Prof. Dr. Jasmin Blanchette
Dr. Martin Desharnais-Schäfer
Dr. Michael Kirsten
Elisabeth Lempa

Ludwig-Maximilians-Universität München
Institut für Informatik
Discussion on 10.12.2025
Homework due on 17.12.2025 at 16:00

Possible solution for Exercise Sheet 9 in

Scientific and Technical English for Computer Scientists

The exercise sheets consist of in-class exercises and homework. The in-class exercises take place during the second half of the lecture time slots. The homework, which is optional and ungraded, can be submitted via the "Homework" section in Moodle. The homework is subject to peer review.

Unless indicated otherwise, generative artificial intelligence assistants such as ChatGPT may be used, as long as you acknowledge how you use them as specified by the Institute's policy on plagiarism.[1] However, you may not use such tools to generate peer reviews for you. In addition, we strongly recommend that you do not use them to generate entire solutions, since that would defeat the purpose of the exercises.

Homework 9-4 ***Explaining a Topic*** For each of the following two papers, write a text of about 250 words that explains the topic using appropriate citations and quotations to acknowledge the paper as your source. You will probably not need to do a close reading of the papers.

a) Han Qiao et al. "To use or not to use: Impatience and overreliance when using generative AI productivity support tools." In: *CHI 2025*. ACM, 2025. URL: `https://dl.acm.org/doi/10.1145/3706598.3714103`

POSSIBLE SOLUTION:

For better or worse, we live in a world where generative artificial intelligence (GenAI) tools are readily available for various tasks. When we choose whether to employ GenAI for a task, we might make this choice for any number of reasons, and, depending on the goal we have in mind, it might be impossible to tell whether we made the correct one. However, when the goal is productivity, and there is suitable data available for both the user's performance and the GenAI tool's latency and error rate, we can compare the user's choice with the optimal choice.

If we know the time and error rates of both the GenAI tool and users performing the task on their own, then the question of whether to use GenAI or

[1] `https://www.medien.ifi.lmu.de/lehre/Plagiate-IfI.pdf`

not becomes a simple arithmetic problem. However, even if they have perfect information about the GenAI tools performance, users do not always make the optimal choice, as shown by Qiao et al. [1]. In their simulated environment, they find that users sometimes chose to complete a task manually, even though employing the assistance would be faster, which Qiao et al. call the "gulf of impatience," and also users sometimes chose to wait for the tool to complete the task, even they would have been faster completing it themselves, which they call the "gulf of overreliance."

References

[1] Han Qiao, Jo Vermeulen, George Fitzmaurice, and Justin Matejka. "To use or not to use: Impatience and overreliance when using generative AI productivity support tools." In: *CHI 2025*. ACM, 2025. URL: `https://dl.acm.org/doi/10.1145/3706598.3714103`.

b) Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified data processing on large clusters." In: *OSDI 2004*. USENIX Association, 2004, pp. 137–150. URL: `http://www.usenix.org/events/osdi04/tech/dean.html`

POSSIBLE SOLUTION:

MapReduce is a functional programming idiom introduced by Dean and Ghemawat [1] that abstracts away the complexity of parallelizing and distributing simple computations on a large cluster of computers. The user must express their computation in terms of two functions: `map` : $k_1 \times v_1 \to$ `list` $(k_2 \times v_2)$ and `reduce` : $k_2 \times$ `list` $v_2 \to$ `list` v_2. A MapReduce implementation splits the input problem into a set of key–value pairs to which it applies the `map` function. The `map` function takes a key–value pair as input and outputs a list of intermediate keys and values. The MapReduce implementation then partitions the intermediate values by keys and applies the `reduce` function. The `reduce` function takes an intermediate key and a list of intermediate values as input and merges them into a smaller list that it outputs. For each intermediate key, the reduced list of intermediate values is interpreted as part of the computation's final output.

For example, counting the number of occurrences of each word in a collection of documents can be expressed using MapReduce. The `map` function gets a document name and content as input and outputs a list of each word with its number of occurrences. The `reduce` function gets a word and a list of number of occurrences as input and outputs a singleton list containing the sum of of all occurrences of the word.

Dean and Ghemawat report great success using this technique at Google

and claim that "restricting the programming model makes it easy to parallelize and distribute computations and to make such computations fault-tolerant."

References

[1] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified data processing on large clusters." In: *OSDI 2004*. USENIX Association, 2004, pp. 137–150. URL: `http://www.usenix.org/events/osdi04/tech/dean.html`.