KI-gestützte Modernisierung von Altanwendungen: Anwendungsfelder von LLMs im Software Reengineering

Sandro Hartenstein, Andreas Schmietendorf
Hochschule für Wirtschaft und Recht Berlin
sandro.hartenstein@hwr-berlin.de, andreas.schmietendorf@hwr-berlin.de

1. Motivation

Die Integration von Large Language Models, kurz LLMs, in den Modernisierungsprozess von Altanwendungen bietet nicht nur eine Vielzahl technologischer Möglichkeiten, sondern dient auch als starke Motivation für Unternehmen, ihre bestehenden Systeme zu verbessern. LLMs repräsentieren einen bedeutsamen Fortschritt in der künstlichen Intelligenz (KI), veraltete Anwendungen können mit, aber auch durch LLMs aufgewertet werden. Durch die Nutzung von LLMs können Unternehmen nicht nur ihre Effizienz steigern, sondern auch ihre Wettbewerbsfähigkeit auf dem Markt stärken. Diese Ausarbeitung adressiert die folgenden Fragen zur Implementierung von LLMs im Modernisierungsprozess:

FF1 Wie können LLMs die Modernisierung von Altanwendungen im Anforderungsmanagement unterstützen?

FF2 Inwiefern ermöglichen LLMs effiziente Software Reengineering Prozesse?

2. LLMs im Software Reengineering

LLMs wie OpenAI's ChatGPT bieten vielfältige Möglichkeiten im Software Reengineering. Neben der Klassifizierung des Aufwands von Anforderungen können sie auch zur automatischen Dokumentationserstellung, natürlichsprachlichen Anforderungsspezifikation, Anforderungsanalyse und -verarbeitung sowie zur Codegenerierung und -migration eingesetzt werden. Darüber hinaus unterstützen LLMs die Qualitätssicherung und das Testing, indem sie bei der automatischen Generierung von Testfällen (ggf. auch Testdaten) und -szenarien helfen.

3. Existierende Arbeiten

Eine Analyse existierender Arbeiten zum Einsatz Generativer KI, insbesondere LLMs im Software Reengineering Kontext brachte die folgenden Schwerpunkte:

LLMs in der Softwaretechnik haben die Fähigkeit, Codeschnipsel oder komplette Programme zu generieren, Code zum besseren Verständnis zusammenzufassen und Schwachstellen und Fehler im Code zu erkennen. Sie dienen als wertvolle Werkzeuge für Aufgaben wie automatisierte Codegenerierung, Codedokumentation und Codeanalyse und sind daher in allen Phasen des Software Engineerings nützlich [1].

Die Implementierung der Koordination von LLMs und Plattformwissen für die Software-Modernisierung beinhaltet die Gestaltung von Arbeitsabläufen, die LLMs mit Prompt-Engineering, Qualitätskontrolle und Fusions-/Aggregationsstrategien einbeziehen. Auf der Ebene des Ensembles können parallele Muster mit mehreren Verzweigungen und Aggregation sowie Sequenzmuster mit Modellausgaben als Eingaben verwendet werden. Auf der Ebene der einzelnen LLM können Parameterabstimmung und RAG-Strategien (Red, Amber, Green) eingesetzt werden. Darüber hinaus können Low-Code und automatische Koordinationslogik entwickelt werden, um den Aufwand für Ingenieure zu reduzieren [2].

Zu den ermittelten Schwerpunkten wurden Studien identifiziert, welche die Leistungsfähigkeit unterschiedlicher LLMs in den Anwendungsfeldern des Software Reengineering untersuchten:

Eine Studie von Federico Cruciani et al. (2023) untersuchte die Leistungsfähigkeit von LLMs bei der Klassifikation von Anforderungen unter Verwendung eingebetteter Wörter. Die Analyse umfasste die Klassifizierung von Anforderungen in funktionale und nicht-funktionale Kategorien sowie die Identifizierung der häufigsten Klassen. Darüber hinaus wurde die Leistung der LLMs bei der Klassifizierung in alle vorhandenen Klassen untersucht. [3]

In der Softwaretechnikforschung wird versucht, Entwicklungsprozesse basierend auf informell formulierten Absichten der Entwickler durch automatisierte Ansätze zu unterstützen. Eine Reihe von Methoden zielen darauf ab, natürlichsprachliche Beschreibungen in Quellcode umzuwandeln, wobei jedoch noch kein umfassendes Verständnis für deren Effektivität und Ergänzung untereinander besteht. Eine groß angelegte empirische Studie zeigt, dass kombinierte Techniken zur natürlichsprachlichen Codesuche und -generierung die Leistung um 35% im Vergleich zur besten eigenständigen Methode verbessern können. [4]

4. Prototypische Klassifikation von Anforderungen

Um die Forschungsfragen zu beantworten, haben wir uns für experimentelles Prototyping entschieden. Diese Methode ermöglicht praxisnahe und qualitätsorientierte Ergebnisse. Wir verwenden die vorklassifizierten Datensätze aus der Evaluation von AutoML von 2023 [5]. Diese 147 Softwareanforderungen haben folgenden Struktur:

„The software should have a community forum feature, Low“.

Der Prototyp ist in Jupiter Notebook mit phyton implementiert. Es wird als ML Vertreter scikit-learn [6] trainiert und getestet. Hierfür werden die Datensätze in Trainings- (80%) und Testdaten (20%) aufgeteilt, wobei ein Random Seed von 42 verwendet wurde. Die LLM Vertreter sind ChatGPT 3.5 Turbo und ChatGPT 4 von OpenAI [7]. Diese werden per restful WeoAPI verwendet mit jeweils einen Prompt pro Datensatz, ohne die Vorklassifizierung mitzusenden. Somit erfolgt kein Training (ZeroShot).
In Abbildung 1 wird der Ablauf der Experimente kurz dargestellt.

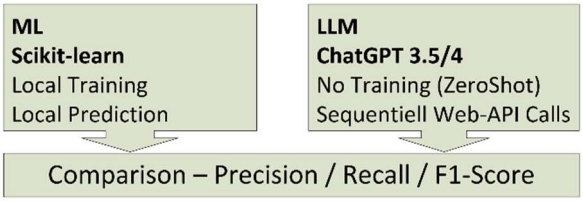

Abbildung 1 Schematischer Ablauf der Experimente im Prototypen

Der Vergleich zwischen den beiden Ansätzen erfolgt anhand verschiedener Metriken: Für die ML-Modelle wurden gewichtete Durchschnittswerte verwendet, während bei den LLM-Modellen die Übereinstimmungen mit den manuellen Klassifikationen aus den Rückgaben der Web-APIs verglichen wurden. Die ersten Ergebnisse sind in Tabelle 1 aufgeführt.

Tabelle 1 Vergleich von LLMs und legacy KI

	scikit-learn	ChatGPT 3.5	ChatGPT 4
time	1sec	15min	50min
Precision	0.60	0.55	0.29
Recall	0.63	0.35	0.01
F1 Score	0.54	0.29	0.01

5. Reflektion und Ausblick

Die Ergebnisse belegen, dass trainierte ML-Modelle für die Klassifizierung des Aufwands von Anforderungen eindeutig besser geeignet sind als LLMs. Die Präzision, Recall und F1-Score der ML-Modelle waren deutlich höher als die der LLMs. Dies lässt sich zum einen auf die effektive Vorverarbeitung der Daten zurückführen, die für die ML-Modelle durchgeführt wurde, und zum anderen auf das gezielte Finetuning, das auf die spezifische Aufgabe der Aufwandsklassifizierung abgestimmt war.

Die Laufzeiten der verschiedenen Ansätze variieren erheblich. Während ML-Modelle schnellere Ergebnisse liefern, erfordern LLMs deutlich mehr Zeit. Dies ist bedingt durch die Netzwerkkommunikation und die generative Funktionsweise, die wesentlich mehr Ressourcen benötigt.

Die Ergebnisse lassen den Schluss zu, dass es Raum für Verbesserungen bei der Nutzung von LLMs für die Klassifizierung von Anforderungen gibt. Mögliche Ansätze sind eine feinere Abstimmung der Modelle, Training, Finetuning und die Optimierung des Anfragepromptes. Die Initialfragen können beantwortet werden:

FF1 LLMs wie ChatGPT 3.5 und ChatGPT 4 können die Modernisierung von Altanwendungen unterstützen, indem sie natürlichsprachliche Anforderungen analysieren und klassifizieren. Allerdings ist die Ergebnisqualität untrainiert im Vergleich zu traditionellen ML-Modellen möglicherweise begrenzt.

FF2 LLMs können effiziente Software-Reengineering-Prozesse unterstützen, indem sie Anforderungen automatisiert analysieren und verarbeiten. Jedoch sind ihre Ressourcenanforderungen im Vergleich zu traditionellen ML-Modellen höher.

Insgesamt bieten die Experimente Einblicke in die Leistungsfähigkeit verschiedener Ansätze zur Klassifizierung des Aufwands von Anforderungen und zeigen potenzielle Bereiche für zukünftige Forschung und Verbesserungen auf.

6. Quellenverzeichnis

[1] ZHENG, Z. ; NING, K. ; CHEN, J. ; WANG, Y. ; CHEN, W. ; GUO, L. ; WANG, W.: *Towards an Understanding of Large Language Models in Software Engineering Tasks*. URL http://arxiv.org/pdf/2308.11396.pdf. – Aktualisierungsdatum: 2023-08-22

[2] LINH TRUONG ; MAJA VUKOVIC ; RAJU PAVULURI: *On Coordinating LLMs and Platform Knowledge for Software Modernization and New Developments*. WorkingPaper. URL https://acris.aalto.fi/ws/portalfiles/portal/133671133/collms_position.pdf. – Aktualisierungsdatum: 2023-11-22

[3] FEDERICO CRUCIANI ; SAMUEL MOORE ; CD NUGENT: Comparing general purpose pre-trained Word and Sentence embeddings for Requirements Classification, Bd. 3378. In: *Joint Proceedings of REFSQ-2023 Workshops, Doctoral Symposium, Posters & Tools Track and Journal Early Feedback (REFSQ-JP 2023)* : CEUR-WS, 2023

[4] WANG, S. ; GENG, M. ; LIN, B. ; SUN, Z. ; WEN, M. ; LIU, Y. ; LI, L. ; BISSYANDÉ, T.F. ; MAO, X.: Natural Language to Code: How Far Are We? In: CHANDRA, Satish; BLINCOE, Kelly; TONELLA, Paolo (Hrsg.): *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA : ACM, 2023, S. 375–387

[5] SCHMIETENDORF, A. ; HARTENSTEIN, S. ; JOHNSON, S.L.: KI-gestützte Modernisierung von Altanwendungen: (Sentiment-) Analysen im Diskurs des Anforderungsmanagements. In: *Workshop Software-Reengineering und -Evolution WSRE 2023*, 2023, S. 20–21

[6] PEDREGOSA, F. ; VAROQUAUX, G. ; GRAMFORT, A. ; MICHEL, V. ; THIRION, B. ; GRISEL, O. ; BLONDEL, M. ; PRETTENHOFER, P. ; WEISS, R. ; DUBOURG, V. ; VANDERPLAS, J. ; PASSOS, A. ; COURNAPEAU, D. ; BRUCHER, M. ; PERROT, M. ; DUCHESNAY, E.: *Scikit-learn: Machine Learning in Python*. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830

[7] OPENAI: *ChatGPT API*. 2022. URL https://platform.openai.com/docs/overview