

# Optimizing ML Pipelines: An In-Depth Analysis of WindTunnel's Methodology and Applications

A Comprehensive Review

Yongshu Cui

Peng Zhao

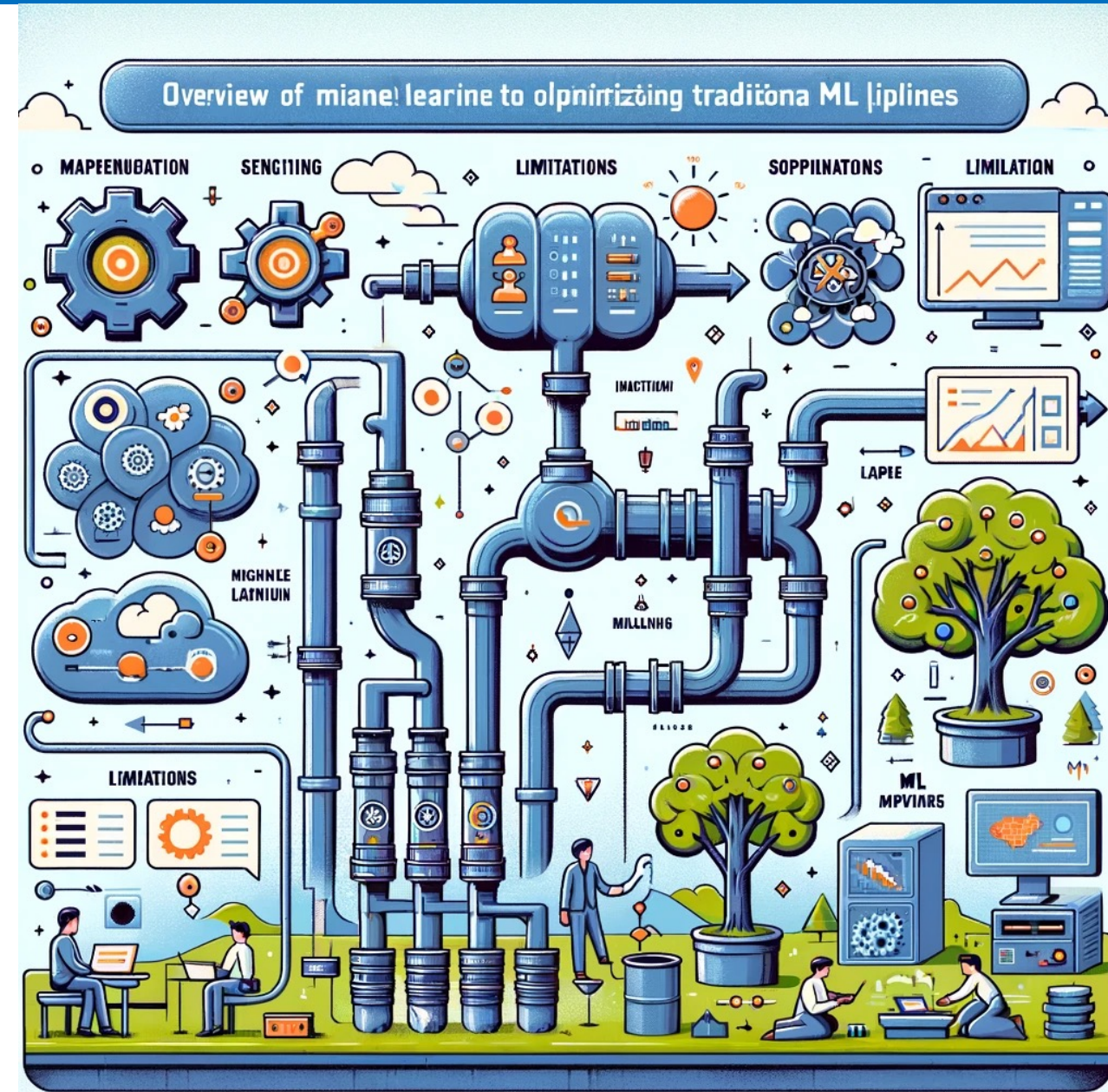
Xiaolin Liu

Ting Yang

2023/11/20

# Introduction

- Overview of machine learning pipelines and their importance in various fields.
- Limitations of traditional ML pipelines.
- The need for optimizing ML pipelines

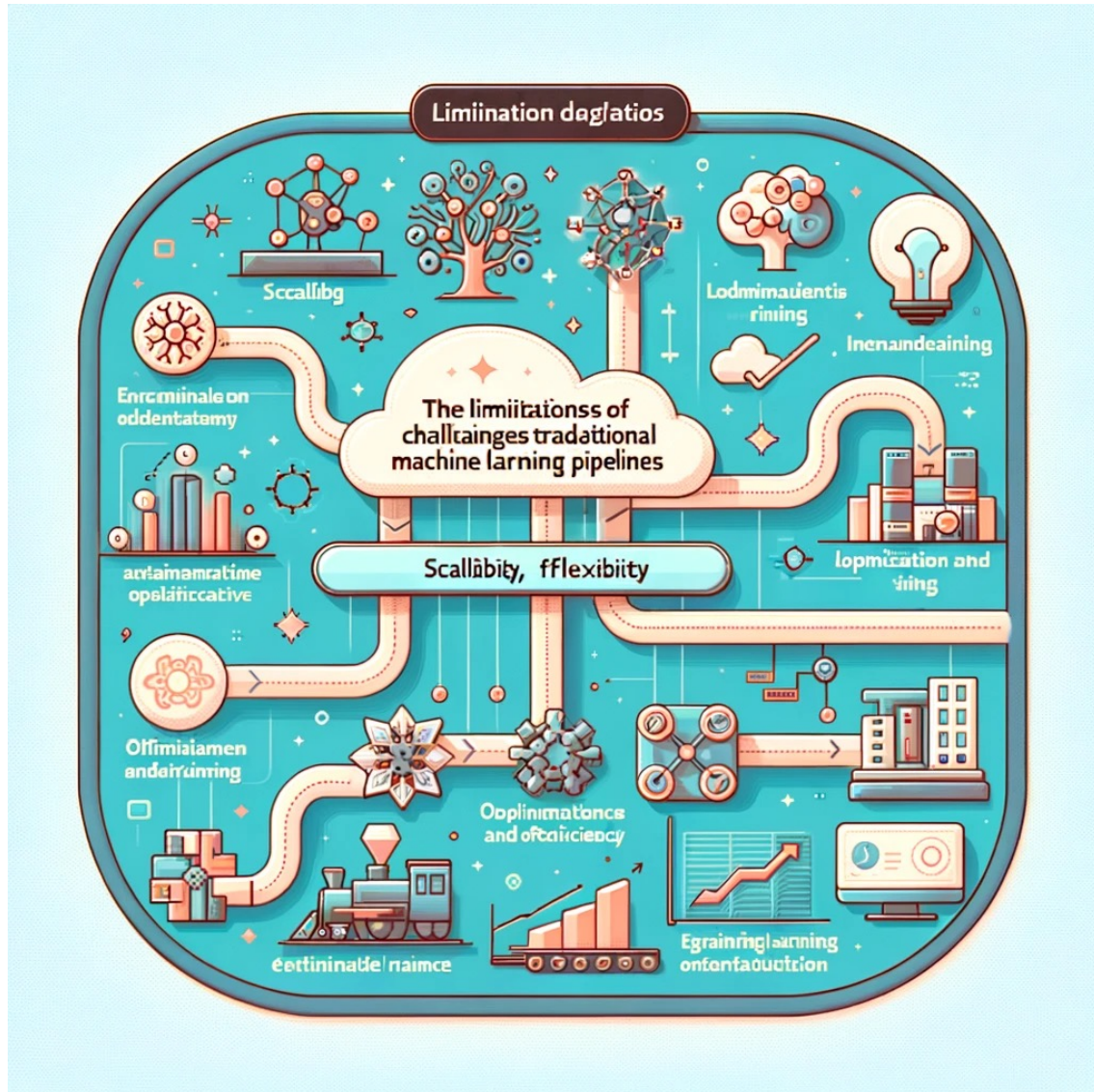


# WindTunnel: The Proposed Solution

- Introduction to the WindTunnel method.
- Key ideas: Layered differentiable programming and differentiable pipeline connections.
- The goal of achieving differentiable ML pipelines.



# Problem Statement

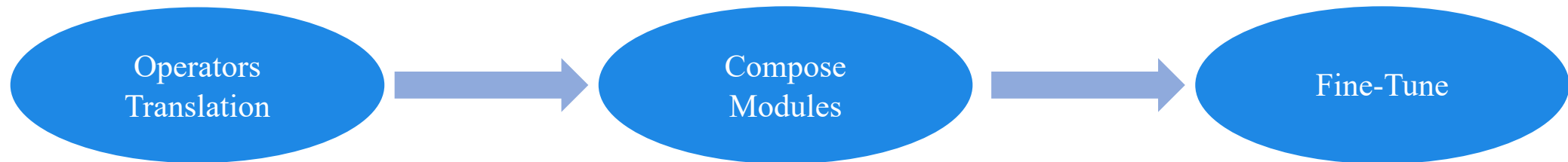


1. The challenges with classical ML pipelines.

2. The gap between ML and Deep Neural Networks (DNNs) in terms of optimization and training.

# Methodology of WindTunnel

- Transforming ML pipeline components into neural network modules.
- Differentiating between arithmetic and algorithmic operators.
- Fine-tuning process and global optimization strategy.of optimization and training.



# WindTunnel's currently support ML operators

Linear models

Normalizers

categorical  
encoders

SVM

PCA

LDA

Kmeans

naive bayes

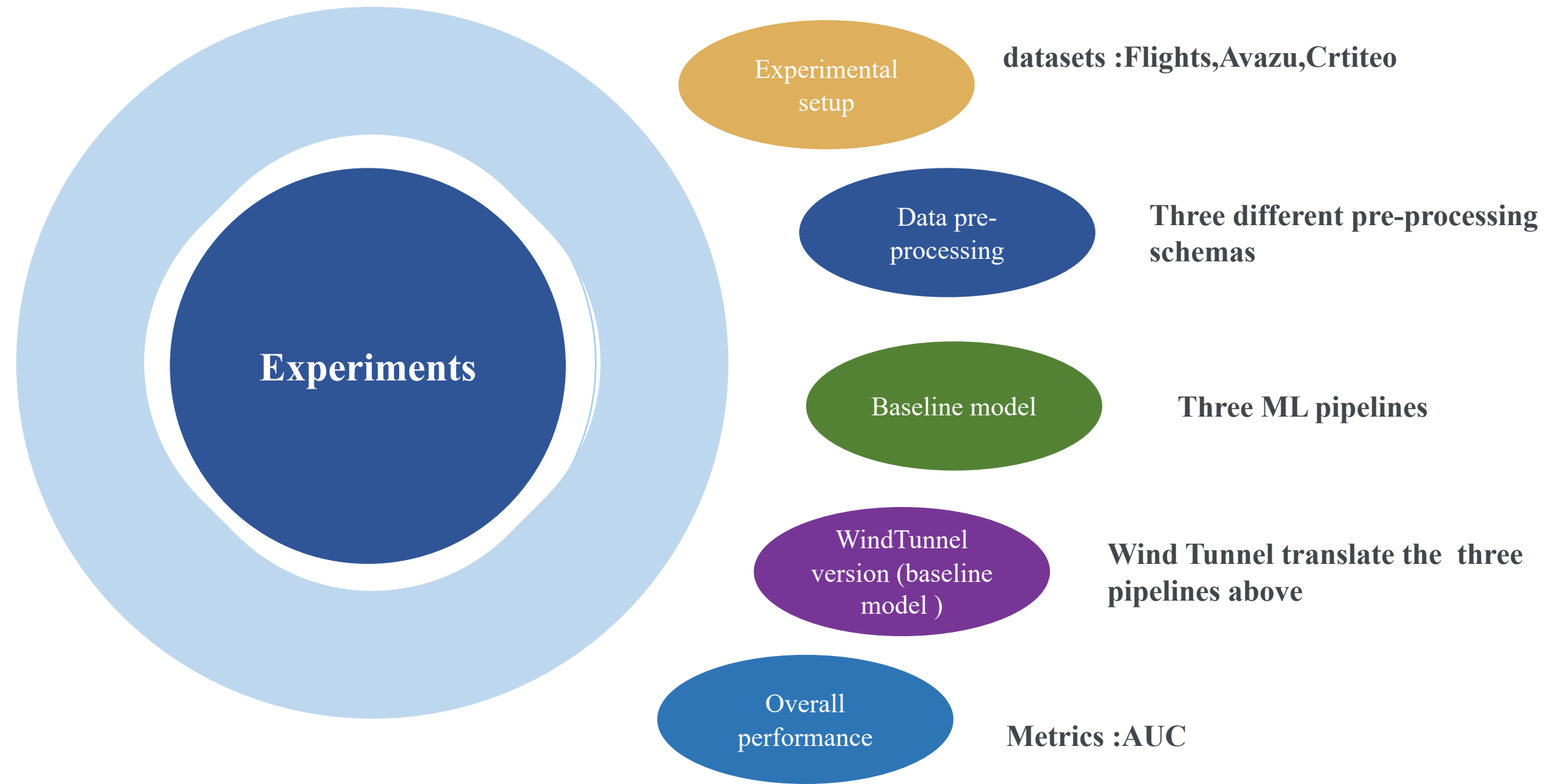
random forest

gradient  
boosting tress

matrix  
factorization

factorization  
machine

# Details inside the Wind Tunnel



# Experimental setup

## Flight

Flight dataset is utilized to forecast if a planned flight will experience a delay of 15 minutes or more

## Avazu

Avazu dataset on Kaggle is used for click-through rate prediction in online advertising.

## Criteo

Criteo's Kaggle dataset is designed for predicting ad click-through rates based on user features.

## Train /Test

For Flight, 64.8% / 35.2%  
For Avazu, 90% / 10%  
For Criteo, 90% / 10%



Model	Flight			Avazu			Criteo		
	Pre1	Pre2	Pre3	Pre1	Pre2	Pre3	Pre1	Pre2	Pre3
ML (Pipe1)	0.6783	0.6847	0.7126	0.6896	0.7264	0.7553	0.7167	0.7442	0.7769
ML (Pipe2)	0.7358	0.7427	0.7507	0.7521	0.7550	0.7718	0.7739	0.7781	0.7925
ML (Pipe3)	0.7519	0.7547	0.7467	0.7597	0.7616	0.7728	0.7838	0.7884	0.7954
DeepGBM	0.7793	0.7695	0.7726	0.7682	0.7680	0.7760	0.7965	0.7918	0.7972
W.T. (Pipe1)	0.6913	0.6910	0.7244	0.7588	0.7589	0.7637	0.7750	0.7804	0.7903
W.T. (Pipe2)	0.7790	0.7897	0.7960	<b>0.7753</b>	<b>0.7742</b>	<b>0.7763</b>	0.8006	0.8053	0.8041
W.T. (Pipe3)	<b>0.7829</b>	<b>0.7906</b>	<b>0.7989</b>	0.7746	0.7718	0.7753	<b>0.8014</b>	<b>0.8058</b>	<b>0.8048</b>

Overall performance comparison

We report AUC on test split following the previous work [24]. ML is the original ML pipeline, while W.T. is for WindTunnel. PreX means different preprocessing schemes, and PipeX denotes different ML pipelines. The best result is marked bold.

Model	Pre2	Pre3
ML (Pipe2)	0.7781	0.7925
GBDT2NN (Pipe2)	0.7962	0.7998
W.T. (Pipe2)	0.8053	0.8041

AUC of the Criteo dataset using different translation scopes. AUC of the Criteo dataset using different translation scopes.

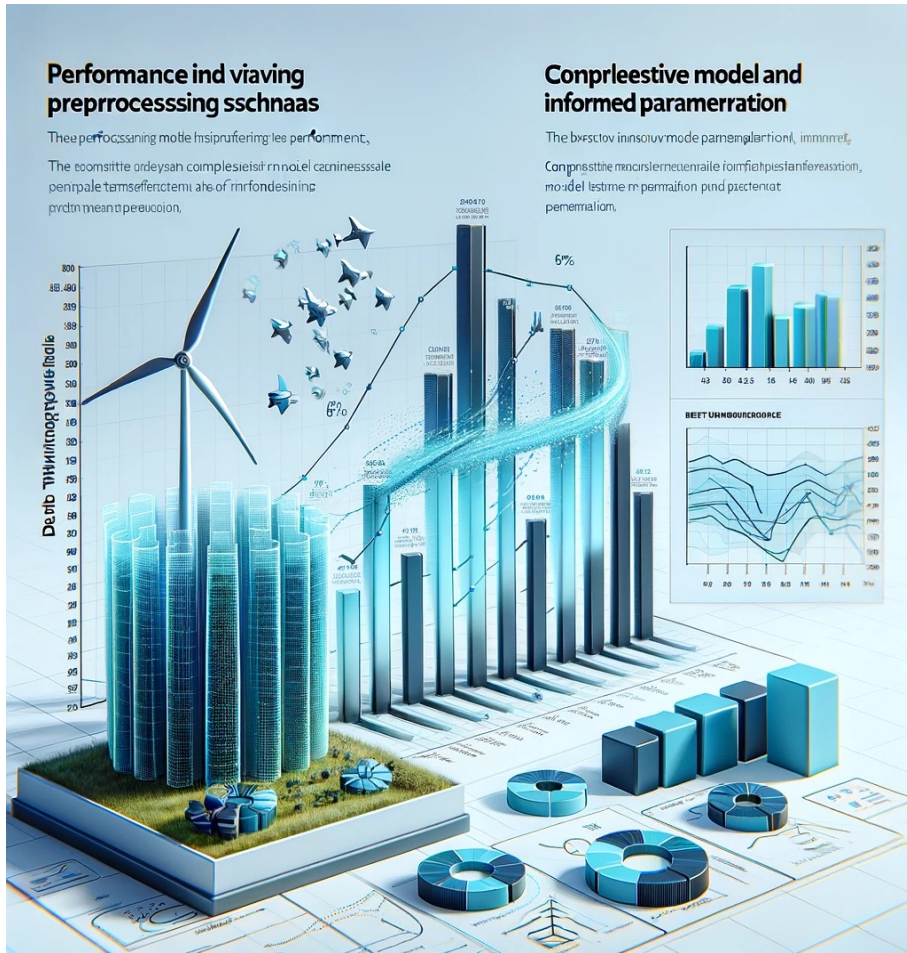
Model	Pre2		Pre3	
	Cold	Warm	Cold	Warm
W.T. (Pipe2)	0.7983	0.8053	0.7990	0.8041
W.T. (Pipe3)	0.7966	0.8058	0.7975	0.8048

AUC of the Criteo dataset using different parameter initialization regimes

Model	100%	10%	1%
ML (Pipe2)	0.7781	0.7777	0.7657
W.T. (Pipe2, L2)	0.7924	0.7874	0.7685
W.T. (Pipe2, L3)	0.8051	0.7912	0.7699
W.T. (Pipe2, L4)	0.8053	0.7906	0.7691

AUC of the Criteo dataset using different GBDT parametrization levels and dataset sizes (1%, 10%, 100%).

# Insights from the Experiments



- Performance improvements achieved with WindTunnel.
- Impact of varying preprocessing schemas on model performance.
- Benefits of comprehensive model translation and informed parameterization.

# The WindTunnel can be useful in production

hummingbird


0.4.9

Search docs

Main website

hummingbird

Welcome to Hummingbird



This is the API documentation for the Python interface.

Entrypoint for Hummingbird modules.

- Main website
- hummingbird
  - hummingbird.ml
    - hummingbird.ml.\_executor
    - hummingbird.ml.\_parse
    - hummingbird.ml.\_topology
    - hummingbird.ml.\_utils
    - hummingbird.ml.convert
    - hummingbird.ml.exceptions
    - hummingbird.ml.supported
    - hummingbird.ml.containers
    - hummingbird.ml.operator\_converters

Next

## Examples

See the [notebooks](#) section for examples that demonstrate use and speedups.

In general, Hummingbird syntax is very intuitive and minimal. To run your traditional ML model on DNN frameworks, you only need to `import hummingbird.ml` and add `convert(model, 'dnn_framework')` to your code. Below is an example using a [scikit-learn random forest](#) model and [PyTorch](#) as target framework.

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from hummingbird.ml import convert, load

# Create some random data for binary classification
num_classes = 2
X = np.random.rand(100000, 28)
y = np.random.randint(num_classes, size=100000)

# Create and train a model (scikit-learn RandomForestClassifier in this case)
skl_model = RandomForestClassifier(n_estimators=10, max_depth=10)
skl_model.fit(X, y)

# Use Hummingbird to convert the model to PyTorch
model = convert(skl_model, 'pytorch')

# Run predictions on CPU
model.predict(X)

# Run predictions on GPU
model.to('cuda')
model.predict(X)

# Save the model
model.save('hb_model')

# Load the model back
model = load('hb_model')
```





# Conclusion

- Summary of WindTunnel's impact on ML pipeline optimization.
- Its role in bridging the gap between classical ML and DNNs.



**Thank you very much for watching !**