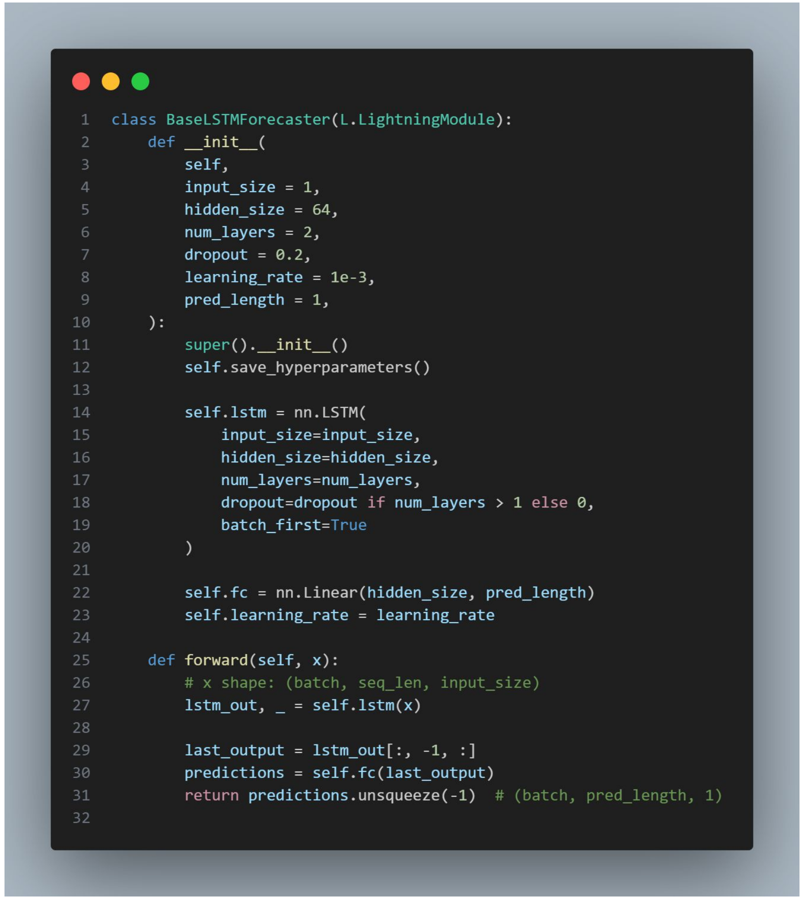
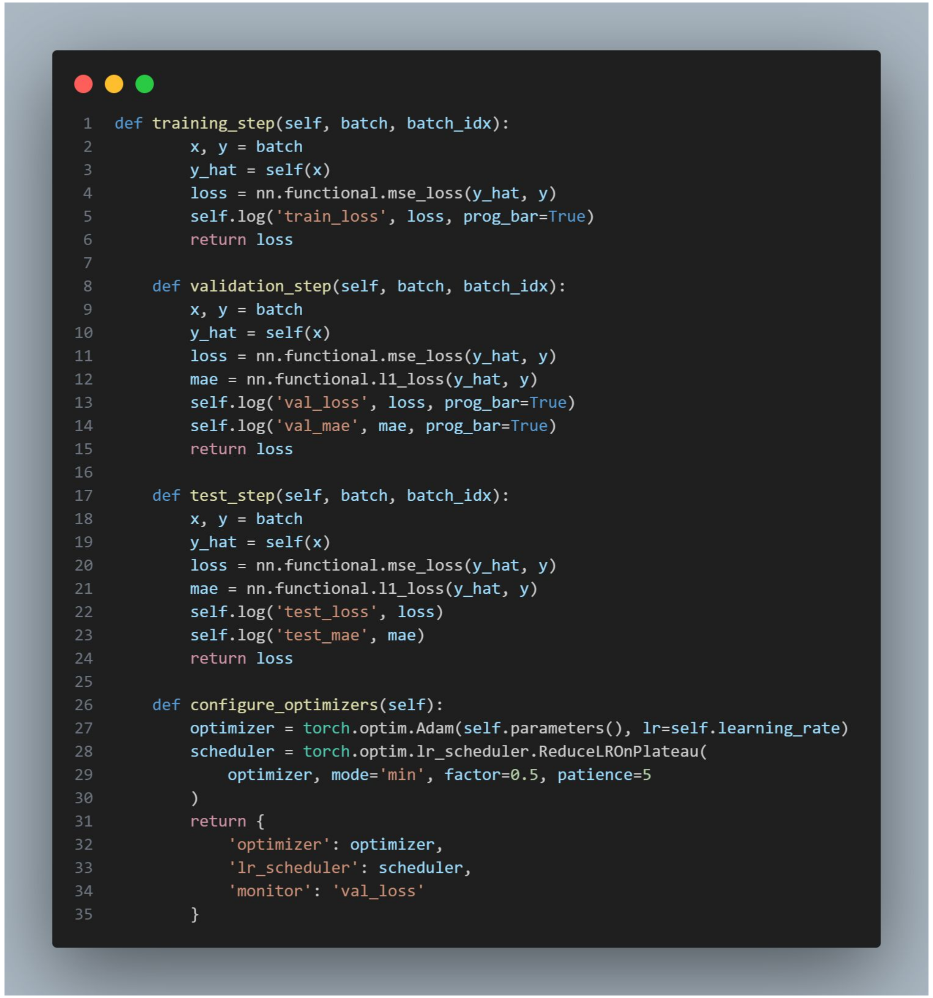
MLOps 1

Bartłomiej Pukacki 151942

Lightning Module

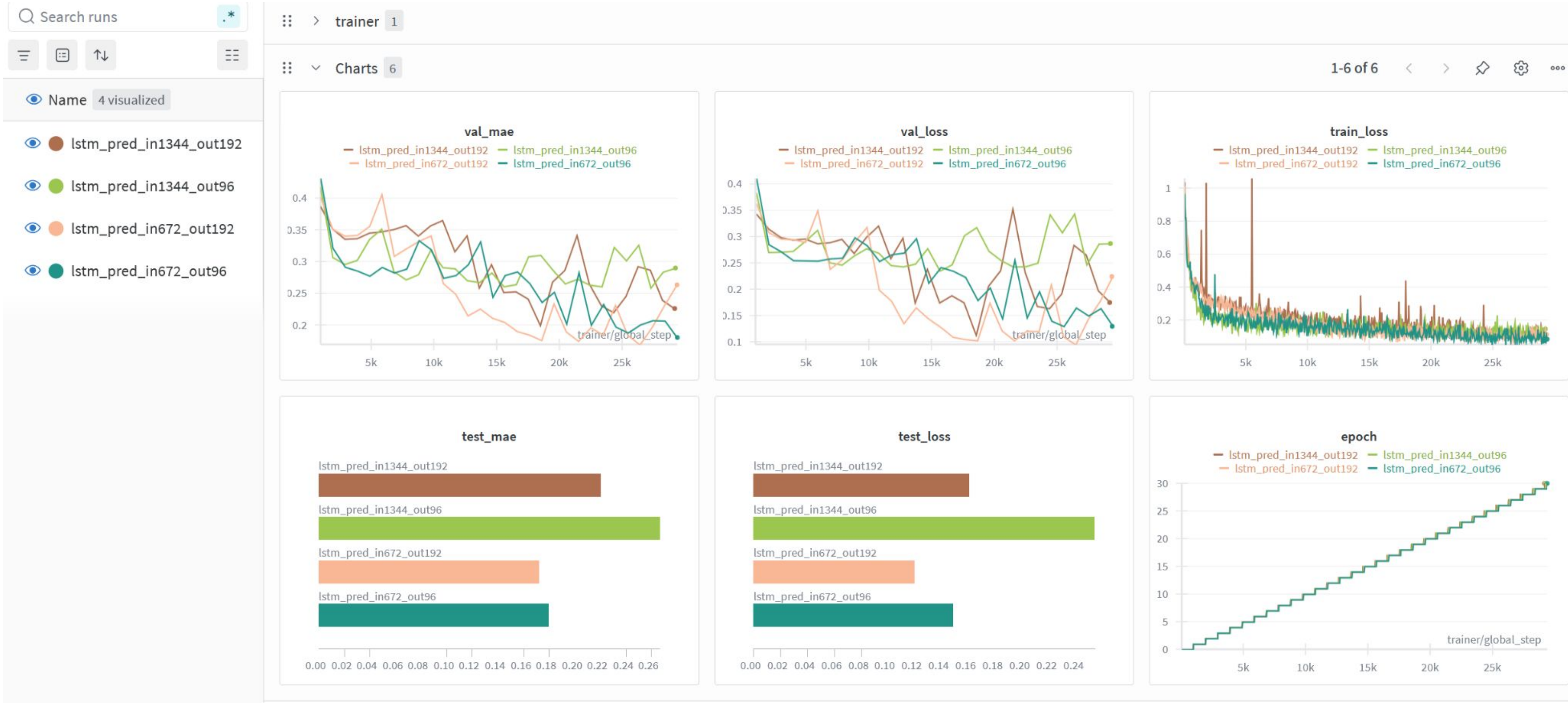


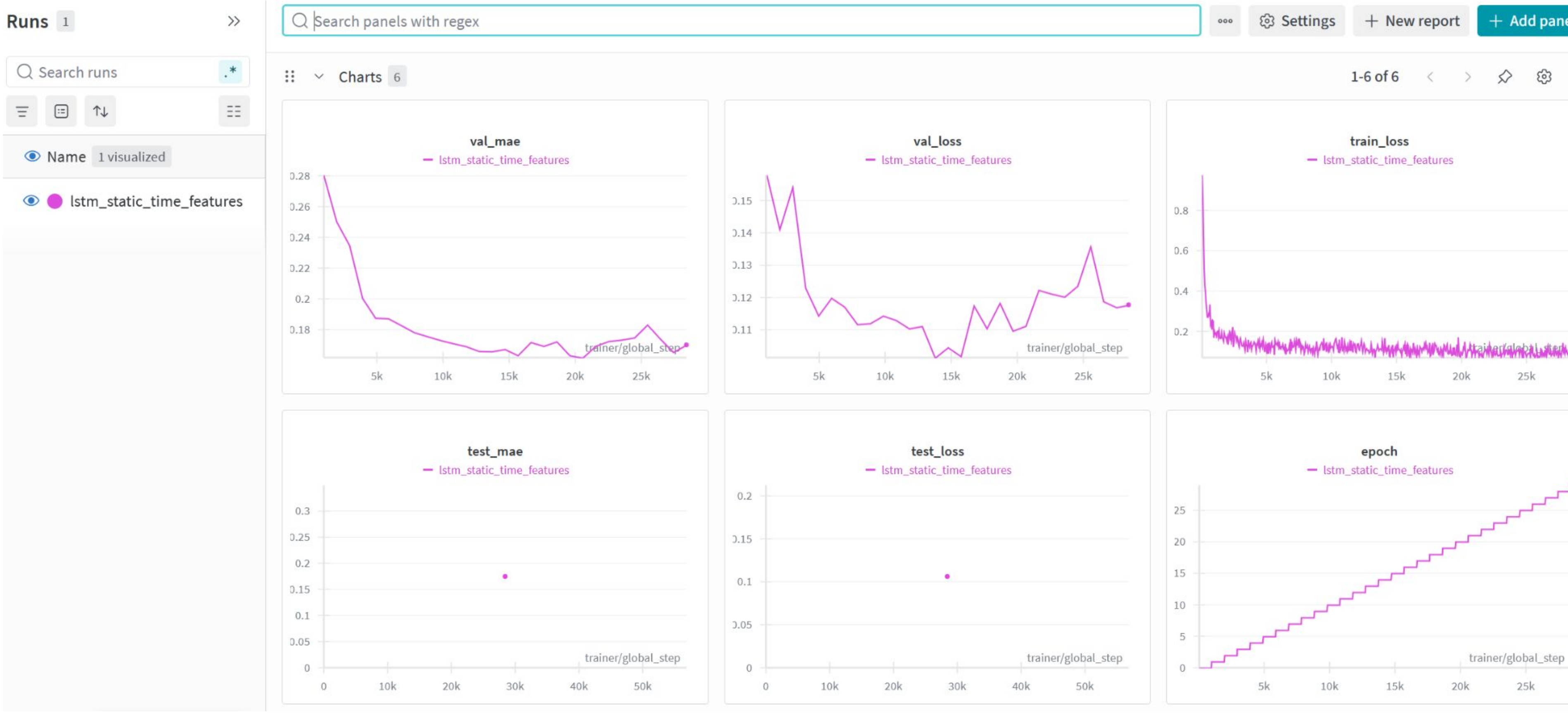
```python
class BaseLSTMForecaster(L.LightningModule):
    def __init__(
        self,
        input_size = 1,
        hidden_size = 64,
        num_layers = 2,
        dropout = 0.2,
        learning_rate = 1e-3,
        pred_length = 1,
    ):
        super().__init__()
        self.save_hyperparameters()

        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            dropout=dropout if num_layers > 1 else 0,
            batch_first=True
        )

        self.fc = nn.Linear(hidden_size, pred_length)
        self.learning_rate = learning_rate

    def forward(self, x):
        # x shape: (batch, seq_len, input_size)
        lstm_out, _ = self.lstm(x)

        last_output = lstm_out[:, -1, :]
        predictions = self.fc(last_output)
        return predictions.unsqueeze(-1)  # (batch, pred_length, 1)

```



```python
def training_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self(x)
        loss = nn.functional.mse_loss(y_hat, y)
        self.log('train_loss', loss, prog_bar=True)
        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self(x)
        loss = nn.functional.mse_loss(y_hat, y)
        mae = nn.functional.l1_loss(y_hat, y)
        self.log('val_loss', loss, prog_bar=True)
        self.log('val_mae', mae, prog_bar=True)
        return loss

    def test_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self(x)
        loss = nn.functional.mse_loss(y_hat, y)
        mae = nn.functional.l1_loss(y_hat, y)
        self.log('test_loss', loss)
        self.log('test_mae', mae)
        return loss

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=self.learning_rate)
        scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
            optimizer, mode='min', factor=0.5, patience=5
        )
        return {
            'optimizer': optimizer,
            'lr_scheduler': scheduler,
            'monitor': 'val_loss'
        }
```

Wandb Results

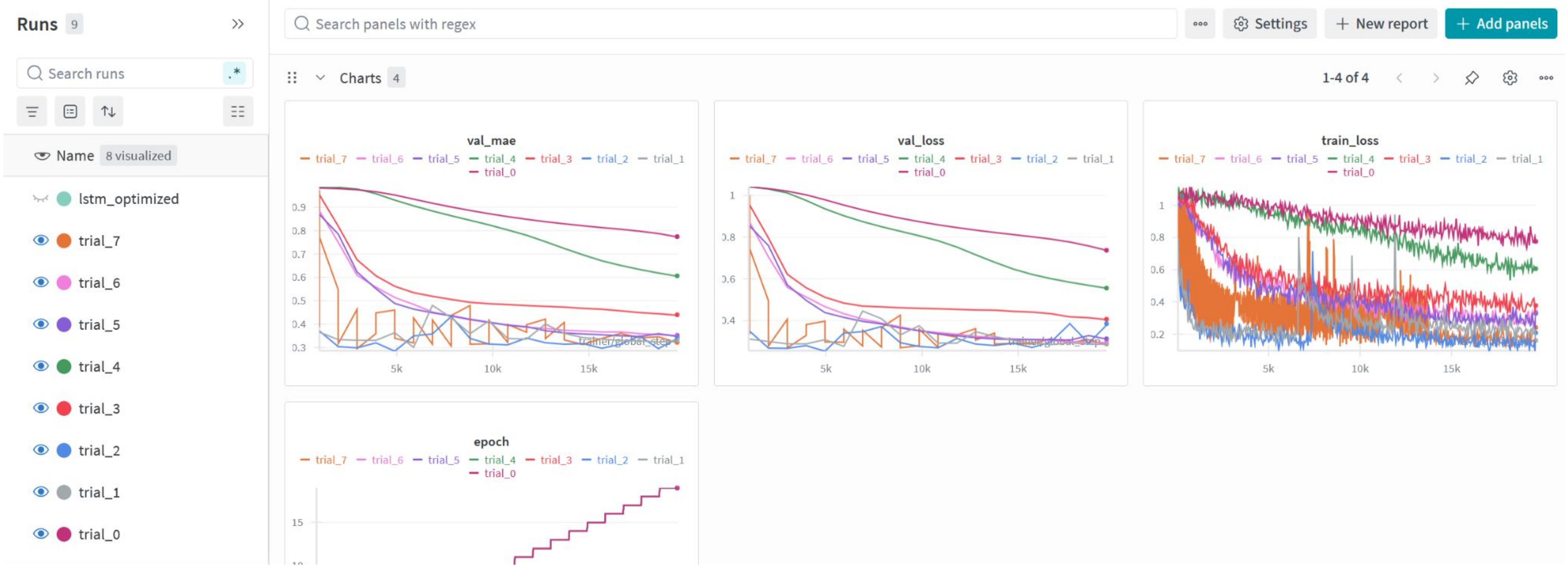


Runs 1
Search runs
Name 1 visualized
lstm_static_time_features
Search panels with regex
Settings
New report
Add pane
Charts 6
1-6 of 6

Optuna Results

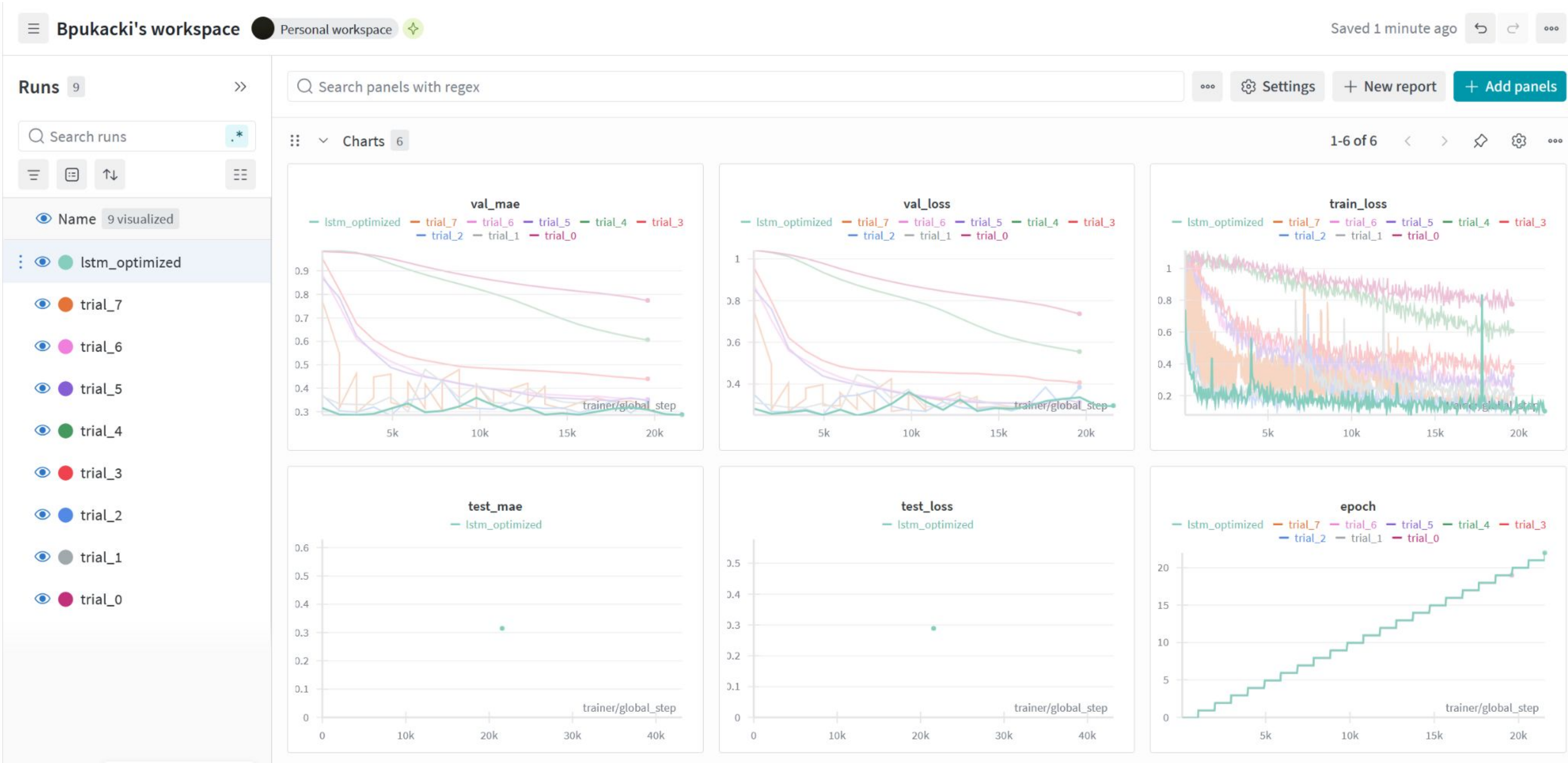
Optimization settings:

- Max 20 epochs, 10 trials (due to time constraints)
- Learning rate: trial.suggest_loguniform("learning_rate", 1e-5, 1e-2)
- LSTM hidden size: trial.suggest_int("hidden_size", 8, 32, step=4)

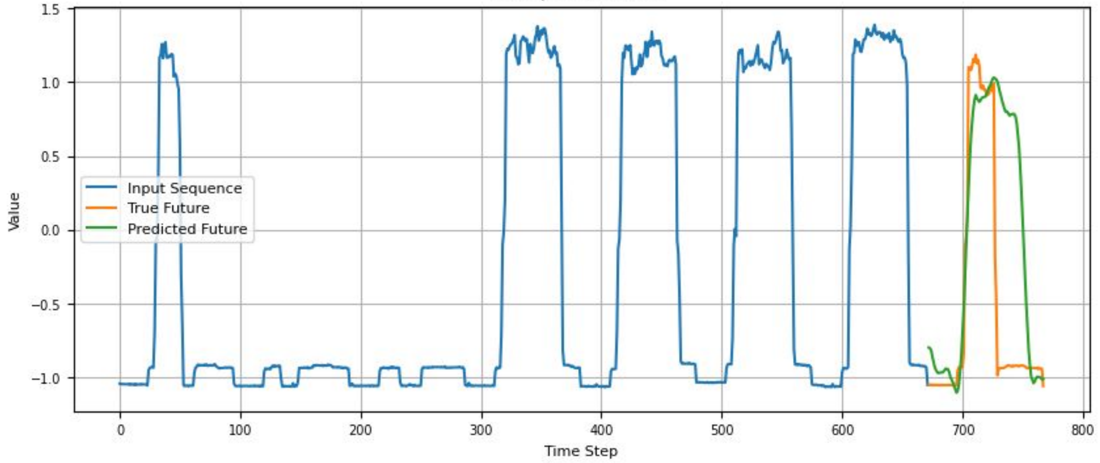


Note: only 8/10 trials are saved for some reason

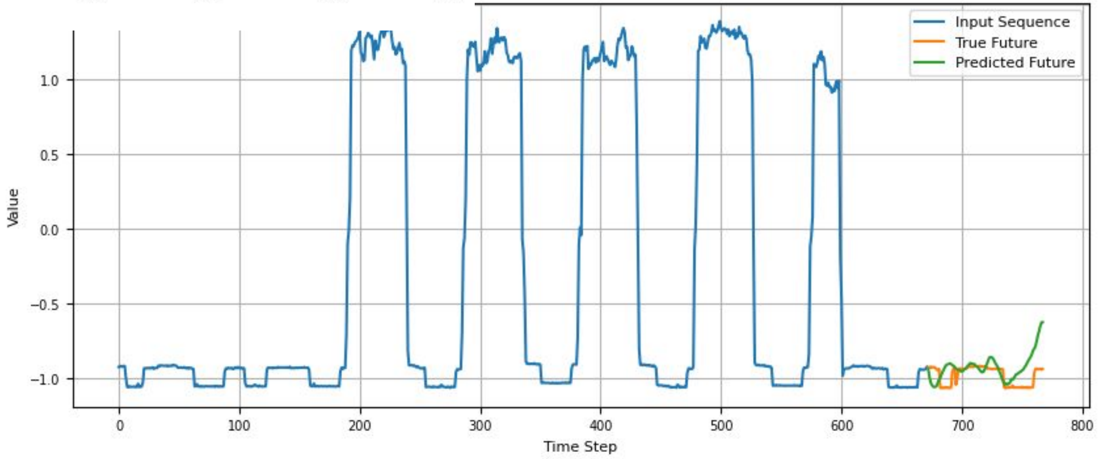
Best: HS=16, LR=0.005 (val_loss=0.288, test_loss=0.289)



Example Prediction 1



Example Prediction 2



Encountered Problems

- Issues with optimizing training for GPU:
  - Num_workers > 0 in dataloaders would result in very long startup
  - Wasn’t sure whether lightning correctly utilizes GPU - I had to double check in the same way I would if I used plain torch although lightning did work in the first place
- Optuna takes a lot of time on top of model which already takes a while to train
  - Reduced max epochs
  - Reduced num of trials
  - Reduced max hidden size
  - Tested only two parameters
  - Still took 1.5h (would take two days at least with preferred settings)
  - Should’ve logged tested hyperparams in wandb
- Wandb didn’t save all optuna trials
- Using knowledge extraction projects might be a mistake - it would take less time to set up and play around with a simple model for MNIST :)