💻 KAST 🧪

🧬 K-atalystic Automated Screening Taskflow

Automated Deep Learning Pipeline for Molecular Bioactivity Prediction. A comprehensive, user-friendly solution for training and deploying Machine Learning models in drug discovery

Contents

🚀 What is K-talysticFlow?

K-talysticFlow or **K**-atalystic **A**utomated **S**creening **T**askflow (KAST) is a fully automated, interactive pipeline designed to streamline the process of training, evaluating, and using Deep Learning models for predicting molecular bioactivity. Built on a robust stack including DeepChem, RDKit, and TensorFlow, it provides an end-to-end solution for computational drug discovery.

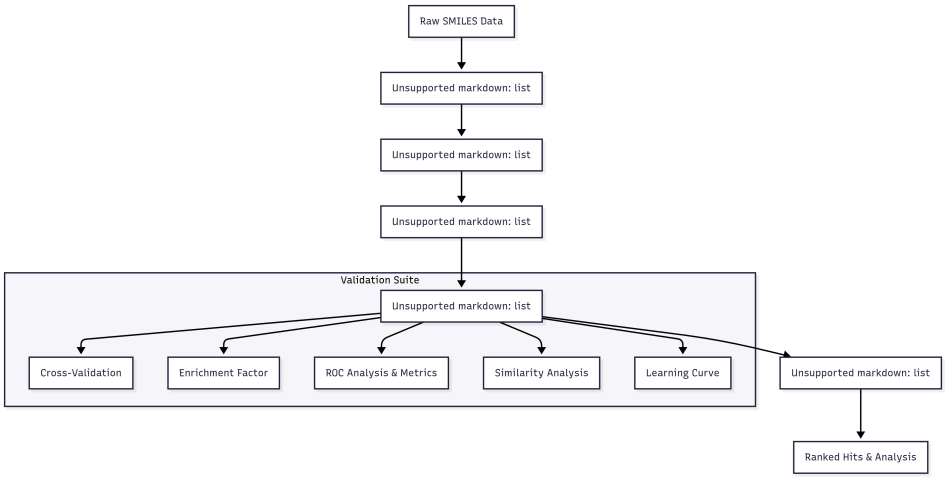
✨ Key Features

- 🤖 **Fully Automated**: Interactive menu-driven interface for a seamless workflow.
- 🧪 **Deep Learning Model**: Utilizes a Multi-Layer Perceptron (MLP) trained on Morgan Fingerprints for high-performance prediction.
- 📊 **Comprehensive Validation Suite**: Rigorous model assessment including ROC analysis, Enrichment Factor, k-fold Cross-Validation with Scaffold Splitting, and Learning Curve generation.
- 🔄 **Complete End-to-End Pipeline**: Manages the entire process from raw SMILES data to actionable predictions.
- 🐧 **Cross-Platform**: Compatible with Windows and Linux.
- 📈 **Analysis-Ready Outputs**: Generates clear reports, graphs, and CSV files for easy interpretation and further analysis.

📚 Quick Navigation

Section	Description
🚀 Installation Guide	Complete setup instructions and requirements
📖 User Manual	Step-by-step usage guide with examples
🔬 Pipeline Steps	Detailed documentation of each script
📊 K-Prediction Score Analysis	How to interpret results and metrics
❓ FAQ	Frequently asked questions
🛠️ Troubleshooting	Common issues and solutions

🔄 Pipeline Overview



Pipeline Overview

Pipeline Steps:

- 📋 **Data Preparation** (`1_preparation.py`) - Cleans and splits molecular datasets using Scaffold Splitting.
- 🧬 **Featurization** (`2_featurization.py`) - Converts SMILES to Morgan Fingerprints (ECFP).
- 🤖 **Model Training** (`3_training.py`) - Trains a Multi-Layer Perceptron (MLP) deep neural network.
- 📊 **Model Evaluation** (`4_*.py`) - Performs a comprehensive performance assessment with a full suite of validation scripts.
- 🔮 **Predictions** (`5_*.py`) - Predicts the activity score for new molecules.

🎯 Getting Started

1. [Install K-talysticFlow] - Set up your environment.
2. [Follow the User Manual] - Run your first analysis.
3. [Understand the Outputs] - Interpret your results.

📞 Support & Contact

- **GitHub Issues**: Report bugs or request features
- **Wiki**: Browse this documentation for detailed guides.
- **Discussions**: Community discussions (if enabled)

Last Updated: 2025-08-23

Made with ❤️ for the computational chemistry community by Késsia Souza (@kelsouzs)

🚀 Installation

Complete setup instructions for K-talysticFlow on Windows and Linux systems.

📋 System Requirements

Minimum Requirements

- **Python**: 3.8 - 3.10 (3.11+ not fully supported by DeepChem)
- **RAM**: 8GB minimum, 16GB+ recommended
- **Storage**: 5GB free space for dependencies
- **OS**: Windows 10+ or Ubuntu 18.04+

Recommended for Large Datasets

- **CPU**: Multi-core processor (4+ cores)
- **RAM**: 16GB+ for datasets with >10,000 compounds
- **Storage**: SSD for faster I/O operations

🛠️ Installation Methods

Method 1: Conda (Recommended)

```
# Create conda environment
conda create -n ktalysticflow python=3.9
conda activate ktalysticflow

# Install core dependencies
conda install -c conda-forge rdkit-pypi
pip install deepchem[tensorflow]
pip install pandas numpy scikit-learn matplotlib seaborn tqdm

# Clone the repository
git clone https://github.com/kelsouzs/KAST.git
cd KAST

# Test installation
python bin/check_env.py
```

Method 2: pip + Virtual Environment

```
# Create virtual environment
python -m venv ktalysticflow
source ktalysticflow/bin/activate  # Linux
# ktalysticflow\Scripts\activate     # Windows

# Install dependencies
pip install rdkit-pypi
pip install deepchem[tensorflow]
pip install pandas numpy scikit-learn matplotlib seaborn tqdm

# Clone repository
git clone https://github.com/kelsouzs/KAST.git
cd KAST

# Test installation
python bin/check_env.py
```

🔧 Detailed Dependencies

Core Libraries

```
# Essential packages
tensorflow≥2.8.0
deepchem≥2.7.0
rdkit-pypi≥2022.9.1
pandas≥1.3.0
```

```
numpy≥1.21.0
scikit-learn≥1.0.0

# Visualization
matplotlib≥3.5.0
seaborn≥0.11.0

# Utilities
tqdm≥4.62.0
```

Optional Dependencies

```
# For advanced analysis
jupyter≥1.0.0
plotly≥5.0.0
```

✅ Verification Steps

1. Check Environment

```
cd KAST
python bin/check_env.py
```

Expected Output:

```
✅ Python version: 3.9.x
✅ TensorFlow: 2.x.x
✅ DeepChem: 2.x.x
✅ RDKit: 202x.x.x
✅ All dependencies satisfied!
```

🐛 Common Installation Issues

Issue 1: RDKit Installation Failed

```
# Solution 1: Use conda
conda install -c conda-forge rdkit

# Solution 2: Build from source (advanced)
pip install rdkit-pypi --no-cache-dir
```

Issue 2: TensorFlow Import Error

```
# Check TensorFlow version
python -c "import tensorflow as tf; print(tf.__version__)"

# Install compatible version
pip install tensorflow==2.10.0
```

Issue 3: DeepChem Import Error

```
# Downgrade Python if using 3.11+
conda create -n ktalysticflow python=3.9
conda activate ktalysticflow

# Reinstall DeepChem
pip install --upgrade deepchem
```

Issue 4: Memory Issues During Training

```
# Set environment variables
export TF_CPP_MIN_LOG_LEVEL=2

# On Windows
set TF_CPP_MIN_LOG_LEVEL=2
```

🔄 Update Instructions

Update K-talysticFlow

```
cd KAST
git pull origin main
```

Update Dependencies

```
# Update all packages
pip install --upgrade deepchem tensorflow pandas numpy

# Check for compatibility
python bin/check_env.py
```

🏗️ Development Setup

For Contributors

```
# Clone with development branch
git clone -b develop https://github.com/kelsouzs/KAST.git

# Install in development mode
pip install -e .

# Install development dependencies
pip install pytest black flake8
```

Running Tests

```
# Run test suite (when available)
pytest tests/

# Lint code
```

```
black bin/
flake8 bin/
```

🆘 Getting Help

If you encounter issues:

1. **Check the logs** in `logs/` directory
2. **Run environment check**: `python bin/check_env.py`
3. **Search existing issues**: [GitHub Issues]
4. **Create new issue** with error logs and system info

📖 User Manual

Complete guide to using K-talysticFlow for molecular bioactivity prediction.

🎯 Quick Start

1. Launch the Pipeline

```
cd KAST
python main.py
```

2. Choose Your Workflow

Option A: Full Training Pipeline 1. `[1] Data Preparation` → `[2] Featurization` → `[3] Training` → `[4] Evaluation`

Option B: Prediction Only 1. `[5] Load Database & Featurize` → `[2] Only Predict`

Option C: Analysis Tools 1. `[4] Evaluate Model` → `[Cross-validation]` → `[Enrichment]` → and more...

📊 Data Requirements

Input Data Format

Training Data (`data/` folder):

```
smiles,activity
CCO,1
CCC,0
c1ccccc1,1
CC(C)O,0
```

Prediction Data (`data/` folder):

```
smiles
CCO
CCC
```

```
CCCO
c1ccccc1O
```

Data Quality Guidelines

- ✅ **Valid SMILES**: Use canonical SMILES when possible
- ✅ **Balanced Dataset**: Similar numbers of active/inactive compounds
- ✅ **Clean Data**: Remove duplicates and invalid structures
- ✅ **Size**: Minimum 1000 compounds for training, no limit for prediction

---

🔄 Complete Workflow Guide

Phase 1: Data Preparation

Step 1: Data Preparation (`1_preparation.py`)

```
[1] Data Preparation
```

What it does: - Loads your .smi file with SMILES and activity data - Validates molecular structures using RDKit - Removes invalid/duplicate SMILES - Splits data into train/validation/test sets - Saves cleaned datasets

Input: Raw .smi file in `data/` folder **Output:** Clean train/val/test CSV files in `data/prepared/`

Interactive Process: 1. Select your input .smi file 2. Choose split ratios (default: 70/15/15) 3. Review data statistics 4. Confirm data splits

Step 2: Featurization (`2_featurization.py`)

```
[2] Featurization
```

What it does: - Converts SMILES to Morgan Circular Fingerprints - Creates binary fingerprint vectors (default: 2048 bits) - Saves featurized datasets for training

Settings (in `settings.py`):

```
FP_RADIUS = 3       # Fingerprint radius
FP_SIZE = 2048      # Fingerprint size
```

Output: Featurized datasets in `data/featurized/`

Phase 2: Model Training

Step 3: Model Training (`3_training.py`)

```
[3] Model Training
```

What it does: - Builds Multi-Layer Perceptron (MLP) model using DeepChem's MultitaskClassifier" - Trains on featurized data - Implements early stopping - Saves trained model

Model Architecture: - **Input:** Morgan fingerprints (2048 dimensions) - **Hidden Layers:** 3 layers with dropout - **Output:** Binary classification (active/inactive) - **Optimizer:** Adam with learning rate scheduling

Training Process: 1. Load featurized training data 2. Initialize GCN model 3. Train with validation monitoring 4. Save best model checkpoint

Phase 3: Model Evaluation

Step 4: Main Evaluation (`4_0_evaluation_main.py`)

```
[4] Evaluate the Model
```

What it does: - Tests model on held-out test set - Calculates comprehensive metrics - Generates ROC curve - Exports predictions for further analysis

Metrics Calculated: - **ROC-AUC**: Area under ROC curve - **Precision/Recall**: Classification accuracy metrics - **F1-Score**: Harmonic mean of precision/recall - **Matthews Correlation**: Balanced metric for imbalanced data

Outputs: - `4_0_evaluation_report.txt` : Text summary - `4_0_roc_curve.png` : ROC curve plot - `4_0_test_predictions.csv` : Detailed predictions

Advanced Evaluation Tools

Cross-Validation (`bin/4_1_cross_validation.py`)

```
[4] Evaluate Model → [Cross-validation]
```

- 5-fold cross-validation using scaffold splitting
- Reports mean AUC ± standard deviation
- Validates model robustness

Enrichment Factor (`bin/4_2_enrichment_factor.py`)

```
[4] Evaluate Model → [Enrichment]
```

- Calculates enrichment at 1%, 5%, 10% thresholds
- Measures early recognition performance
- Essential for virtual screening validation

Tanimoto Similarity Analysis (`bin/4_3_tanimoto_similarity.py`)

```
[4] Evaluate Model → [Tanimoto Similarity]
```

- Computes Tanimoto similarity between molecular fingerprints
- Assesses chemical diversity and redundancy in datasets and predictions
- Useful for analyzing scaffold hopping and diversity in hits

Learning Curve Plot (`bin/4_4_learning_curve.py`)

```
[4] Evaluate Model → [Learning Curve]
```

- Plots training and validation metrics across epochs

- Assesses model convergence, overfitting, and data sufficiency
- Aids in hyperparameter tuning and dataset size decisions

Phase 4: Predictions

Step 5: Prediction Workflow

5.0: Featurize New Data (`5_0_featurize_for_prediction.py`)

```
[5] Load Database & Featurize for Prediction
```

1. Select SMILES file from `data/` folder
2. Featurize all molecules
3. Save featurized dataset

5.1: Run Predictions (`5_1_run_prediction.py`)

```
[2] Only Predict
```

1. Load featurized prediction data
2. Load trained model
3. Generate predictions
4. Save ranked results

Output: `predictions.csv` with molecules ranked by predicted activity

🎛️ Configuration Options

Main Settings (`settings.py`)

Data Processing

Featurization

Model Training

Evaluation

🚀 Advanced Usage

Custom Fingerprints

```
# Modify in settings.py
```

Next: 🔬 Pipeline Steps | 📊 Output Analysis

🔬 Pipeline Steps

This documentation describes the scripts contained in the `/bin` directory of the **K-talysticFlow** pipeline, including purpose, inputs, outputs, and dependency workflow.

📋 Script Overview

Script	Purpose	Input	Output
check_env.py	Validates Python environment	-	Console report
1_preparation.py	Loads, cleans, and splits data	`.smi` files (actives & inactives)	`01_train_set.csv`, `01_test_set.csv`
2_featurization.py	Converts SMILES to numeric vectors	Clean CSVs from step 1	Featurized datasets in `/featurized_datasets`
3_training.py	Trains the MLP model	Featurized training data	Trained model in `/trained_model`
4_0_evaluation_main.py	Main model evaluation	Featurized test data + trained model	Report, ROC curve, prediction CSV
4_1_cross_validation.py	Assesses model robustness (k-fold CV)	Complete `.smi` dataset	Cross-validation report
4_2_enrichment_factor.py	Calculates enrichment metrics	Prediction CSV from `4_0`	EF report
4_3_tanimoto_similarity.py	Analyzes similarity between train and test	Clean CSVs from step 1	Report, histogram, and metrics
4_4_learning_curve.py	Generates learning curves	Complete `.smi` dataset	Learning curve plot and data
5_0_featurize_for_prediction.py	Featurizes new molecules for screening	New `.smi` file	Featurized data in `/prediction_featurized`
5_1_run_prediction.py	Predicts activity of new molecules	Featurized data + trained model	Ranked predictions (CSV)

🔧 `check_env.py`

- **Purpose:** Validates that all dependencies (Python, DeepChem, RDKit, etc.) are installed.
- **Usage:**

  ```
  python bin/check_env.py
  ```

  Output: Console report with installed versions or missing packages.

📊 1_preparation.py

Purpose: Loads SMILES data, assigns labels (1 = actives, 0 = inactives), cleans, and splits into train/test.

Key Features: - Loads from `.smi` as per `settings.py`. - Uses Scaffold Splitting by default. - Switches to Stratified Splitting if needed.

Output: - `results/01_train_set.csv` - `results/01_test_set.csv`

🧬 2_featurization.py

Purpose: Converts SMILES into numeric vectors (fingerprints).

Key Features: - Generates Morgan fingerprints (ECFP-like). - Creates DeepChem DiskDataset.

Settings (`settings.py`):

```
FP_RADIUS = 3
FP_SIZE = 2048
```

Output: `results/featurized_datasets/` (subfolders train/ and test/).

🤖 3_training.py

Purpose: Trains MLP model (MultitaskClassifier).

Config (`settings.py`):

```
MODEL_PARAMS = {
    'n_tasks': 1,
    'layer_sizes': [1000, 500],
    'dropouts': 0.25,
    'learning_rate': 0.001,
    'mode': 'classification',
    'nb_epoch': 50
}
```

Output: - Final model in `results/trained_model/` - Log: `results/03_training_log.txt`

📈 4_0_evaluation_main.py

Purpose: Evaluates model on the test set.

Metrics: ROC-AUC, Accuracy, Precision, Recall, Specificity, F1-Score.

Output: - `results/4_0_evaluation_report.txt` - `results/4_0_roc_curve.png` - `results/4_0_test_predictions.csv`

🔄 4_1_cross_validation.py

Purpose: Assesses model stability via k-fold CV (default: 5-fold).

Output: `results/4_1_cross_validation_results.txt`

📊 4_2_enrichment_factor.py

Purpose: Calculates Enrichment Factor (EF) for virtual screening.

Input: `4_0_test_predictions.csv`

Output: `resultados/enrichment_metrics_results.txt`

🧪 4_3_tanimoto_similarity.py

Purpose: Analyzes chemical similarity between train and test.

Key Features: - Tanimoto coefficient on Morgan fingerprints.

Output: - `results/4_3_similarity_analysis_log.txt` - `results/4_3_similarity_test_actives_to_train.png` - `results/4_3_test_actives_similarity_to_train.csv`

📈 4_4_learning_curve.py

Purpose: Generates learning curves (overfitting/underfitting).

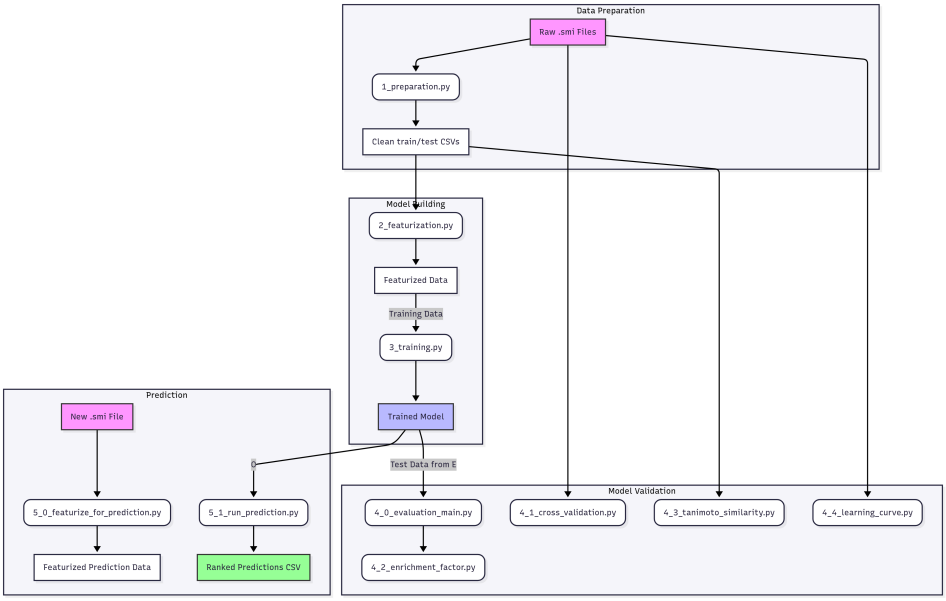
Output: - `results/4_4_learning_curve.png` - `results/4_4_learning_curve_data.csv`

🔮 5_0_featurize_for_prediction.py & 5_1_run_prediction.py

5_0_featurize_for_prediction.py
- **Input:** New `.smi`
- **Output:** `results/5_0_prediction_featurized/`

5_1_run_prediction.py
- **Input:** Featurized data + trained model
- **Output:** `results/5_0_new_molecule_predictions.csv`

🔗 Script Dependencies & Workflow



Script Dependencies & Workflow

Shared dependencies: `settings.py`, `utils.py`, and main libraries (DeepChem, RDKit, etc.).

📊 Output Analysis

An in-depth analysis of the mathematical foundation and interpretation of K-talysticFlow prediction scores.

🎯 K-Prediction Score: Mathematical Foundation

1. Score Function Definition

The **K-Prediction Score** represents the predicted probability that a compound exhibits bioactivity, based on its molecular fingerprint representation. It is the final result of a complex non-linear function learned by a neural network.

Fundamental Equation

```
K-Prediction Score = Softmax(f_MLP(x))
```

Where: - **Softmax** = Softmax activation function, which converts raw scores into probabilities - **f_MLP(x)** = The output of the Multi-Layer Perceptron (MLP) neural network before the final activation - **x** = The Morgan Fingerprint input vector (2048 dimensions)

Detailed Mathematical Implementation

```python
def k_prediction_score_equation(morgan_fingerprint):
    """
    K-prediction Score = Softmax(z_final)[active_class]

    Where z_final is calculated as:
    z_final = h₂ · W_final + b_final
    h₂ = ReLU(h₁ · W₂ + b₂)
    h₁ = ReLU(x · W₁ + b₁)

    Parameters:
    - x = Morgan Fingerprint input vector (2048D)
    - W₁, W₂, W_final = Learned weight matrices [2048→1000], [1000→500], [500→2]
    - b₁, b₂, b_final = Learned bias vectors
    - ReLU(z) = max(0, z)
    - Softmax(zᵢ) = exp(zᵢ) / Σ exp(zⱼ)
    """

    # Layer 1: Input → Hidden Layer 1
    z₁ = W₁ @ morgan_fingerprint + b₁
    h₁ = ReLU(z₁)  # Output with 1000 dimensions

    # Layer 2: Hidden Layer 1 → Hidden Layer 2
    z₂ = W₂ @ h₁ + b₂
    h₂ = ReLU(z₂)  # Output with 500 dimensions

    # Output Layer: Generating Logits
    z_final = W_final @ h₂ + b_final  # Output with 2 dimensions [inactive_logit, active_logit]

    # Softmax Activation to obtain probabilities
    probabilities = Softmax(z_final) # 2D vector, e.g., [0.05, 0.95]

    k_prediction_score = probabilities[1] # Probability of the active class

    return k_prediction_score
```

2. Output Function Properties (Softmax)

The **Softmax** function is ideal for classification as it converts a vector of raw scores (logits) into a probability distribution.

Mathematical Characteristics

```python
def softmax_properties_analysis():
    """
    Softmax Function: Softmax(zᵢ) = exp(zᵢ) / Σⱼ exp(zⱼ)

    Important properties:
    - Σᵢ Softmax(zᵢ) = 1.0 (valid probability distribution)
    - Softmax is monotonic: if zᵢ > zⱼ, then Softmax(zᵢ) > Softmax(zⱼ)
    - Sensitive to differences between logits
    """

    # Interpretation of logits for K-Prediction Score
    logit_interpretations = {
        'active_logit >> inactive_logit': 'K-Prediction Score → 1.0 (high confidence active)',
        'active_logit << inactive_logit': 'K-Prediction Score → 0.0 (high confidence inactive)',
        'active_logit ≈ inactive_logit': 'K-Prediction Score ≈ 0.5 (model uncertainty)'
    }

    return logit_interpretations
```

Sensitivity Analysis

- The **logits** (z_final) represent the evidence that the model has accumulated for each class
- An **active_logit** much larger than the **inactive_logit** will result in a K-Prediction Score close to **1.0**
- An **active_logit** much smaller than the **inactive_logit** will result in a K-Prediction Score close to **0.0**
- If the logits are similar, the K-Prediction Score will be close to **0.5**, indicating **model uncertainty**

⚖️ Score Interpretation and Usage

1. Probabilistic Interpretation

Calibration and Practical Meaning

```python
def score_interpretation_framework():
    """
    The K-Prediction Score is a point probability generated by the model.

    IMPORTANT: Without formal calibration, the predicted probability (e.g., 0.8)
    does NOT necessarily mean an 80% real chance of activity.

    Instead, it should be interpreted as a reliable RANKING SCORE.
    """

    ranking_interpretation = {
        'fundamental_principle': 'K-Prediction Score of 0.9 > Score of 0.8 > Score of 0.7',
        'reliable_ordering': 'The relative ordering of compounds is highly reliable',
        'absolute_probability': 'The absolute value may not reflect real probability',
        'auc_roc_validation': 'The excellent AUC-ROC performance validates the ranking quality'
    }

    return ranking_interpretation
```

Practical Interpretation Example

```
📊 K-Prediction Score Interpretation:

Score 0.95: Compound A
Score 0.87: Compound B
Score 0.72: Compound C
Score 0.34: Compound D

✅ CORRECT Interpretation:
A > B > C > D (priority order for experimental testing)

❌ INCORRECT Interpretation:
"Compound A has a 95% real chance of being active"
```

2. Decision Threshold Optimization

While the **default threshold** for classification is **0.5**, KAST allows for a deeper analysis to find an optimal threshold depending on the screening objective.

Mathematical Implementation of Optimal Threshold

```python
def optimal_threshold_calculation(scores, true_labels):
    """
    Mathematical optimization of the K-Prediction Score threshold for decision.
    This is implemented in the KAST validation suite.

    Optimizes for: argmax_t [Sensitivity(t) + Specificity(t) - 1] (Youden's J)
    """

    from sklearn.metrics import roc_curve
    import numpy as np

    # Calculate the ROC curve
    fpr, tpr, thresholds = roc_curve(true_labels, scores)

    # Youden's J statistic to find the optimal threshold
    # Maximizes the difference between true positive rate and false positive rate
    j_scores = tpr - fpr
    optimal_idx = np.argmax(j_scores)
    optimal_threshold = thresholds[optimal_idx]

    threshold_analysis = {
        'youden_optimal_threshold': optimal_threshold,
        'sensitivity_at_optimal': tpr[optimal_idx],
        'specificity_at_optimal': 1 - fpr[optimal_idx],
    }

    return threshold_analysis
```

⁉️ FAQ

Common questions and answers about K-talysticFlow usage, troubleshooting, and best practices.

🚀 Getting Started

Q: What are the minimum system requirements?
A:
Python: 3.9 or 3.10. Python 3.11+ is not yet fully supported by all dependencies.
RAM: 8 GB minimum. For datasets with over 10,000 molecules, 16 GB+ is recommended.
Storage: ~5 GB free space for the Conda environment and dependencies.
OS: Windows 10+ or a modern Linux distribution (e.g., Ubuntu 18.04+).

Q: Can I run K-talysticFlow on a standard laptop?
A: Yes! KAST is designed to work on standard laptops using CPU only. For very large datasets, training will be significantly faster on a machine with a dedicated NVIDIA GPU.

📊 Data Requirements

Q: What data format do I need?
A: You need two simple text files (.smi) located in the data/ directory:
ativas.smi: Contains one SMILES string per line for your active molecules.
inativas.smi: Contains one SMILES string per line for your inactive molecules or decoys.
KAST will automatically process these files, assign labels (1 for actives, 0 for inactives), and create the necessary CSV files.

Q: How many compounds do I need for training?
A: More high-quality data is always better, especially for Deep Learning. However, KAST has shown strong performance even on focused datasets.
Minimum suggested: ~50-100 active compounds.
Recommended: Several hundred active compounds.
Important: A sufficient number of high-quality inactives or decoys is also crucial.

Q: What's a good active/inactive ratio?
A: Virtual screening is an imbalanced problem. KAST is designed to handle this.
Realistic Scenario: A ratio of 1:10 to 1:50 (or even higher) is common and provides a more rigorous test for the model, simulating a real-world screening scenario.
Balanced Sets (e.g., 1:1): Can also be used, but the model's performance on highly imbalanced datasets might differ.

🔧 Technical Issues

Q: "ModuleNotFoundError: No module named 'deepchem'" - What do I do?
A: This indicates that your Conda/virtual environment is not activated or a library is missing.
First, ensure your environment is active: conda activate kast_env.
If the error persists, the library is likely missing. Reinstall it using the recommended Conda command: conda install -c conda-forge deepchem.

Q: My training is very slow. How can I speed it up?
A: Training time is influenced by dataset size and model complexity.
Use a GPU: This is the most effective way to accelerate training. Ensure you have a compatible version of TensorFlow for your GPU.
Reduce model complexity: In settings.py, you can try smaller layer_sizes (e.g., [512, 256]).
Reduce number of epochs: In settings.py, lower nb_epoch in MODEL_PARAMS (e.g., to 25), but this may result in an under-trained model.

Q: I get a "Memory Error" during featurization or training.
A: This happens when the dataset is too large for your RAM.
Ensure you have at least 8-16 GB of RAM and that other memory-intensive applications are closed.

For very large datasets (>100,000 molecules), consider running the pipeline on a machine with more RAM. The use of DiskDataset in KAST helps mitigate this, but featurization can still be memory-intensive.

🔮 Predictions

Q: How do I interpret the K-Activity Score?
A: It is the model's predicted probability that a compound is active. It should be used for ranking.
Score near 1.0: The model is very confident the compound is active. These are your top candidates.
Score near 0.5: The model is uncertain.
Score near 0.0: The model is very confident the compound is inactive.

🛠️ Configuration

Q: Can I change the fingerprint settings?
A: Yes, in settings.py. The recommended defaults are robust.

```
FP_RADIUS = 3
FP_SIZE = 2048
```

Q: How do I adjust the model architecture?
A: Modify MODEL_PARAMS in settings.py.

```
MODEL_PARAMS = {
    'layer_sizes': [1000, 500],  # Default 2 hidden layers
    'dropouts': 0.25,            # Default regularization
    'learning_rate': 0.001       # Default learning rate
}
```

Q: How do I change the train/test split ratio?
A: Modify TEST_SET_FRACTION in settings.py.

```
TEST_SET_FRACTION = 0.2  # Sets aside 20% of data for testing
```

📁 File Management

Q: Where are my results saved?
A: All outputs are saved in the results/ directory. Each script prefixes its output files (e.g., 4_0_evaluation_report.txt, 4_1_cross_validation_results.txt).

Q: Can I move my trained model to another computer?
A: Yes. You need to copy the entire results/trained_model/ directory, as it contains the model weights and necessary metadata.

Q: Do I need to re-run featurization every time I train a model?
A: No. As long as your input data (.smi) and your fingerprint settings in settings.py have not changed, you can re-run the training (Step 3) multiple times using the existing featurized data.

🛠️ Troubleshooting

Comprehensive guide to diagnosing and fixing common issues in K-talysticFlow.

🚨 Installation Issues

Issue 1: DeepChem Installation Failed

Symptoms

```
ERROR: Failed building wheel for deepchem
ERROR: Could not build wheels for deepchem
ModuleNotFoundError: No module named 'deepchem'
```

Solutions

```
# Solution 1: Use conda (recommended)
conda create -n ktalysticflow python=3.9
conda activate ktalysticflow
conda install -c conda-forge rdkit deepchem

# Solution 2: Pip with specific version
pip install deepchem==2.7.1
pip install tensorflow==2.10.0

# Solution 3: Force reinstall
pip uninstall deepchem tensorflow
pip install --no-cache-dir deepchem[tensorflow]
```

Prevention

- Always use Python 3.8-3.10 (avoid 3.11+)
- Use conda environments to avoid conflicts
- Install dependencies in correct order

Issue 2: RDKit Import Errors

Symptoms

```
ImportError: cannot import name 'Chem' from 'rdkit'
ModuleNotFoundError: No module named 'rdkit'
```

Solutions

```
# Solution 1: Conda installation
conda install -c conda-forge rdkit

# Solution 2: Pip installation
pip install rdkit-pypi

# Solution 3: Complete reinstall
pip uninstall rdkit rdkit-pypi
conda install -c conda-forge rdkit
```

Issue 3: TensorFlow Compatibility Issues

Symptoms

```
ImportError: cannot import name 'utils' from 'tensorflow.python'
AttributeError: module 'tensorflow' has no attribute 'Session'
```

Solutions

```
# Check TensorFlow version
python -c "import tensorflow as tf; print(tf.__version__)"

# Install compatible version
pip install tensorflow==2.10.0  # Most stable with DeepChem

# For older systems
pip install tensorflow==2.8.0
```

💾 Data Loading Problems

Issue 4: CSV File Not Found

Symptoms

```
FileNotFoundError: [Errno 2] No such file or directory: 'data/your_file.csv'
ERROR: Could not find any CSV files in the data directory
```

Solutions

```
# Check file location
ls data/                    # Linux
dir data\                   # Windows

# Ensure correct structure
KAST/
├── data/
│   ├── your_dataset.csv   # Your CSV file here
│   └── ...
├── bin/
└── ...

# Check file permissions
chmod 644 data/your_file.csv  # Linux
```

File Format Requirements

```
# Correct format
smiles,activity
CCO,1
CCC,0
c1ccccc1,1

# Common mistakes to avoid:
# ❌ Wrong column names: "SMILES,Activity"
# ❌ Missing header row
```

```
# ❌ Extra columns without proper handling
# ❌ Non-binary activity values
```

Issue 5: Invalid SMILES Structures

Symptoms

```
⚠️ WARNING: 245 invalid SMILES found and removed
ERROR: Not enough valid molecules for training
RDKit WARNING: [molecule parsing error]
```

Diagnosis

```python
# Check your SMILES validity
from rdkit import Chem
import pandas as pd

df = pd.read_csv('data/your_file.csv')
valid_count = 0
invalid_smiles = []

for smi in df['smiles']:
    mol = Chem.MolFromSmiles(smi)
    if mol is None:
        invalid_smiles.append(smi)
    else:
        valid_count += 1

print(f"Valid: {valid_count}, Invalid: {len(invalid_smiles)}")
print("Sample invalid SMILES:", invalid_smiles[:5])
```

Solutions

```python
# Clean your data before training
def clean_smiles_data(csv_file):
    df = pd.read_csv(csv_file)
    valid_rows = []

    for _, row in df.iterrows():
        mol = Chem.MolFromSmiles(row['smiles'])
        if mol is not None:
            valid_rows.append(row)

    clean_df = pd.DataFrame(valid_rows)
    clean_df.to_csv('data/cleaned_dataset.csv', index=False)
    print(f"Saved {len(clean_df)} valid molecules")

clean_smiles_data('data/your_file.csv')
```

🤖 Model Training Issues

Issue 6: Training Stops Immediately

Symptoms

```
Epoch 1/100: Loss: nan, Validation AUC: 0.5000
Training stopped due to early stopping
Model training failed
```

Causes & Solutions

```
# Cause 1: Learning rate too high
MODEL_PARAMS['learning_rate'] = 0.0001  # Reduce from 0.001

# Cause 2: Bad data scaling
# Check for extreme values in your features
# Solution: Use standard fingerprints (handled automatically)

# Cause 3: Insufficient data
# Ensure you have:
# - At least 1000 compounds
# - At least 10% actives in dataset
# - Balanced train/val/test splits

# Cause 4: Memory issues
MODEL_PARAMS['batch_size'] = 32  # Reduce from 128
```

Issue 7: Memory Errors During Training

Symptoms

```
OOM when allocating tensor
MemoryError: Unable to allocate array
ResourceExhaustedError: Out of memory
```

Solutions

```
# Solution 1: Reduce model size
MODEL_PARAMS = {
    'layer_sizes': [500, 500],      # Smaller layers
    'batch_size': 32,               # Smaller batches
}

# Solution 2: Reduce fingerprint size
FP_SIZE = 1024  # Instead of 2048

# Solution 3: System-level fixes


# Close other applications
# Check available RAM
free -h                     # Linux
wmic OS get TotalVisibleMemorySize /value  # Windows

# Restart Python session
# Use 64-bit Python (if on 32-bit)
```

Issue 8: Poor Model Performance

Symptoms

```
ROC-AUC: 0.52 (barely better than random)
Cross-validation: 0.54 ± 0.15 (high variance)
Enrichment Factor @ 1%: 0.8 (worse than random)
```

Diagnostic Steps

```python
# Step 1: Check data quality
def analyze_dataset(csv_file):
    df = pd.read_csv(csv_file)

    print(f"Total compounds: {len(df)}")
    print(f"Active compounds: {df['activity'].sum()}")
    print(f"Inactive compounds: {len(df) - df['activity'].sum()}")
    print(f"Activity ratio: {df['activity'].mean():.2%}")

    # Check for duplicates
    duplicates = df['smiles'].duplicated().sum()
    print(f"Duplicate SMILES: {duplicates}")

    # Check SMILES length distribution
    lengths = df['smiles'].str.len()
    print(f"SMILES length: {lengths.mean():.1f} ± {lengths.std():.1f}")

analyze_dataset('data/your_file.csv')
```

Solutions

```python
# Solution 1: Improve data quality
# - Remove duplicates
# - Balance active/inactive ratio (aim for 20-80%)
# - Increase dataset size (>5000 compounds recommended)

# Solution 2: Adjust model parameters
MODEL_PARAMS = {
    'layer_sizes': [2000, 1000, 500],  # Larger network
    'dropouts': 0.3,                   # Less regularization
    'weight_decay_penalty': 0.0001,    # Less penalty
}

# Solution 3: Try different fingerprint settings
FP_RADIUS = 3      # Larger radius
FP_SIZE = 4096     # More features

# Solution 4: Check for data leakage
# Ensure test compounds are truly different from training
```

🔮 Prediction Issues

Issue 9: Prediction Script Fails

Symptoms

```
ERROR: No featurized prediction data found
ERROR: Could not load trained model
```

```
FileNotFoundError: Model file not found
```

Solutions

```
# Check required files exist
ls models/best_model/                    # Model should be here
ls data/prediction_featurized/           # Featurized data should be here

# Re-run featurization if needed
python bin/5_0_featurize_for_prediction.py

# Check model training completed successfully
ls models/best_model/
# Should contain: model files, config.json, etc.
```

Issue 10: Unrealistic Prediction Scores

Symptoms

```
# All predictions are extreme values
Score: 0.9999 for simple molecules
Score: 0.0001 for complex drugs
# Or all predictions are similar (e.g., 0.5-0.6)
```

Diagnosis

```
# Check prediction distribution
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('results/predictions.csv')
plt.hist(df['K-prediction Score'], bins=50)
plt.xlabel('Prediction Score')
plt.ylabel('Frequency')
plt.title('Prediction Score Distribution')
plt.show()

# Healthy distribution should be spread across 0-1
# Problematic: All values clustered in narrow range
```

Solutions

```
# Solution 1: Check model calibration
# Re-evaluate model on test set to verify performance

# Solution 2: Validate input SMILES
# Ensure prediction molecules are in valid format

# Solution 3: Check for domain shift
# Are prediction molecules very different from training set?

# Solution 4: Model retraining may be needed
# If predictions don't match expected chemistry
```

📁 File System Issues

Issue 11: Permission Denied Errors (Windows)

Symptoms

```
PermissionError: [Errno 13] Permission denied
OSError: [WinError 5] Access is denied
Cannot create directory
```

Solutions

```
# Solution 1: Run as Administrator
# Right-click Command Prompt → "Run as administrator"

# Solution 2: Change directory location
# Move K-talysticFlow to Documents folder
C:\Users\YourName\Documents\KAST\

# Solution 3: Check folder permissions
# Right-click folder → Properties → Security → Full Control

# Solution 4: Disable antivirus temporarily
# Some antivirus software blocks file creation
```

Issue 12: Disk Space Issues

Symptoms

```
OSError: [Errno 28] No space left on device
IOError: Not enough space to write file
```

Check Available Space

```
# Check disk space
df -h                              # Linux
dir                               # Windows (check available space)

# Check K-talysticFlow folder size
du -sh KAST/                      # Linux
```

Space Requirements

```
Typical space usage:
├── Dependencies: ~2-3 GB
├── Training data (10K compounds): ~100 MB
├── Featurized data: ~200 MB
├── Model files: ~50-100 MB
├── Results: ~10-50 MB
└── Total: ~3-4 GB for complete pipeline
```

Solutions

```
# Clean up space
rm -rf data/featurized/*/temp_*    # Remove temp files
rm -rf logs/old_*                  # Remove old logs
rm -rf models/old_*/               # Remove old models

# Move to larger drive
# Copy entire KAST folder to drive with more space
```

⚡ Performance Issues

Issue 13: Very Slow Training

Symptoms

```
Epoch 1/100 - ETA: 2 hours 45 minutes
Training taking much longer than expected
```

Diagnosis

```
# Check dataset size
import pandas as pd
df = pd.read_csv('data/prepared/train_prepared.csv')
print(f"Training compounds: {len(df)}")

# Large datasets (>50K) will naturally take longer
```

Solutions

```
# Solution 1: Optimize batch size
MODEL_PARAMS['batch_size'] = 256  # Increase if you have RAM

# Solution 2: Reduce model complexity
MODEL_PARAMS = {
    'layer_sizes': [1000, 500],     # Fewer/smaller layers
    'epochs': 50,                   # Fewer epochs
}

# Solution 3: Use subset for testing
# Test with smaller dataset first to verify settings
```

Issue 14: High Memory Usage

Symptoms

```
System becomes unresponsive
Other applications crash
Python process uses >8GB RAM
```

Monitoring Memory Usage

```python
# Add memory monitoring to your scripts
import psutil
import os

def check_memory():
    process = psutil.Process(os.getpid())
    memory_mb = process.memory_info().rss / 1024 / 1024
    print(f"Memory usage: {memory_mb:.1f} MB")

# Call periodically during training
check_memory()
```

Solutions

```python
# Solution 1: Reduce fingerprint size
FP_SIZE = 1024  # Instead of 2048

# Solution 2: Process in smaller batches
# Modify featurization to use smaller chunks

# Solution 3: Use memory-efficient formats
# K-talysticFlow already optimized for this

# Solution 4: Add garbage collection
import gc
gc.collect()  # Call periodically
```

🔧 Configuration Issues

Issue 15: Settings Not Applied

Symptoms

```
Changed FP_SIZE to 1024 but still using 2048
Modified layer_sizes but model architecture unchanged
```

Solutions

```python
# Solution 1: Verify settings.py location
# Must be in same directory as scripts

# Solution 2: Check for syntax errors
python -c "import settings; print(settings.FP_SIZE)"

# Solution 3: Clear cached data
# Delete and regenerate featurized data
rm -rf data/featurized/
python bin/2_featurization.py

# Solution 4: Restart Python session
# Some settings are cached in memory
```

Issue 16: Import Path Issues

Symptoms

```
ModuleNotFoundError: No module named 'settings'
ModuleNotFoundError: No module named 'utils'
```

Solutions

```
# Solution 1: Run from correct directory
cd KAST/                    # Must be in project root
python bin/1_preparation.py

# Solution 2: Check file structure
KAST/
├── settings.py            # Must exist here
├── utils.py              # Must exist here
├── bin/
│   ├── 1_preparation.py
│   └── ...

# Solution 3: Fix Python path
export PYTHONPATH="${PYTHONPATH}:$(pwd)"  # Linux
set PYTHONPATH=%PYTHONPATH%;%cd%          # Windows
```

🚨 Emergency Troubleshooting

Complete Reset Procedure

If everything is broken, start fresh:

```
# Step 1: Backup important data
cp data/your_dataset.csv ~/backup/
cp settings.py ~/backup/
cp -r results/ ~/backup/

# Step 2: Clean installation
rm -rf venv/                    # Remove virtual environment
rm -rf data/featurized/         # Remove featurized data
rm -rf data/prepared/           # Remove prepared data
rm -rf models/                  # Remove models

# Step 3: Fresh install
python -m venv venv
source venv/bin/activate        # Linux
# venv\Scripts\activate         # Windows

pip install --upgrade pip
pip install deepchem[tensorflow]
pip install rdkit-pypi pandas numpy scikit-learn matplotlib seaborn tqdm

# Step 4: Test installation
python bin/check_env.py

# Step 5: Restore data and start over
cp ~/backup/your_dataset.csv data/
python main.py
```

Collecting Debug Information

When reporting issues, collect this information:

```
# System information
python --version
python -c "import deepchem; print('DeepChem:', deepchem.__version__)"
python -c "import tensorflow; print('TensorFlow:', tensorflow.__version__)"
python -c "import rdkit; print('RDKit:', rdkit.__version__)"

# Check environment
python bin/check_env.py > debug_info.txt

# Check file structure
find . -name "*.py" | head -20 >> debug_info.txt
ls -la data/ >> debug_info.txt
ls -la models/ >> debug_info.txt

# Include any error logs
cat logs/error.log >> debug_info.txt
```

📞 Getting Help

Before Asking for Help

1. **Check this troubleshooting guide** - Many issues are covered here
2. **Review error messages carefully** - Often contain solution hints
3. **Check existing GitHub issues** - Your problem might already be solved
4. **Test with sample data** - Isolate whether it's your data or the software

When Creating a GitHub Issue

Include: - **System information** (OS, Python version, package versions) - **Complete error message** (copy-paste, don't paraphrase) - **Steps to reproduce** (what did you do before the error?) - **Your data format** (anonymized sample of your CSV) - **Settings modifications** (if you changed anything in settings.py)

Useful Commands for Debug Information

```
# Generate comprehensive debug report
python bin/check_env.py > debug_report.txt
echo "═══ System Info ═══" >> debug_report.txt
python --version >> debug_report.txt
echo "═══ File Structure ═══" >> debug_report.txt
ls -la >> debug_report.txt
echo "═══ Recent Logs ═══" >> debug_report.txt
tail -50 logs/*.log >> debug_report.txt 2>/dev/null || echo "No logs found"
```

📞 **Support:** Create Issue | 📧 **Contact:** kelsouzs.uefs@gmail.com