Scalability Analysis

First, IWUkNN employs a fixed and small number of iterations (five), which strictly bounds the worst-case computational overhead to a constant factor over standard kNN. Moreover, repeated neighbor searches are performed only for the subset of test samples that remain misclassified in each iteration. Empirically, this subset shrinks rapidly after the first iteration, which significantly reduces the effective computational cost even for large datasets.

For very large-scale datasets with millions of instances, IWUkNN inherits the same scalability challenges as standard kNN, since distance computation remains the dominant cost. In such settings, the algorithm can directly benefit from approximate nearest neighbor (ANN) search methods, such as KD-trees, ball trees, locality-sensitive hashing, or graph-based ANN techniques. Because IWUkNN does not alter the distance metric or neighborhood definition, these ANN methods can be seamlessly integrated without modifying the core algorithm.

Importantly, the iterative weight update mechanism in IWUkNN operates independently of how neighbors are retrieved. Therefore, approximate neighbor searches only affect the neighbor identification step and do not interfere with the weight update or convergence behavior. This makes IWUkNN well-suited for scalable implementations using modern ANN libraries, parallel processing, or GPU acceleration.

In summary, while IWUkNN is more computationally intensive than standard kNN, its fixed iteration count, rapidly shrinking misclassified set, and compatibility with approximate nearest neighbor methods enable it to scale effectively to very large datasets when combined with appropriate indexing and acceleration techniques.

Experimentally, we run all models on MANIST with considering scalability issue as given in the next Tables. To address the concern regarding the stability of the proposed methods across varying dataset sizes and class complexities, we conducted a controlled evaluation using the MNIST handwritten digit dataset. MNIST provides a natural framework for such analysis, as it allows progressive inclusion of classes while maintaining consistent feature space and data characteristics.

The experiment was designed as follows:

- We began with a binary classification task using digits {0,1}.
- Next, we extended to a three-class problem with digits {0,1,2}.
- We continued this process incrementally, adding one class at a time, until the full 10-class classification task {0–9} was included.

For each stage (2 classes, 3 classes, …, 10 classes), we trained and tested all methods under identical conditions, ensuring fairness of comparison. The classification accuracy was recorded at each stage, thereby providing a clear picture of how performance evolves as both the number of samples and the number of classes increase.

This incremental design offers two key advantages:

- It directly evaluates the scalability and stability of the models as data size and complexity grow.
- It ensures that any observed performance trends are attributable to the increasing classification challenge rather than changes in data source or domain.

The results demonstrate that our proposed methods (LMSL-FkNN and CMDW-FkNN) maintain strong and consistent performance throughout this progression, confirming their robustness and stability with respect to dataset size and class expansion.

The results attest that our proposed IWUkNN and LRkNN have the most similar trend, despite that IWUkNN comes in the second top rank, across Accuracy and F1 metrics. CDkNN comes in the third position. Overall, no significant differences in performance of other models except for ExNRulekNN which comes at the last position. In terms of ROC, IWUkNN and LMRkNN have almost the same trend. The result gives an impression that this model is promising in terms of scalability. Interestingly, HMAkNN comes to affirm its competency, taking the middle position. Generally speaking, all models are top models in kNN literature and all of them have similar performance in terms of scalability, despite some being better than others slightly.

Table 1 – Proposed IWUkNN vs. Rival kNNs over MANIST Dataset (Accuracy, Mean)

Dataset	**ExNRule**	**HMAKNN**	**LMRkNN**	**LMKHNCN**	**MLM-kHNN**	**CDkNN**	**IWUkNN**
MNIST_14780	0.999	0.999	0.999	0.999	0.999	0.999	0.999
MNIST_21770	0.993	0.992	0.996	0.993	0.993	0.992	0.994
MNIST_28911	0.988	0.991	0.994	0.990	0.990	0.991	0.993
MNIST_35735	0.985	0.991	0.994	0.990	0.990	0.991	0.992
MNIST_42048	0.977	0.986	0.991	0.984	0.984	0.987	0.987
MNIST_48924	0.974	0.985	0.989	0.982	0.982	0.985	0.985
MNIST_56217	0.968	0.979	0.986	0.978	0.978	0.980	0.980
MNIST_63042	0.962	0.975	0.983	0.972	0.972	0.975	0.975
MNIST_70000	0.955	0.969	0.979	0.963	0.963	0.970	0.975
Average	0.977	0.985	0.990	0.983	0.983	0.985	0.987

Table 2 – Proposed IWUkNN vs. Rival kNNs over MANIST Dataset (F1, Mean)

Dataset	**ExNRule**	**HMAKNN**	**LMRkNN**	**LMKHNCN**	**MLM-kHNN**	**CDkNN**	**IWUkNN**
MNIST_14780	0.999	0.999	0.999	0.999	0.999	0.999	0.999
MNIST_21770	0.993	0.992	0.996	0.993	0.993	0.992	0.994
MNIST_28911	0.988	0.991	0.994	0.990	0.990	0.991	0.993
MNIST_35735	0.985	0.991	0.994	0.990	0.990	0.991	0.992
MNIST_42048	0.976	0.986	0.991	0.984	0.984	0.986	0.987
MNIST_48924	0.974	0.984	0.989	0.982	0.982	0.985	0.985
MNIST_56217	0.968	0.980	0.986	0.977	0.977	0.980	0.980
MNIST_63042	0.962	0.975	0.983	0.971	0.971	0.975	0.976
MNIST_70000	0.954	0.969	0.979	0.963	0.963	0.969	0.975
Average	0.977	0.985	0.990	0.983	0.983	0.985	0.987

Table 3 – Proposed IWUkNN vs. Rival kNNs over MANIST Dataset (ROC, Mean)

Dataset	**ExNRule**	**HMAKNN**	**LMRkNN**	**LMKHNCN**	**MLM-kHNN**	**CDkNN**	**IWUkNN**
MNIST_14780	0.999	0.998	0.999	0.999	0.999	0.999	0.999
MNIST_21770	0.995	0.992	0.996	0.995	0.995	0.993	0.996
MNIST_28911	0.992	0.992	0.996	0.993	0.993	0.993	0.995
MNIST_35735	0.990	0.992	0.996	0.993	0.993	0.994	0.995
MNIST_42048	0.986	0.989	0.994	0.990	0.990	0.992	0.992
MNIST_48924	0.985	0.987	0.993	0.989	0.989	0.991	0.991
MNIST_56217	0.982	0.984	0.991	0.987	0.987	0.989	0.989

MNIST_63042	0.976	0.981	0.990	0.984	0.984	0.986	0.986
MNIST_70000	0.969	0.978	0.988	0.978	0.979	0.983	0.986
Average	0.986	0.988	0.993	0.989	0.989	0.991	0.993

We have added all these information into our Github repository with title “Models Scalability”.