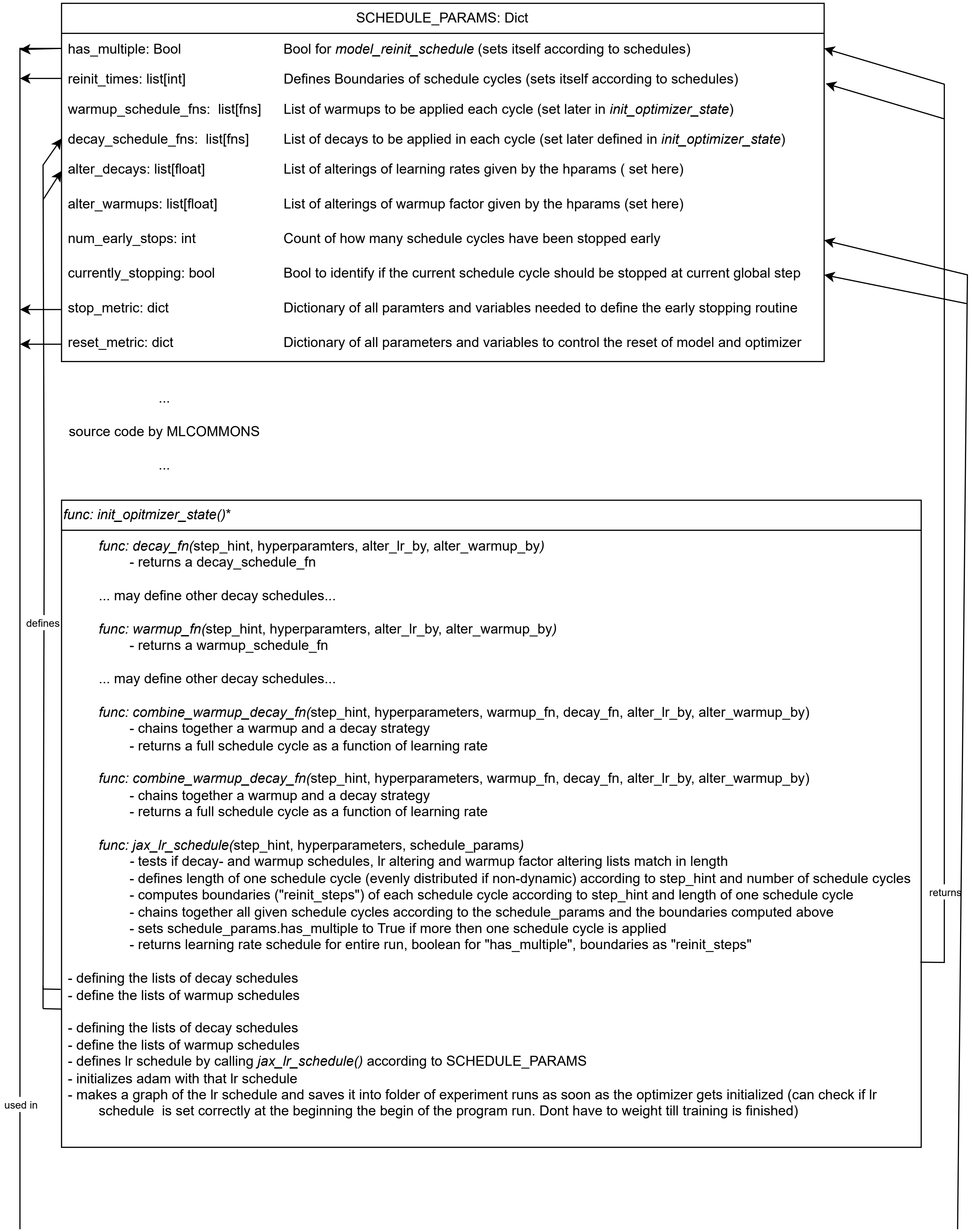


SCHEDULE_PARAMS: Dict

has_multiple: Bool	Bool for *model_reinit_schedule* (sets itself according to schedules)
reinit_times: list[int]	Defines Boundaries of schedule cycles (sets itself according to schedules)
warmup_schedule_fns: list[fns]	List of warmups to be applied each cycle (set later in *init_optimizer_state*)
decay_schedule_fns: list[fns]	List of decays to be applied in each cycle (set later defined in *init_optimizer_state*)
alter_decays: list[float]	List of alterings of learning rates given by the hparams (set here)
alter_warmups: list[float]	List of alterings of warmup factor given by the hparams (set here)
num_early_stops: int	Count of how many schedule cycles have been stopped early
currently_stopping: bool	Bool to identify if the current schedule cycle should be stopped at current global step
stop_metric: dict	Dictionary of all paramters and variables needed to define the early stopping routine
reset_metric: dict	Dictionary of all parameters and variables to control the reset of model and optimizer

...

source code by MLCOMMONS

...

*func: init_opitmizer_state()**

*func: decay_fn(*step_hint, hyperparamters, alter_lr_by, alter_warmup_by*)*
- returns a decay_schedule_fn

... may define other decay schedules...

*func: warmup_fn(*step_hint, hyperparamters, alter_lr_by, alter_warmup_by*)*
- returns a warmup_schedule_fn

... may define other decay schedules...

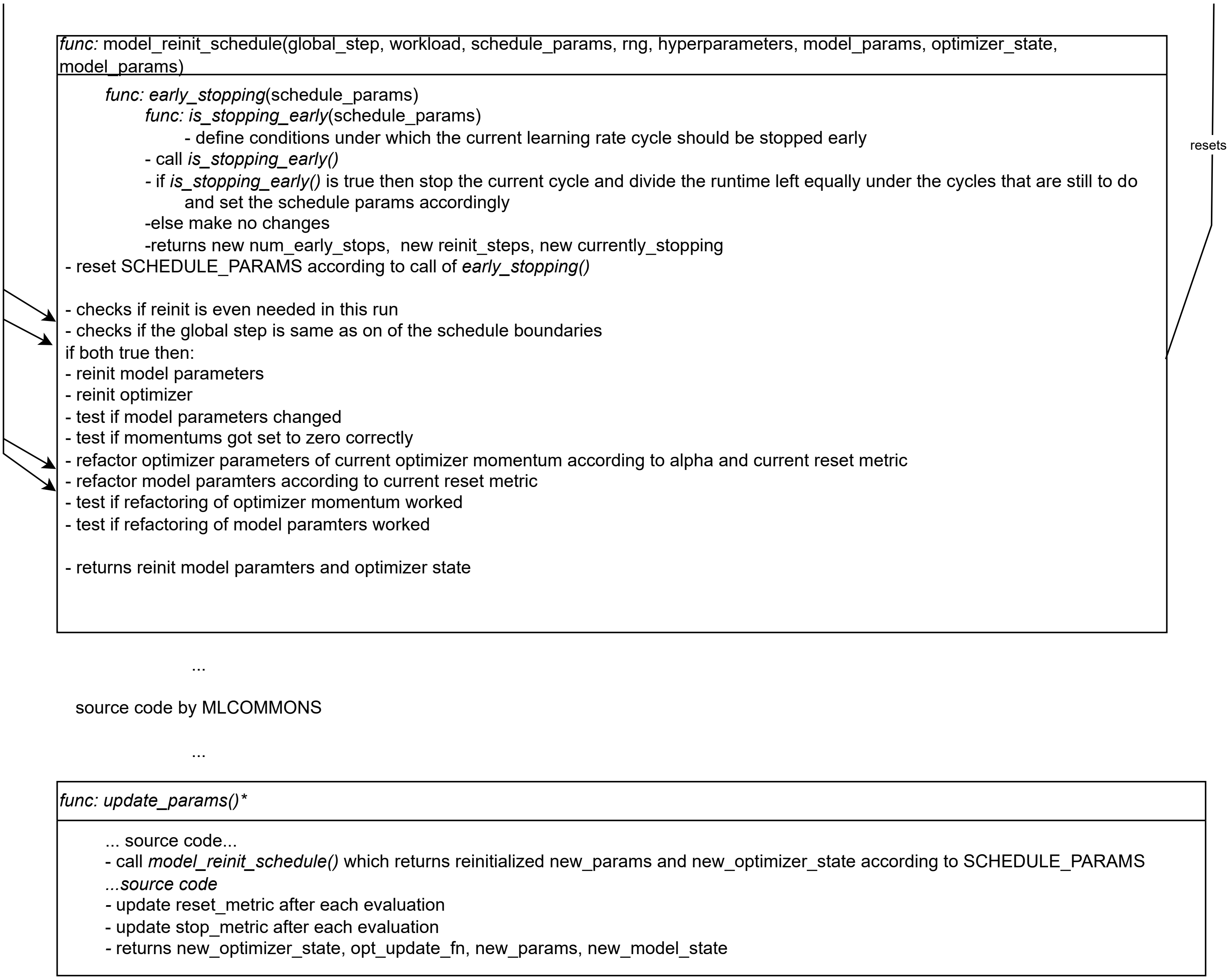
*func: combine_warmup_decay_fn(*step_hint, hyperparameters, warmup_fn, decay_fn, alter_lr_by, alter_warmup_by)
- chains together a warmup and a decay strategy
- returns a full schedule cycle as a function of learning rate

*func: combine_warmup_decay_fn(*step_hint, hyperparameters, warmup_fn, decay_fn, alter_lr_by, alter_warmup_by)
- chains together a warmup and a decay strategy
- returns a full schedule cycle as a function of learning rate

*func: jax_lr_schedule(*step_hint, hyperparameters, schedule_params*)*
- tests if decay- and warmup schedules, lr altering and warmup factor altering lists match in length
- defines length of one schedule cycle (evenly distributed if non-dynamic) according to step_hint and number of schedule cycles
- computes boundaries ("reinit_steps") of each schedule cycle according to step_hint and length of one schedule cycle
- chains together all given schedule cycles according to the schedule_params and the boundaries computed above
- sets schedule_params.has_multiple to True if more then one schedule cycle is applied
- returns learning rate schedule for entire run, boolean for "has_multiple", boundaries as "reinit_steps"

- defining the lists of decay schedules
- define the lists of warmup schedules

- defining the lists of decay schedules
- define the lists of warmup schedules
- defines lr schedule by calling *jax_lr_schedule()* according to SCHEDULE_PARAMS
- initializes adam with that lr schedule
- makes a graph of the lr schedule and saves it into folder of experiment runs as soon as the optimizer gets initialized (can check if lr schedule is set correctly at the beginning the begin of the program run. Dont have to weight till training is finished)



func: model_reinit_schedule(global_step, workload, schedule_params, rng, hyperparameters, model_params, optimizer_state, model_params)

```
        func: early_stopping(schedule_params)
                func: is_stopping_early(schedule_params)
                        - define conditions under which the current learning rate cycle should be stopped early
                - call is_stopping_early()
                - if is_stopping_early() is true then stop the current cycle and divide the runtime left equally under the cycles that are still to do
                        and set the schedule params accordingly
                -else make no changes
                -returns new num_early_stops,  new reinit_steps, new currently_stopping
- reset SCHEDULE_PARAMS according to call of early_stopping()

- checks if reinit is even needed in this run
- checks if the global step is same as on of the schedule boundaries
if both true then:
- reinit model parameters
- reinit optimizer
- test if model parameters changed
- test if momentums got set to zero correctly
- refactor optimizer parameters of current optimizer momentum according to alpha and current reset metric
- refactor model paramters according to current reset metric
- test if refactoring of optimizer momentum worked
- test if refactoring of model paramters worked

- returns reinit model paramters and optimizer state
```

...

source code by MLCOMMONS

...

*func: update_params()**

```
        ... source code...
        - call model_reinit_schedule() which returns reinitialized new_params and new_optimizer_state according to SCHEDULE_PARAMS
        ...source code
        - update reset_metric after each evaluation
        - update stop_metric after each evaluation
        - returns new_optimizer_state, opt_update_fn, new_params, new_model_state
```