For advantages, we identified 6 repeated themes in participants' qualitative responses. The themes are:

Bugs can be detected early on.

During DL model training, abnormal behavior can arise from issues with the data or incorrect hyperparameters defined during the model's design. Therefore, a mock implementation can test assertions post-implementation and before the training, increasing the likelihood of catching the bugs at the correct stage. Some participants noticed this benefit and mentioned: *"Advantage is able to have a focused workflow to debug specific issues at their inducing stage"* (P2); *"I found some pretty basic mistakes that developers can make can be caught early on"* (P8); *"Error can be detected early on and the quality of the code can be improved before the actual training process"* (P20);*"Testing each component results in improved code and reduces the efforts required later on after integration"* (P23); *"Better handling of each component and immediate feedback on the code"*(P29); *"Common issues are detected early and developer can get actionable feedback on the code during development"*(P36).

Makes testing easy/easier to manage.

Decomposing each functionality in a DL program into smaller, specialized classes facilitates easier understanding, testing, and modification. Some participants pointed out that: *"Easier to Manage, Clean Code, Extensible, Reusable, Flexible"* (P9); *"Maintaining the software becomes easier—scalability, which assists in the scaling part of the project"* (P10); *"It's similar to the separation of concerns principle, it can be very helpful separating the work which could help identify bugs/mistakes easily"* (P11); *"Better handling of processes and narrowing down of errors"* (P13); *"It simplifies the debugging process and developers can detect the exact point of failures"* (P26); *"Structured code and loosely coupled components that are easier to test and maintain"* (P28); *"Clean code which is easy to maintain, test, debug and reuse"* (P31); *"Easy to locate issues. Very helpful for junior developers, as preprocessing steps such as class balancing are often overlooked, despite their significant impact on unseen data. Additionally, during code reviews, I've observed that while preprocessing steps are present, an incorrect DataFrame or Numpy array is sometimes used to train the model thereby leading to crashes or performance related issues observed during training. Detecting such issues early on is crucial"*(P33).

Time efficient/saves a lot of time.

In DL programs, feature engineering and hyperparameter tuning are time-consuming processes and are often interconnected. Participants remarked that using mocks was a significant step in separating these processes, leading to savings in overall development time: *"Modular development would be extremely helpful in developing large-scale deep-learning systems which involve plenty of unstructured data. Both the data structuring and model building which include tuning hyper-parameters is time consuming process. Some degree of independence between these tasks holds great promise"* (P4); *"It can helpful in quickly identifying the problems and thus help in the rapid development of correct model"* (P7); "*The advantages are saving time, getting actionable feedback on the code base"* (P17).

Automation reduces human efforts.

By incorporating automation into the testing process, developers can streamline their workflow and minimize manual intervention: *"Automatically generating mock data or model can significantly reduce the human effort required for testing components of a deep learning program, leading to more reliable testing. The common practice is to begin with a simpler model and gradually increase its complexity based on performance. However, junior developers often struggle and make mistakes when designing the simpler models, making it time-consuming to identify the actual cause of abnormal behavior"* (P21); *"Timely feedback on code, automation reduces human efforts and makes testing easy"* (P22); *"Helpful for narrowing down of errors in its inducing stage and automation reduces the efforts required for unit testing"* (P24).

Saving resources.

Loading original data and fine-tuning DL model hyperparameters is a resource-intensive process that incurs high computational costs. Some participants shed light on this and highlighted that the effectiveness of selected features can be verified with the mock model at a lower cost. Additionally, mock data also allowed for quickly iterating over various model configurations without incurring the computational cost of using the original dataset.: *"This has been a positive experience for me coming from years of experience in developing deep learning systems. It helped me avoid "usually unnoticed" issues while developing the model for the task with much cheaper cost. Had this problem persisted with real data whose scale could be a lot higher, it would have been much costly venture"* (P4); *"Able to find common bugs without worrying about training it on large infrastructure"* (P9); *"Testing on small infrastructure and ensuring correctness can help in saving resources"* (P22).

Great experience/helpful/useful.

Participants appreciated the idea of mock testing and found it helpful: *"Extremely helpful for organization of code"* (P1); *"I believe it's a great prototype of the process on unit testing"* (P5); *"Good for junior developers"* (P14); *"The overall experience of the unit testing was very good and straight forward"* (P19); *"I liked the idea and believe it has great potential for early bug detection"* (P21); *"Good for identifying common mistakes"* (P23); *"Helpful in Data Security"* (P25); *"Unit testing is always a good practice in traditional software development. I have never tried unit testing for DL software before. It was a great experience for me and I think the issues that lead to silent bugs during training can be reduced using mocks"* (P30); *"Unit testing using mocks is very helpful for verifying the correct input is provided to the model"*(P33).

For disadvantages, we identified 2 repeated themes in participants' qualitative responses. The themes are:

Implementation in industry could be challenging/overhead to setup.

Participants expressed concerns regarding the instrumentation overhead and its practical applicability: *"A disadvantage is that there is more overhead to setup best practices, but the idea is that time is saved in the long-term at the expense of the short-term"* (P2); *"The disadvantage could be the instrumentalization process which if costly may dissuade its adoption in practice"* (P4); *"One of the disadvantages of modular development is keeping track of the versioning and the integration side of the different tasks"* (P10); *"In practice, the overhead involved in setting up can be significant"* (P18); *"Not sure about the scalability of the approach and efforts required in setup"* (P23); *"Proper coordination is required between data and model designing teams to ensure that changes in one module do not negatively impact others"* (P31).

Incorrect reports/false alarms.

Participants also observed that, as with unit testing, there is always a risk of errors in mocks, potentially causing false alarms: *"The drawback could be incorrect reports"* (P17); *"Drawback could be false alarms on mocks which actually does not occur on original data or model"* (P22); *"There is a possibility of incorrect alarms during unit testing as DL applications are data dependent"* (P35).