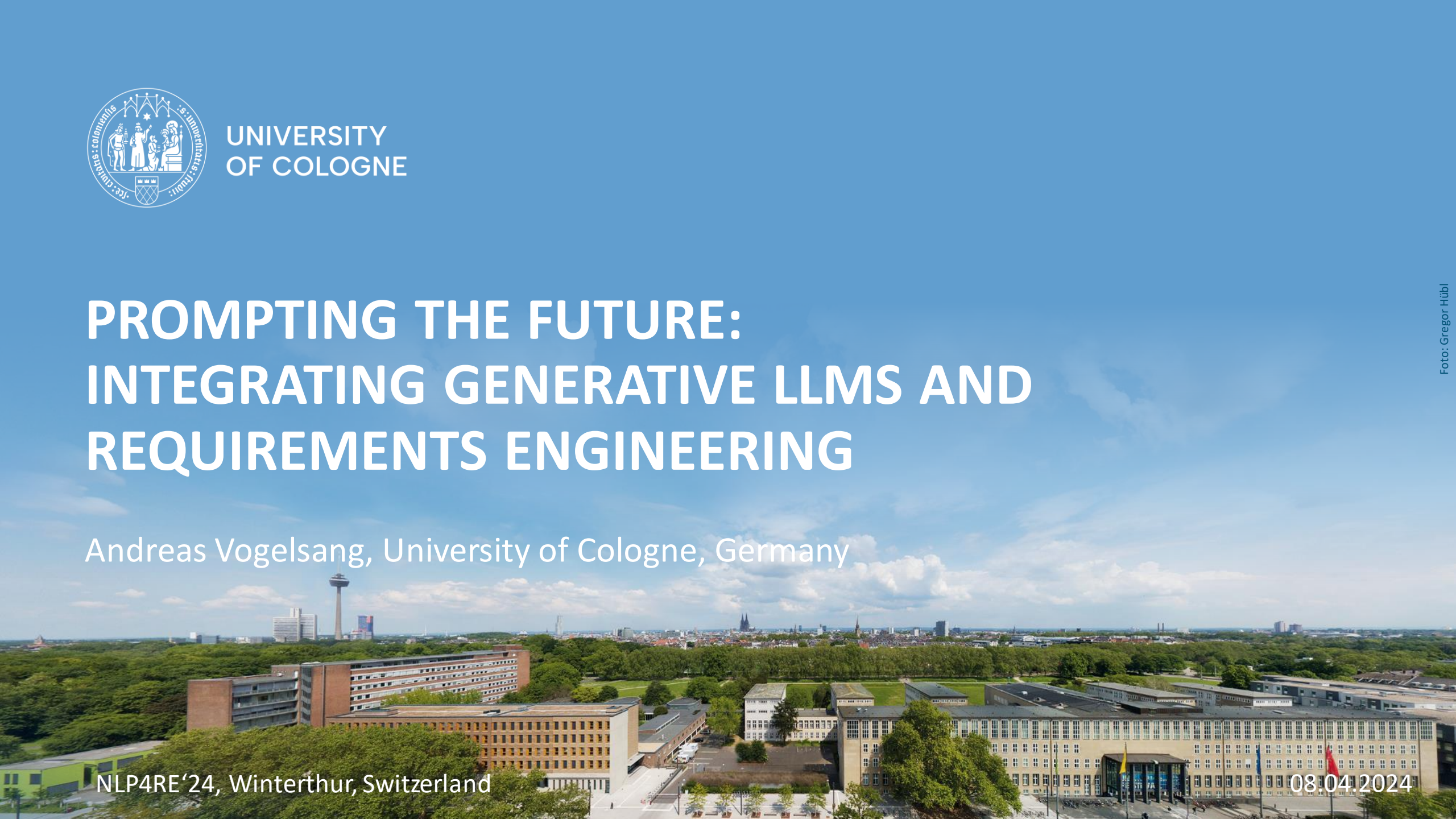


UNIVERSITY OF COLOGNE

PROMPTING THE FUTURE: INTEGRATING GENERATIVE LLMS AND REQUIREMENTS ENGINEERING

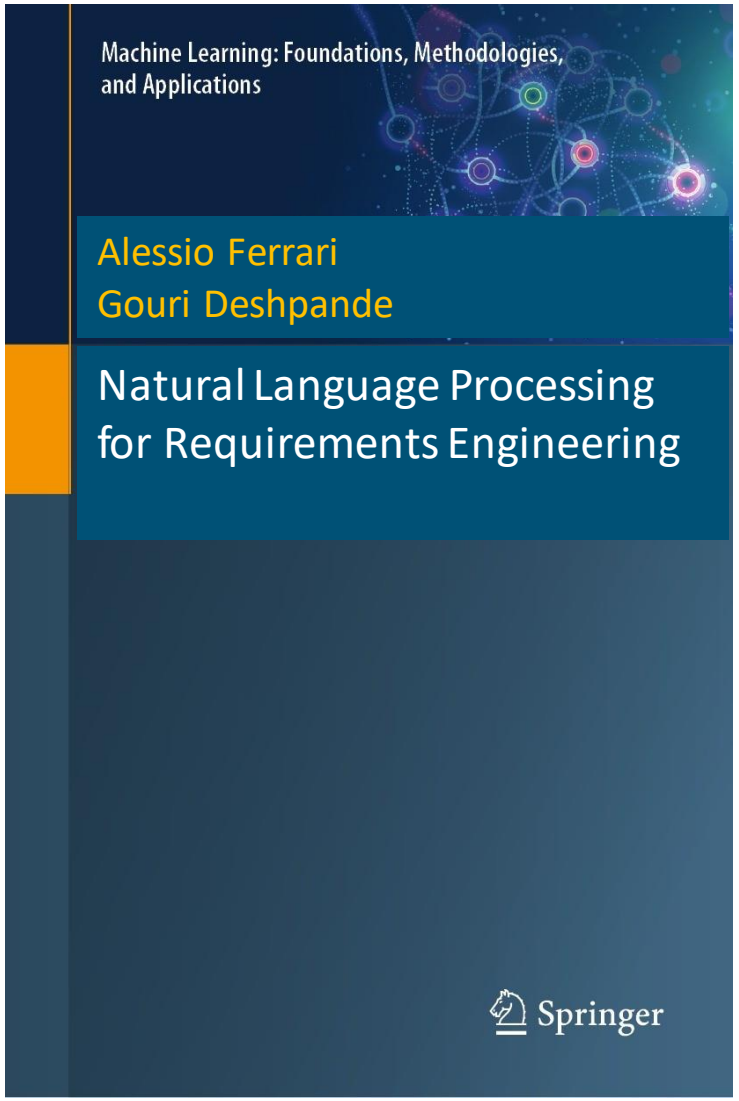
Andreas Vogelsang, University of Cologne, Germany

Foto: Gregor Hübl

NLP4RE‘24, Winterthur, Switzerland 08.04.2024

Disclaimer and Acknowledgements

Parts of the talk are inspired by the chapter “Using Large Language Models for Natural Language Processing Tasks in Requirements Engineering: A Systematic Guideline”
from the upcoming book
“Natural Language Processing for Requirements Engineering”



Acknowledgements

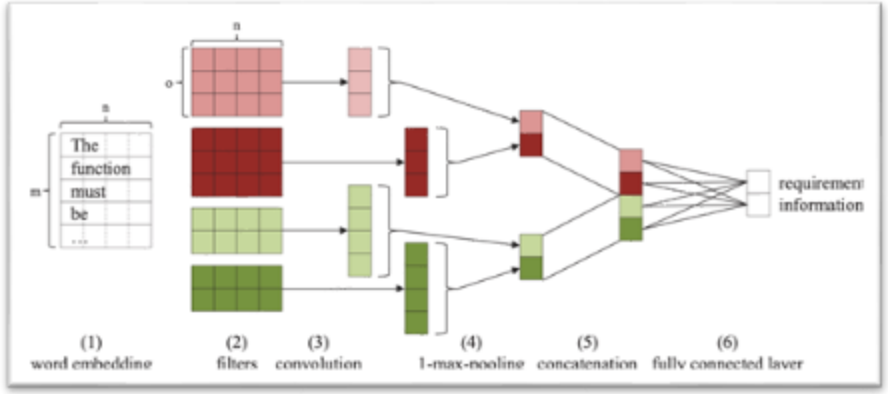




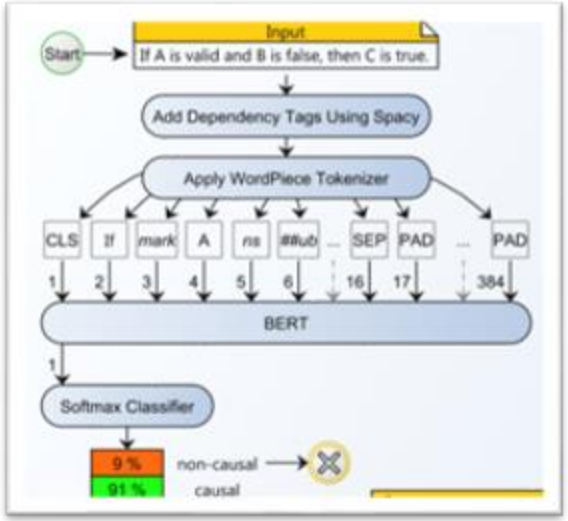

J. Fischbach C. Arora S. Abualhaija
A. Ferrari F. Dalpiaz G. Deshpande

My own story with NLP, ML, and LLMs for RE

- 2016: CNN for Requirements Classification
- 2017: Understanding DNN Classifiers
- 2018: Human-in the-loop evaluations
- 2019: Explainable tools
- Since 2020: Classification and Extraction with BERT et al.
  - Conditionals
  - Kano model factors
  - Explainability needs
  - Test case creation
- Since 2023: Exploring the potentials of Generative LLMs



requirement	the duration until the switch is recognized as hanging must be a configurable parameter .
information	the component conditionally drives an external fan . this fan is required for active ventilation of the headlight .



The times when I got really excited

- 2016: My first experience with end-to-end ML
- 2017: Keynote by Vincenzo Gervasi at AIRE'17
- 2020: BERT -> Data problem seemed to be solved
- 2023: Release of chatGPT

Generating data

(Stacked) RBMs can be used to **generate** ideal forms of data.

A scenario for **unsupervised learning**:

- the network is trained on a number of untagged instances
  - this sets weights on the links
- then, random values are set on the nodes, the network is run until stabilization, and output is read on the "input" nodes
- This is sometimes referred as **machines dreaming**

In RE applications:

- What if I reverse the FR/NFR classifier and ask to generate a requirement (given the class) instead?
  - Creativity-enhancing techniques
- Once we have trained a network in an unsupervised fashion, what can we learn from the synthesized stable states?
- Let's feed requirements at a RBM. It wil implicitly classify them according to "invented" classes
  - Will these classes mirror the IEEE Stds?

https://aire-ws.github.io/aire17/resources/Vincenzo_Gervasi-Of_Forms_and_Structures.pdf

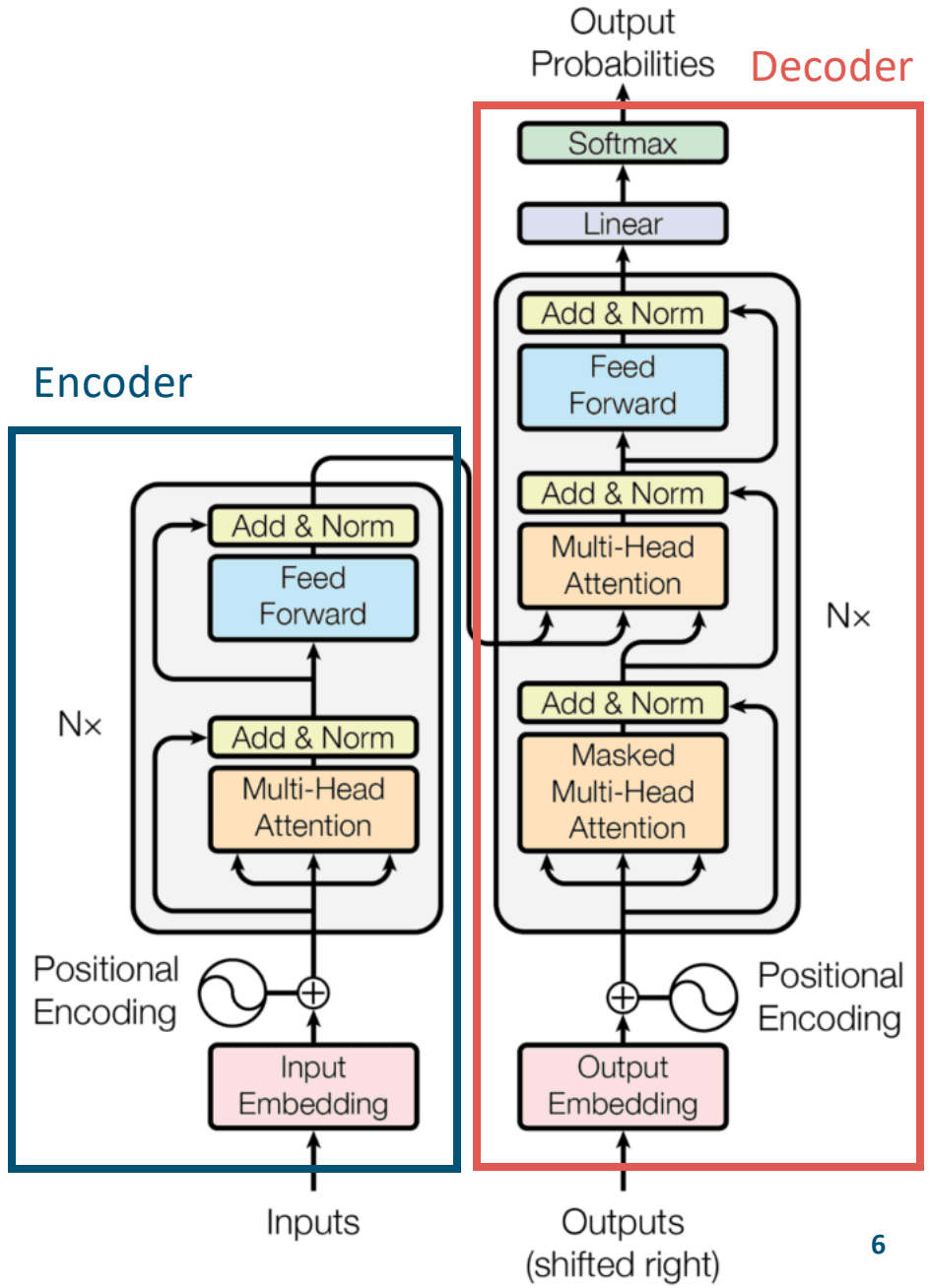
01

Preliminaries: Decoder-only LLMs

UNIVERSITY
OF COLOGNE

The Transformer Architecture

- **Encoder-decoder** architecture (e.g., the T5 family)
  - *Translative** LLMs
  - Translate input into corresponding output text
- **Encoder-only** architecture (e.g., the BERT family)
  - *Predictive* LLMs
  - Predicts existing patterns or outcomes
- **Decoder-only** architecture (e.g., the GPT family)
  - *Generative* LLMs
  - Generates text based on a given context (*prompt*)



* *Category invented by me*

Prompting

- A prompt may include
  - **Instructions**: a specific task or instruction you want the model to perform
  - **Context**: external information or additional context that can steer the model to better responses
  - **Input Data**: the input or question that we are interested in finding a response for
  - **Output Indicator**: the type or format of the output.
- Brings the decoder into a desired state (the context)

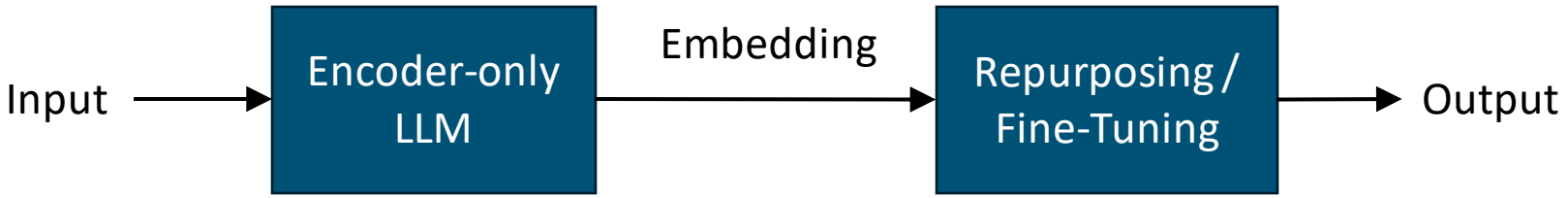
- A new input paradigm
  - No need to „model“ the problem by preprocessing input data
  - Instead, stating the problem in natural language

02

Generative LLMs for RE

Generative LLMs can replace Predictive LLMs

Predictive LLM:



Generative LLM:

Example: Automated Traceability

Below are artifacts from the same software system. Is there a traceability link between (1) and (2)? Answer “yes” or “no“.

(1) The DPU-TMALI shall utilize SCM_DCI_SR, along with ERRNO provided by DPU-DCI to decode errors and place them on an error queue for DPU-CCM.

(2) Error Collection and Reporting The ccmErrEnq() function tracks the last error reported and its frequency of occurrence. Once an error code has been reported it becomes the previously reported error code maintained by ccmErrEnq(). […]

Yes

37% precision and 84% recall without any training

Rodriguez, A.D., Dearstyne, K.R., Cleland-Huang, J.: Prompts matter: Insights and strategies for prompt engineering in automated software traceability. In: *AIRE‘23*

Explanations and Rationales

Below are artifacts from the same software system. Is there a traceability link between (1) and (2)?

(1) The DPU-TMALI shall utilize SCM_DCI_SR, along with ERRNO provided by DPU-DCI to decode errors and place them on an error queue for DPU-CCM.

(2) Error Collection and Reporting The ccmErrEnq() function tracks the last error reported and its frequency of occurrence. Once an error code has been reported it becomes the previously reported error code maintained by ccmErrEnq(). […]

Yes, there is a traceability link between (1) and (2).
(1) mentions that the DPU-TMALI places errors on an error queue for the DPU-CCM.
(2) describes the error collection and reporting mechanism of the DPU-CCM, specifically referring to enqueuing errors.

So (2) provides an implementation detail for the error queuing functionality briefly mentioned in (1). Therefore, there is a traceability link between these two artifacts in the software system.

Starting a Dialog

The explanations is not correct.
(2) does not describe the DPU-CCM but another component.

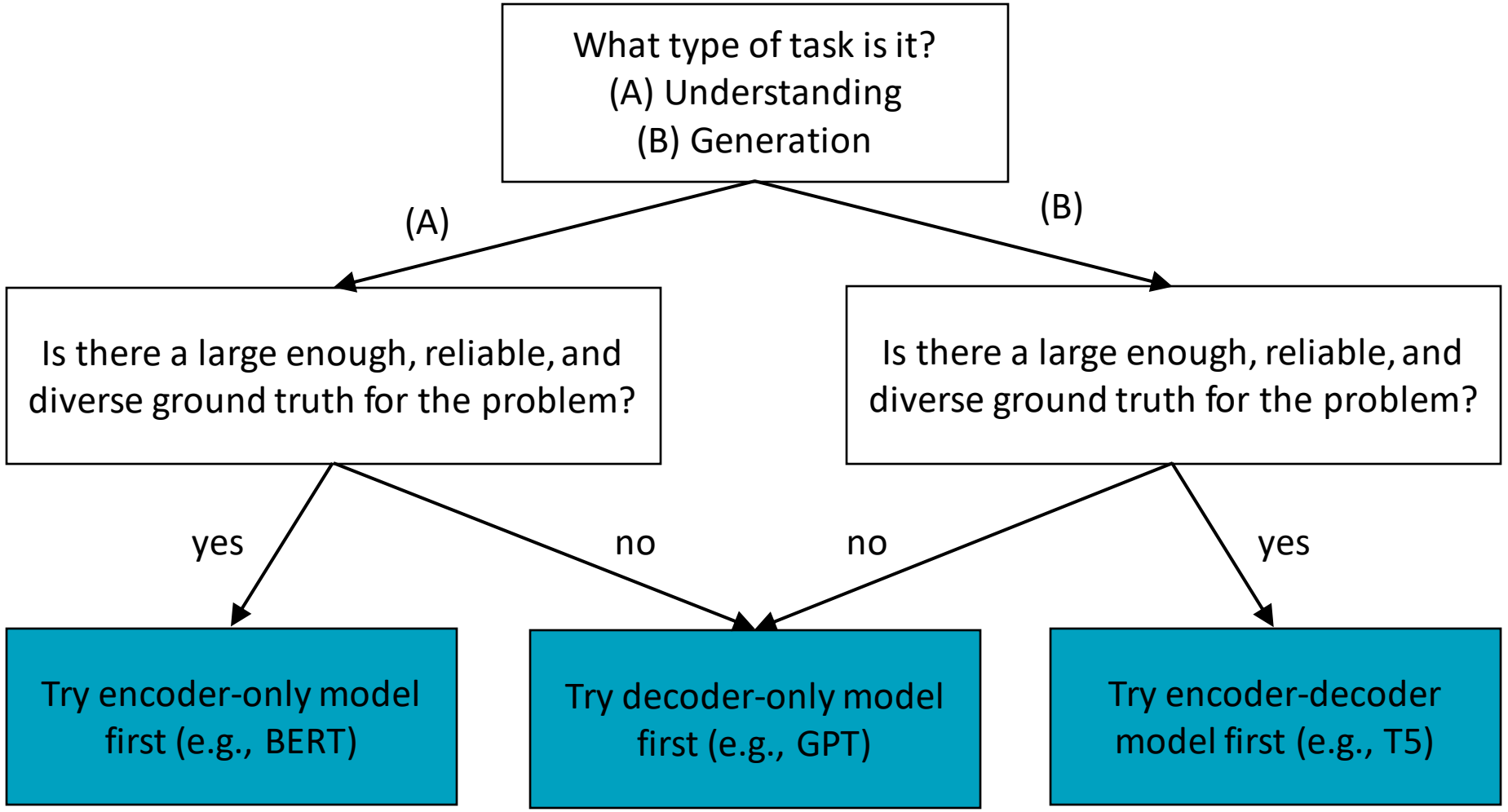
I apologize for the confusion in my previous response. Based on the information provided, there is no direct traceability link between (1) and (2) in terms of specific components or functionalities. (1) describes the error handling process in the DPU-TMALI component, while (2) describes the error collection and reporting mechanism in a different component that is not explicitly mentioned.

Without further context or explicit connections between the components mentioned in (1) and (2), it is not possible to establish a direct traceability link between the two artifacts.

How can I change (2) to improve the traceability?

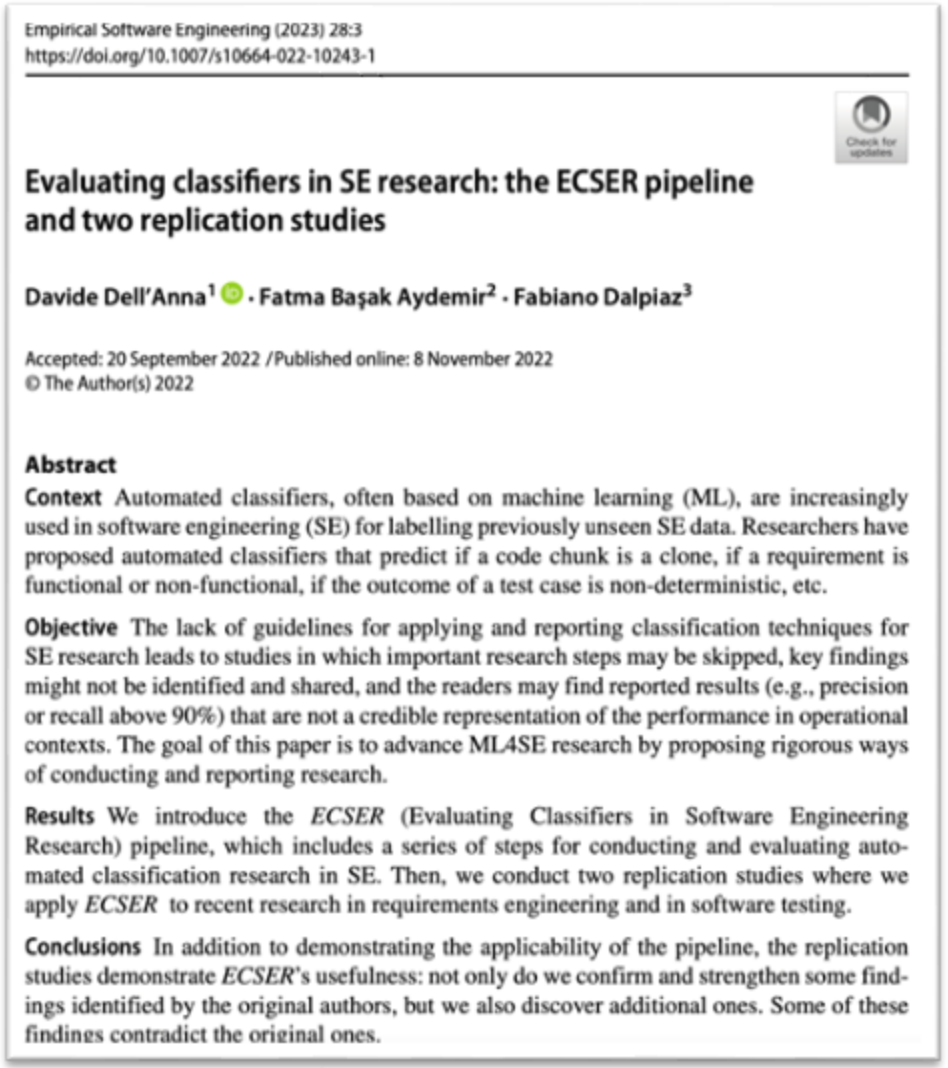
To improve traceability between (1) and (2), you can modify (2) to explicitly mention the relationship with the DPU-CCM component mentioned in (1). Here's how you can adjust the description:
[...]

Challenge: Which Architecture to Choose?



Challenge: How to evaluate Generative LLMs?

- Predictive tasks:
  - Follow best practices
  - Is a generative LLM necessary?
  - Does it justify the additional computational effort?



Empirical Software Engineering (2023) 28:3
https://doi.org/10.1007/s10664-022-10243-1

Evaluating classifiers in SE research: the ECSER pipeline and two replication studies

Davide Dell'Anna[1] · Fatma Başak Aydemir[2] · Fabiano Dalpiaz[3]

Accepted: 20 September 2022 / Published online: 8 November 2022
© The Author(s) 2022

Abstract

Context Automated classifiers, often based on machine learning (ML), are increasingly used in software engineering (SE) for labelling previously unseen SE data. Researchers have proposed automated classifiers that predict if a code chunk is a clone, if a requirement is functional or non-functional, if the outcome of a test case is non-deterministic, etc.

Objective The lack of guidelines for applying and reporting classification techniques for SE research leads to studies in which important research steps may be skipped, key findings might not be identified and shared, and the readers may find reported results (e.g., precision or recall above 90%) that are not a credible representation of the performance in operational contexts. The goal of this paper is to advance ML4SE research by proposing rigorous ways of conducting and reporting research.

Results We introduce the *ECSER* (Evaluating Classifiers in Software Engineering Research) pipeline, which includes a series of steps for conducting and evaluating automated classification research in SE. Then, we conduct two replication studies where we apply *ECSER* to recent research in requirements engineering and in software testing.

Conclusions In addition to demonstrating the applicability of the pipeline, the replication studies demonstrate *ECSER*'s usefulness: not only do we confirm and strengthen some findings identified by the original authors, but we also discover additional ones. Some of these findings contradict the original ones.

Challenge: How to evaluate Generative LLMs?

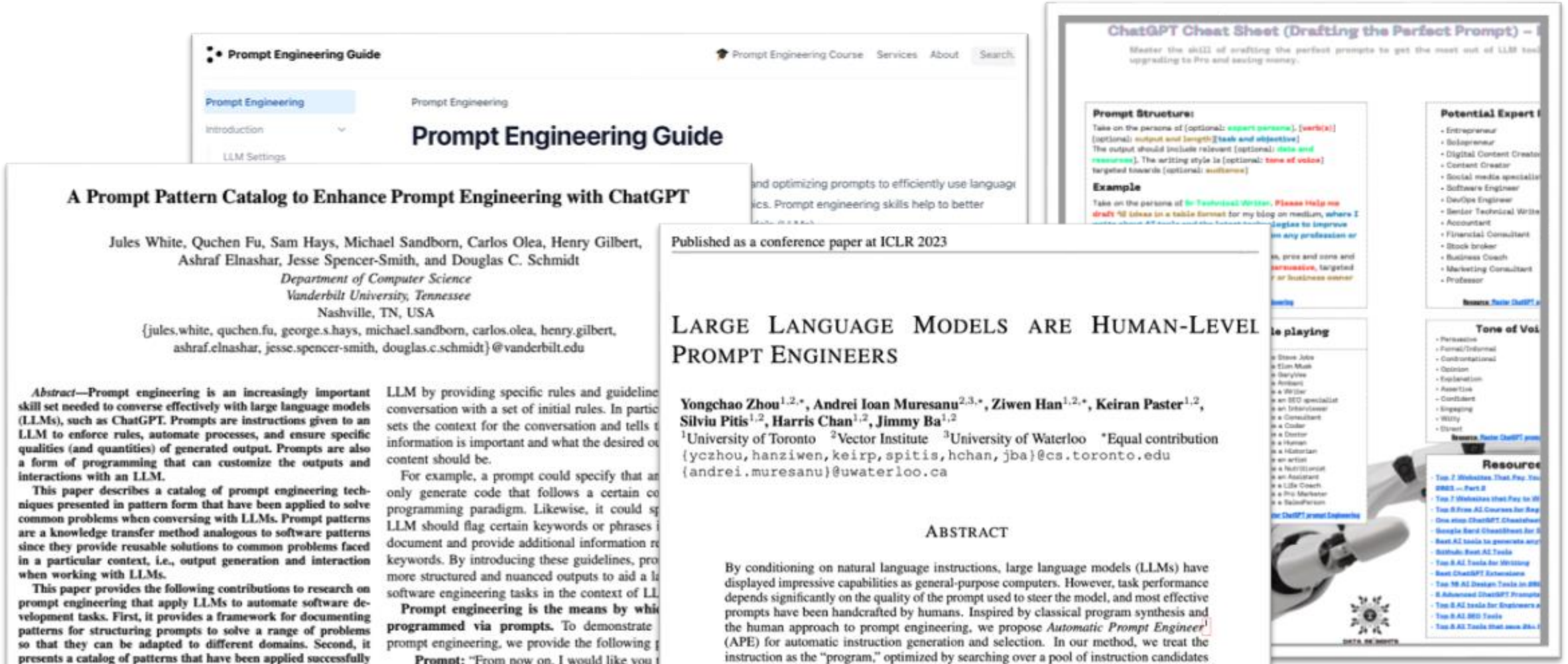
- Performance
  - Overlap-based (e.g., BLEU, ROUGE, METEOR)
  - Semantic Similarity (e.g., BERTScore)
- User
  - HiL performance
  - Perceived quality
  - Acceptance
  - Feedback
- Cost
  - Operation cost
  - Nr. of tokens
- Ethics
  - Regulation
  - Harmfulness
  - Hallucination
  - Transparency

The human in the loop becomes even more important for evaluating generative LLMs

03

RE for Effective Prompting

Prompt Engineering: State of the Art



Prompt Engineering Guide

Prompt Engineering Course Services About Search

Prompt Engineering

Introduction

LLM Settings

Prompt Engineering

Prompt Engineering Guide

...and optimizing prompts to efficiently use language
...ics. Prompt engineering skills help to better

A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT

Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert,
Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt
Department of Computer Science
Vanderbilt University, Tennessee
Nashville, TN, USA
{jules.white, quchen.fu, george.s.hays, michael.sandborn, carlos.olea, henry.gilbert,
ashraf.elnashar, jesse.spencer-smith, douglas.c.schmidt}@vanderbilt.edu

***Abstract*—Prompt engineering is an increasingly important skill set needed to converse effectively with large language models (LLMs), such as ChatGPT. Prompts are instructions given to an LLM to enforce rules, automate processes, and ensure specific qualities (and quantities) of generated output. Prompts are also a form of programming that can customize the outputs and interactions with an LLM.**

This paper describes a catalog of prompt engineering techniques presented in pattern form that have been applied to solve common problems when conversing with LLMs. Prompt patterns are a knowledge transfer method analogous to software patterns since they provide reusable solutions to common problems faced in a particular context, i.e., output generation and interaction when working with LLMs.

This paper provides the following contributions to research on prompt engineering that apply LLMs to automate software development tasks. First, it provides a framework for documenting patterns for structuring prompts to solve a range of problems so that they can be adapted to different domains. Second, it presents a catalog of patterns that have been applied successfully to improve the outputs of LLM conversations. Third, it explains how prompts can be built from multiple patterns and illustrates prompt patterns that benefit from combination with other prompt patterns.

LLM by providing specific rules and guideline
conversation with a set of initial rules. In partic
sets the context for the conversation and tells t
information is important and what the desired ou
content should be.

For example, a prompt could specify that an
only generate code that follows a certain co
programming paradigm. Likewise, it could sp
LLM should flag certain keywords or phrases i
document and provide additional information re
keywords. By introducing these guidelines, pro
more structured and nuanced outputs to aid a la
software engineering tasks in the context of LL

**Prompt engineering is the means by whi
programmed via prompts.** To demonstrate
prompt engineering, we provide the following p

Prompt: "From now on, I would like you t
questions to deploy a Python application
When you have enough information to de
application, create a Python script to auto

Published as a conference paper at ICLR 2023

LARGE LANGUAGE MODELS ARE HUMAN-LEVEL PROMPT ENGINEERS

Yongchao Zhou[1,2,*], Andrei Ioan Muresanu[2,3,*], Ziwen Han[1,2,*], Keiran Paster[1,2], Silviu Pitis[1,2], Harris Chan[1,2], Jimmy Ba[1,2]
[1]University of Toronto [2]Vector Institute [3]University of Waterloo [*]Equal contribution
{yczhou,hanziwen,keirp,spitis,hchan,jba}@cs.toronto.edu
{andrei.muresanu}@uwaterloo.ca

ABSTRACT

By conditioning on natural language instructions, large language models (LLMs) have displayed impressive capabilities as general-purpose computers. However, task performance depends significantly on the quality of the prompt used to steer the model, and most effective prompts have been handcrafted by humans. Inspired by classical program synthesis and the human approach to prompt engineering, we propose *Automatic Prompt Engineer*[1] (APE) for automatic instruction generation and selection. In our method, we treat the instruction as the "program," optimized by searching over a pool of instruction candidates proposed by an LLM in order to maximize a chosen score function. To evaluate the quality of the selected instruction, we evaluate the zero-shot performance of another LLM following the selected instruction. Extensive experiments show that our automatically generated instructions outperform the prior LLM baseline by a large margin and achieve better or comparable performance to the instructions generated by human annotators on 24/24 Instruction Induction tasks and 17/21 curated BIG-Bench tasks. We conduct extensive qualitative and quantitative analyses to explore the performance of APE. We show that APE-engineered prompts are able to improve few-shot learning performance (by simply prepending them to standard in-context learning prompts), find better zero-shot chain-of-thought prompts, as well as steer models toward truthfulness and/or informativeness.[2]

ChatGPT Cheat Sheet (Drafting the Perfect Prompt) –

Prompt Structure:

Example

Potential Expert

Tone of Voi

Resource

UNIVERSITY OF COLOGNE

Prompt Engineering from an RE Perspective

Prompts are expressions of requirements!

- Goals of RE [IREB]
  - Knowing all relevant requirements
  - Achieving a consensus among the stakeholders about these requirements
  - Documenting requirements appropriately
  - Managing requirements systematically
- Goals of PromptRE?
  - Knowing all relevant prompts
  - Achieving a consensus among the stakeholders about these prompts
  - Documenting prompts appropriately
  - Managing prompts systematically

Prompt Smells?

We want to develop a game for mobile devices. The game has two players who throw a dice in each turn. Both players start with 10 points. If the dice shows an even number, the number is added to the points of the player, if it shows an uneven number, the points are subtracted. ***If the points of one player are too low, the game ends and the player loses the game.***

Please create code in Python that implements this game.

Certainly! Below is a Python implementation of the game:
[…]

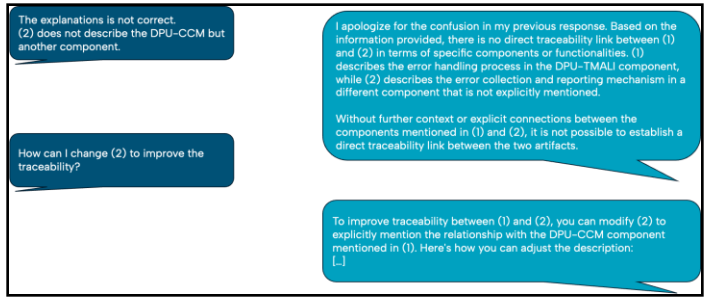
```
if player1_points <= 0:
                print("Player 1 has lost the game!")
                break
```

[…]

RE for and with Prompts

- General
  - What is the role of prompts in the development process? Are they requirements? Intermediate (transitory) artifacts? High-level code?
- Traceability
  - Which prompts have been derived from which requirements?
  - Is all prompt content backed up by requirements?
- Prompt Management
  - Which (version of a) prompt generated which code?
  - Which LLM (version) has been used?
- Prompt documentation
  - Are prompts a good representation of requirements?

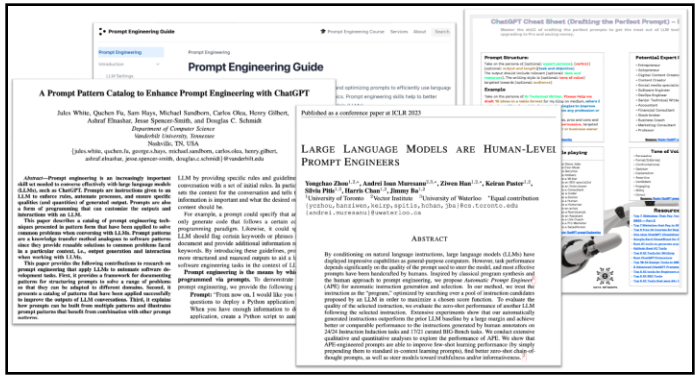
Summary and Take-Home Messages



Explore the new opportunities of generative LLMs (explanations, dialogs)

- Performance
  - Overlap-based (e.g., BLEU, ROUGE, METEOR)
  - Semantic Similarity (e.g., BERTScore)
- User
  - HiL performance
  - Perceived quality
  - Acceptance
  - Feedback
- Cost
  - Operation cost
  - Nr. of tokens
- Ethics
  - Regulation
  - Harmfulness
  - Hallucination
  - Transparancy

Mind the evaluation!



RE for and with prompts is an open field

www.linkedin.com/in/prof-andreas-vogelsang
vogelsang@cs.uni-koeln.de

UNIVERSITY OF COLOGNE