Wav2Vec2 model finetuning is incomplete hence no observations between finetuned model and that of the pretrained model is made.

Further improvements that can be done to the experiment is evaluating and cleaning the dataset to ensure a more balanced and representative dataset of real-world application needs. Identify issues such as accents or noise levels that impact performance. If needed, add data from related datasets (e.g., Common Voice or LibriSpeech) to address gaps, and assess each dataset's impact on accuracy after inclusion.

Use data augmentation techniques like time-stretching, pitch-shifting, background noise injection, and volume perturbation to make the model more robust to real-world variations. Evaluate these augmentations individually and in combination to determine the most effective setup.

In preprocessing, consider noise reduction (e.g., Spectral-Gating) if target environment is noisy, and compare model accuracy with and without noise reduction.

Hyperparameters and metrics like learning rate, batch size, number of epochs, and weight decay can be optimized. Use grid or random search to find optimal settings, and test different loss functions (e.g., Cross-Entropy, CTC, or MSE) to determine which can attain the best performace. Similarly, try out different optimizers like AdamW, RMSprop, or SGD to find one that maximizes convergence and minimizes loss. Add a weight decay parameter to control overfitting, especially with optimizers like AdamW, and set a reasonable maximum number of epochs to avoid overtraining.

To ensure convergence and mitigate overfitting, experiment with learning rate schedulers like cosine annealing or exponential decay. Additionally, use gradient accumulation to simulate larger batch sizes if your hardware limits batch size, testing various accumulation steps to optimize generalization without memory issues.

Experiment with model architecture by freezing or reinitializing certain Wav2Vec2 layers to find the right balance of transfer learning and task-specific tuning. If beneficial, add a custom classification head (e.g., LSTM, GRU, or Transformer layers) to capture complex patterns in your data.

When evaluating, create a domain-specific test set reflecting real-world noise, accents, or vocabulary, and test regularly on this set to ensure generalization. Use k-fold cross-validation to enhance robustness and prevent overfitting.

Conduct an error analysis and apply post-processing. Identify common error types (e.g., numbers, names, or homophones) and monitor error trends to ensure continuous improvement. Finally, consider applying a language model or sequence-to-sequence model as a post-processing step to correct grammar or word errors, which can further improve accuracy.