fouten over het hoofd zien.

- 8 Alles wat gedeeld wordt, kan deel worden van de AI; risico op datalekken.
- 9 Output kan fout zijn door gebrek aan zakelijke context; kennisverlies in het team door overmatige afhankelijkheid.
- 10 Foutieve aansturing tijdens debugging; onbetrouwbare resultaten.
- 11 Code wordt gedeeltelijk openbaar via LLM; risico op misbruik of hergebruik voor aanvallen.
- 12 Hallucinaties en foutieve code als resultaat.
- 13 LLM begrijpt niet altijd de bedoeling van de gebruiker.
- 14 Sterke afhankelijkheid van juiste invoer.
- 15 Verlies van praktische kennis en kritisch denkvermogen bij medewerkers door overmatig gebruik.
- 16 Privacyrisico.
- 17 Veranderingen van de LLM zijn niet per definitie correct; altijd controleren.
- 18 Geen garantie op vertrouwelijkheid van gedeelde informatie.
- 19 LLM begrijpt de intentie niet altijd; onduidelijk wat er met gedeelde info gebeurt.
- 20 Foutieve output kan leiden tot tijdverlies.
- 21 Zonder kritische beoordeling: tijdverspilling, complexe code, beveiligingsproblemen. Grotere afhankelijkheid van AI vergroot kennisverschil tussen junior en senior developers.
- 22 Moeilijk om LLM-output blijvend kritisch te beoordelen omdat het overtuigend lijkt.
- 23 Niet van toepassing.
- 24 Hallucinaties maken LLM's onbetrouwbaar, vooral bij weinig beschikbare context (zoals monolithische codebases). LLM's missen projectkennis zonder specifieke training. Persoonlijk geen vertrouwen in automatisch refactoreren of codegeneratie.

ChatGPT zei:

Hieronder volgt een thematische analyse van de antwoorden op de vraag:

“Wat vindt u de grootste nadelen of risico's van het gebruik van LLM's bij onderhoudstaken?”

De thema's zijn gebaseerd op herhaalde motieven in de antwoorden, gericht op risico's en nadelen van LLM-gebruik binnen softwareonderhoud. Per thema geef ik ook de respondent-ID's weer.

1. Privacy en vertrouwelijkheid van gegevens

Beschrijving: Zorg over waar de gedeelde code/informatie terechtkomt, hoe deze wordt opgeslagen en of deze kan worden misbruikt.

Respondent-ID's: 1, 5, 8, 11, 16, 18, 19

2. Hallucinaties en foutieve output

Beschrijving: LLM's geven onjuiste, verzonnen of misleidende antwoorden, vooral bij grote of complexe codebases.

Respondent-ID's: 2, 12, 24

3. Te veel vertrouwen zonder kritische controle

Beschrijving: Gebruikers nemen output te snel over zonder te controleren, wat leidt tot fouten of verlies van kritisch denkvermogen.

Respondent-ID's: 3, 15, 17, 21, 22

4. Beperkte contextkennis van de LLM

Beschrijving: LLM's missen domein- of projectcontext, wat kan leiden tot foutieve suggesties of verkeerde aannames.

Respondent-ID's: 9, 13, 19, 24

5. Afhankelijkheid en kennisverlies in teams

Beschrijving: Overmatig gebruik van LLM's kan leiden tot afname van kennisopbouw, begrip en vaardigheden bij developers.

Respondent-ID's: 3, 9, 15, 21

6. Beveiligingsrisico's en misbruik

Beschrijving: LLM-gebruik kan leiden tot beveiligingslekken, vooral bij authenticatie, rechtenbeheer of hergebruik van code.

Respondent-ID's: 4, 5, 6, 11, 21

7. Foutgevoeligheid en onnauwkeurigheid

Beschrijving: LLM's kunnen fouten introduceren of bestaande fouten over het hoofd zien.

Respondent-ID's: 5, 7, 10, 20

8. Afhankelijkheid van juiste prompt/input

Beschrijving: LLM-output is sterk afhankelijk van goede invoer; bij onduidelijke prompts is de kans op fouten groter.

Respondent-ID's: 14

9. Ongeschiktheid voor specifieke taken of contexten

Beschrijving: LLM's zijn minder geschikt voor taken met weinig publieke informatie of complexe legacy-omgevingen.

Respondent-ID's: 24

10. Niet van toepassing / geen antwoord

Beschrijving: De respondent heeft geen (relevant) antwoord gegeven op de vraag.

Respondent-ID's: 23

Wil je dat ik deze analyse omzet naar een overzichtelijke tabel of verwerk in een format geschikt voor rapportage?

Je zei:

Kun je een thematische analyse doen over de volgende antwoorden. Deze antwoorden komen van een enquête die gaat over het achterhalen hoe software engineers LLM tools inzetten bij hun werk en wat de redenen daarbij zijn om wel of niet een LLM in te zetten. Bij deze analyse gaat het om antwoorden van de vraag "Wat vindt u de grootste voordelen van het gebruik van LLM's bij onderhoudstaken?" We willen dus weten wat de voordelen zijn van het gebruik van LLM's. Kun je bij elk thema ook aangeven het ID van de antwoorden waarin dit thema voorkomt. 1 LLM's herkennen patronen en fouten die mensen missen, wat tijd bespaart en de kwaliteit verhoogt.

- 2 Grote tijdsbesparing bij juist gebruik; dagen werk kan in minuten worden gedaan.
- 3 Verhoogt de snelheid van het ontwikkelproces aanzienlijk.
- 4 Snelle antwoorden zonder langdurig zoeken; mogelijkheid tot doorvragen.
- 5 Enorme tijdsbesparing.
- 6 Snelle en eenvoudige kennisopbouw; geen lange zoektochten op GitHub nodig.
- 7 Snelle en efficiënte inzet van LLM's.
- 8 Sneller dan zoeken via Google of Stack Overflow.
- 9 Verhoogt efficiëntie; routinetaken zoals documentatie kunnen worden gedelegeerd.
- 10 Versnelt het schrijven van code.
- 11 Vermindert handmatig en repetitief werk.
- 12 Helpt sneller code te schrijven.
- 13 Bespaart tijd door direct correcte syntax te geven.
- 14 Automatisering van documentatie.
- 15 Legt technologische kennis vast; ondersteunt onboarding van nieuwe medewerkers.

- 16 Efficiëntie.
- 17 Tijdsbesparing.
- 18 Bespaart tijd en levert direct bruikbare documentatie van hoge kwaliteit.
- 19 Versnelt werk en vermindert fouten.
- 20 Werkt als second opinion.
- 21 Meer kwaliteit in minder tijd door codegeneratie, minder zoekwerk en bredere kennis.
- 22 Meer kennis dan één persoon; kan de kwaliteit verhogen.
- 23 Niet van toepassing.
- 24 Handig bij uitleg en debugging van algemene fouten; minder geschikt voor edge cases.

ChatGPT zei:

Hieronder volgt een thematische analyse van de antwoorden op de vraag:

“Wat vindt u de grootste voordelen van het gebruik van LLM's bij onderhoudstaken?”

De thema's zijn gebaseerd op terugkerende voordelen van LLM-gebruik zoals genoemd door software engineers. Per thema zijn de bijbehorende respondent-ID's vermeld.

1. Tijdsbesparing

Beschrijving: LLM's zorgen voor significante tijdwinst bij ontwikkel- en onderhoudstaken, doordat ze snel antwoorden geven of werk automatiseren.

Respondent-ID's: 1, 2, 3, 4, 5, 8, 10, 12, 13, 17, 18, 19

2. Efficiëntieverbetering

Beschrijving: LLM's maken onderhoudswerk efficiënter door minder handmatige inspanning en betere taakverdeling.

Respondent-ID's: 3, 7, 9, 11, 16

3. Automatisering van repetitieve of eenvoudige taken

Beschrijving: Taken zoals documentatie, boilerplate code, of standaardstructuren worden geautomatiseerd met LLM's.

Respondent-ID's: 9, 11, 14

4. Kwaliteitsverhoging

Beschrijving: LLM's helpen fouten verminderen, patronen herkennen, of betere alternatieven voorstellen, wat leidt tot hogere kwaliteit van code of onderhoud.

Respondent-ID's: 1, 19, 21, 22

5. Ondersteuning bij kennisoverdracht en documentatie

Beschrijving: LLM's helpen bij het vastleggen van kennis, uitleggen van code of ondersteunen van nieuwe teamleden.

Respondent-ID's: 14, 15, 18, 24

6. Sneller vinden van informatie / alternatief voor zoeken

Beschrijving: LLM's vervangen deels het zoeken op Stack Overflow of Google, en geven snellere, gerichte antwoorden.

Respondent-ID's: 4, 6, 8, 13

7. Hulp bij codegeneratie

Beschrijving: LLM's genereren code op basis van input of specificaties, wat het ontwikkelproces versnelt.

Respondent-ID's: 10, 12, 13, 2

8. Second opinion / sparringpartner

Beschrijving: LLM's worden gebruikt om suggesties of alternatieven te toetsen — als extra denkkraft naast de developer.

Respondent-ID's: 20, 21

9. Niet van toepassing

Beschrijving: Geen voordeel genoemd of geen ervaring met LLM's.

Respondent-ID's: 23

Wil je deze thematische analyse ook als tabel aangeleverd hebben, of verwerkt in een format dat je kunt opnemen in een onderzoeksverslag of presentatie?

UITLEG TOEPASSINGEN VAN LLM'S

Thema	Uitleg
Debugging en foutanalyse (10 keer)	LLM's worden gebruikt om foutmeldingen, logs of bugs te analyseren en mogelijke oorzaken te achterhalen. Ze helpen ontwikkelaars sneller tot een oplossing te komen bij problemen in bestaande code.
Uitleg en verduidelijking van code (10 keer)	LLM's worden geraadpleegd om bestaande code begrijpelijker te maken, bijvoorbeeld door uitleg te geven over de werking van functies of variabelen. Dit is vooral nuttig bij overgenomen of complexe code.
Refactoring van code (8 keer)	LLM's worden ingezet om bestaande code te herschrijven met als doel deze leesbaarder, efficiënter of onderhoudsvriendelijker te maken — zonder de functionaliteit te wijzigen.
Genereren en bijwerken van documentatie (8 keer)	LLM's ondersteunen bij het automatisch opstellen, aanvullen of actualiseren van technische documentatie op basis van code, uitleg of projectinformatie.
Geen gebruik / niet van toepassing (7 keer)	De respondent geeft aan (nog) geen gebruik te maken van LLM's voor onderhoudstaken of acht het niet relevant voor hun situatie.
Testgeneratie (unit/integration) (4 keer)	LLM's worden gebruikt om automatisch testcases te genereren, zoals unit tests of edge-case tests, op basis van bestaande code of specificaties.
Codegeneratie op basis van beschrijvingen (4 keer)	LLM's genereren complete stukken code op basis van natuurlijke taalbeschrijvingen of functionele eisen, bijvoorbeeld bij nieuwe functionaliteit of scripts.
Prestatieverbetering en optimalisatie (4 keer)	LLM's geven suggesties om code sneller of efficiënter te maken, vaak in combinatie met refactoring. Denk aan het opsporen van onnodige bewerkingen of verbeteren van algoritmes.
Takenrationalisatie en werkverdeling (denkpartner) (4 keer)	LLM's worden gebruikt als sparringpartner: ze genereren alternatieven, geven feedback of vergelijken oplossingen, waardoor ontwikkelaars hun keuzes kunnen toetsen of verbeteren.
Ondersteuning bij updates en migraties (3 keer)	LLM's ondersteunen bij het updaten van afhankelijkheden, bibliotheken of scripts en helpen bij migraties naar nieuwere versies of systemen.
Overige of algemene optimalisatie en assistentie (3 keer)	Algemene hulp bij onderhoudstaken die niet direct binnen andere thema's vallen, zoals algemene verbetering van werkwijze of ondersteuning bij uiteenlopende vragen.
Analyse en filtering van storingen of logs (2 keer)	Algemene hulp bij onderhoudstaken die niet direct binnen andere thema's vallen, zoals algemene verbetering van werkwijze of ondersteuning bij uiteenlopende vragen.
Onderhoudsondersteuning buiten directe code (zoals commit messages of communicatie) (2 keer)	Algemene hulp bij onderhoudstaken die niet direct binnen andere thema's vallen, zoals algemene verbetering van werkwijze of ondersteuning bij uiteenlopende vragen.

UITLEG NADELEN

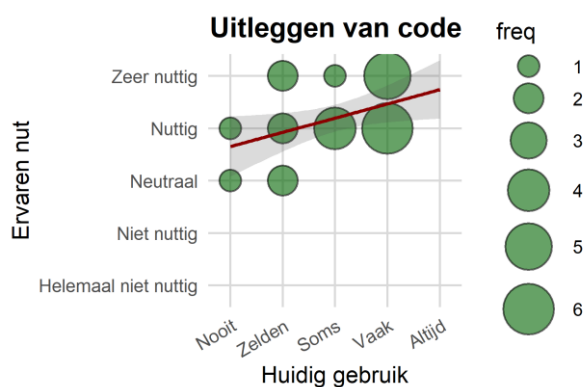
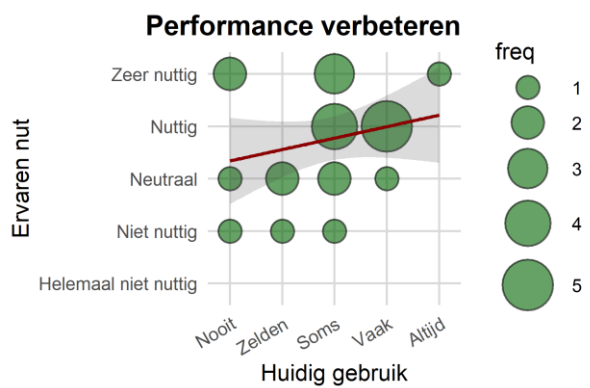
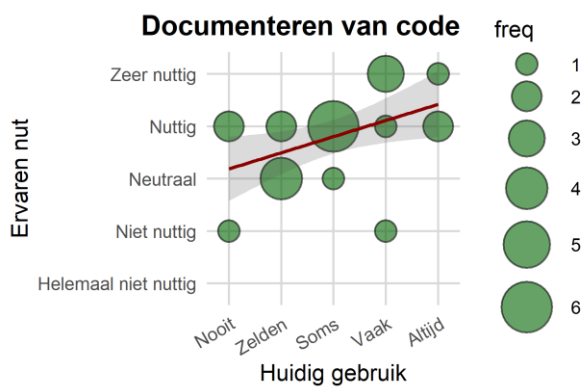
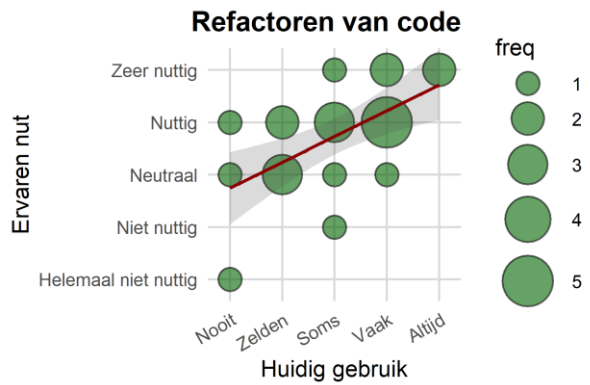
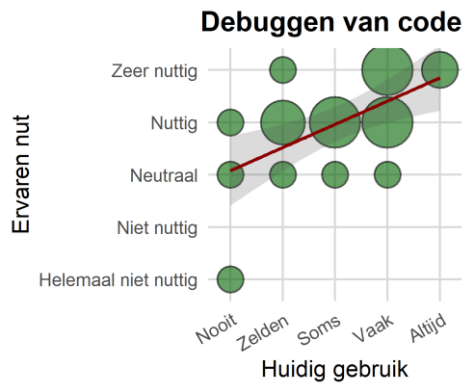
Thema	Uitleg
Privacy en vertrouwelijkheid van gegevens (7 keer)	Er is onzekerheid over wat er met gedeelde code of informatie gebeurt wanneer deze naar een LLM wordt gestuurd, vooral bij cloudgebaseerde modellen. Dit kan leiden tot datalekken of ongewenst hergebruik van gevoelige informatie.
Te veel vertrouwen zonder kritische controle (5 keer)	Gebruikers kunnen geneigd zijn om LLM-antwoorden zonder controle over te nemen. Hierdoor bestaat het risico dat fouten over het hoofd worden gezien of dat developers minder kritisch gaan denken.
Beveiligingsrisico's en misbruik (5 keer)	LLM's kunnen onveilige code genereren of onbedoeld kwetsbaarheden introduceren, bijvoorbeeld bij authenticatie of rechtenbeheer. Ook kan gedeelde code opnieuw worden gebruikt op een manier die beveiligingsrisico's oplevert.
Bepaalde contextkennis van de LLM (4 keer)	Omdat een LLM geen volledige toegang heeft tot de project- of businesscontext, kan het onjuiste of irrelevante suggesties geven. Dit maakt het minder betrouwbaar bij complexe of bedrijfsspecifieke situaties.
Afhankelijkheid en kennisverlies in teams (4 keer)	Overmatig vertrouwen op LLM's kan ertoe leiden dat developers minder zelf nadenken, kennis minder gedeeld wordt en teamleden minder vaardig worden in het zelfstandig oplossen van problemen.
Foutgevoeligheid en onnauwkeurigheid (4 keer)	LLM's kunnen verkeerde antwoorden geven, bestaande fouten niet opmerken of zelf subtiele fouten introduceren die lastig te detecteren zijn.
Hallucinaties en foutieve output (3 keer)	LLM's kunnen ogenschijnlijk overtuigende maar volledig verzonnen of foutieve informatie genereren, vooral wanneer de input vaag is of de context ontbreekt.
Afhankelijkheid van juiste prompt/input (1 keer)	De kwaliteit van de output hangt sterk af van de kwaliteit van de input. Onduidelijke of foutieve prompts leiden snel tot verkeerde resultaten.
Ongeschiktheid voor specifieke taken of contexten (1 keer)	Voor bepaalde onderhoudstaken, zoals werken met legacy-systemen of zeer domeinspecifieke projecten, zijn LLM's vaak onbetrouwbaar of missen ze benodigde kennis.
Niet van toepassing / geen antwoord (1 keer)	De respondent heeft geen relevante nadelen ervaren of geen antwoord op de vraag gegeven.

UITLEG VOORDELEN

Thema	Uitleg
Tijdsbesparing (12 keer)	LLM's kunnen snel oplossingen of suggesties geven, waardoor taken zoals debugging, documentatie of code schrijven aanzienlijk minder tijd kosten dan wanneer dit handmatig gebeurt.
Efficiëntieverbetering (5 keer)	LLM's maken onderhoudstaken efficiënter door processen te versnellen, stappen over te slaan of automatisch alternatieven te bieden. Hierdoor kunnen developers zich richten op complexere werkzaamheden.
Kwaliteitsverhoging (4 keer)	LLM's helpen fouten voorkomen, detecteren inefficiënte patronen en geven verbeterde suggesties, wat leidt tot een hogere codekwaliteit en minder technische schuld.
Ondersteuning bij kennisoverdracht en documentatie (4 keer)	LLM's helpen bij het documenteren van systemen, uitleggen van code en vastleggen van kennis, wat nuttig is voor onboarding en kennisdeling binnen teams.
Sneller vinden van informatie / alternatief voor zoeken (4 keer)	In plaats van handmatig zoeken op Stack Overflow of in documentatie, bieden LLM's direct gerichte antwoorden op ontwikkelvragen, wat het zoekproces versnelt.
Hulp bij codegeneratie (4 keer)	LLM's kunnen op basis van natuurlijke taalbeschrijvingen of specificaties automatisch bruikbare code genereren, wat vooral handig is bij standaardfunctionaliteit of boilerplate.
Automatisering van repetitieve of eenvoudige taken (3 keer)	Terugkerende of standaardtaken zoals documentatie aanmaken, eenvoudige scripts schrijven of updates doorvoeren kunnen met behulp van LLM's automatisch worden uitgevoerd.
Second opinion / sparringpartner (2 keer)	LLM's kunnen functioneren als virtuele collega: ze bieden alternatieve oplossingen, denken mee over keuzes of helpen ideeën toetsen, vooral bij twijfel of verkenning.
Niet van toepassing (1 keer)	De respondent maakt (nog) geen gebruik van LLM's voor onderhoudstaken of heeft geen voordeel benoemd.

BIJLAGE E – DIAGRAMMEN VERGELIJKING ERVAREN NUT EN TOEKOMST VERSUS HUIDIG GEBRUIK

DIAGRAMMEN ERVAREN NUT IN VERGELIJKING MET HUIDIG GEBRUIK PER TAAK



DIAGRAMMEN VERWACHT TOEKOMSTIG GEBRUIK IN VERGELIJKING MET HUIDIG GEBRUIK PER TAAK

