Task 1.1: Summarize main findings of Allison et al. 2010, focusing on potential relevance in the agricultural setup, max 200 words.

The article proposes a new approach to modeling carbon fluxes in soil under climate change, specifically the impact of increasing temperatures on soil organic carbon (SOC) and dissolved organic carbon (DOC). The model includes an enzyme-driven feedback loop between SOC and DOC that could explain the observed decline in CO2 loss from soil after an initial increase. The research is relevant to agriculture because carbon fluxes and microbial activity are critical for both climate change mitigation and agricultural productivity. However, the model has limitations for direct application in agriculture. Firstly, it assumes microbial pool composition, which is unrealistic for farmlands due to the impact of fertilization and management practices [1]. Secondly, the model assumes a discrete temperature increase, while real-world soils are subject to daily and seasonal fluctuations[2] that have an impact on CUE and MIC. Lastly, the model does not account for soil moisture, which influences the microbe/enzyme relationship, CUE, and DOC [3]. Therefore, further research and adaptation of the model are necessary before it can be applied directly to agricultural practices.

[1] Dilly, Oliver, et al. "Variation of stabilised, microbial and biologically active carbon and nitrogen in soil under contrasting land use and agricultural management practices." *Chemosphere* 52.3 (2003): 557-569.
[2]Parkin, Timothy B., and Thomas C. Kaspar. "Temperature controls on diurnal carbon dioxide flux: Implications for estimating soil carbon loss." *Soil Science Society of America Journal* 67.6 (2003): 1763-1772.
[3] Bian, Hongfeng, et al. "Soil moisture affects the rapid response of microbes to labile organic C addition." *Frontiers in Ecology and Evolution* (2022): 604.

Task 1.2: . Plan the implementation of the model into a code (eg. in Python, R, other) based on the paper Allison et al. 2010 and the supplementary material.

Model design:

The model is constituted by the following processes and variables:

- Processes: Catalysis, Uptake, Enzyme production, Enzyme decay, Microbial death
- Variables: SOC, DOC, ENZ, MIC, CO2, TEMP

In a python implementation of the model processes can be conceived as classes, each with its own parameters and with a '*.calc()*' method, that takes in relevant variables as input and produces an output expressed in one or more of the variables depending on the specific process (see details below). Additionally, a '*ModelRunner*' class, that takes in the process classes by injection and is responsible for: (a) storing the variable values before and recalculating after each computation (2) running all calculations based on *dtemp* and *dtime* values provided by the user (3) store the variables values at each stage for plotting. Optionally the model runner could have a '*set_initial_conditions()*' and a '*set_params()*' methods that help prepare the model for prediction. Additional helper functions could be created for result visualisation and export that could become methods of the ModelRunner class if necessary.

This architecture ensures:

- **Modularity**: processes are defined separately and can be easily modified without influencing the others
- **Cohesion**: equations and parameters for each process are stored together
- **Flexibility**: Thanks to the *ModelRunner* class, the model can be easily developed into a standalone microservice, a command line interface or a python package /module to be called by other scripts.

Data input and outputs:

Within the ModelRunner class and between processes inputs and outputs are exchanged as dictionaries. After each computation the model runner collects all outputs from each process and combines them together to obtain the latest state of the system. This is also appended to a separate list of dictionaries called history.

The computation can be launched with a single (*dtime*, *dtemp*) tuple or as a series of values for example from a pandas dataframe. History can be easily trtansformed in a dataframe as well, to show the evolution of the system across a timespan, while the final values of the variables can be accessed from the *ModelRunner* attributes directly.

Visualisation can be created with any dataviz tool desired, since list of dictionaries (that is JSON file) and dataframes are largely used by most libraries.

Results can also be exported in JSON format for later analysis