

K-talysticFlow

Complete Documentation - Automated Deep Learning Pipeline for Molecular Bioactivity Prediction

Késsia Souza Santos

October 10, 2025

Contents

1	About This Documentation	9
2	Home	10
3	KAST Wiki	10
3.1	K-atalystic Automated Screening Taskflow	10
3.2	What is K-talysticFlow?	10
3.3	Key Features	10
3.3.1	Core Capabilities	10
3.3.2	Advanced Features	10
3.4	Quick Navigation	11
3.5	Pipeline Overview	11
3.6	Pipeline Steps	12
3.6.1	1 Data Preparation (1_preparation.py)	12
3.6.2	2 Featurization (2_featurization.py)	12
3.6.3	3 Model Training (3_training.py)	12
3.6.4	4 Model Evaluation Suite	12
3.6.5	5 Prediction on New Molecules	13
3.7	Getting Started	14
3.7.1	Quick Start Guide	14
3.7.2	Recommended Workflow	14
3.8	System Requirements	14
3.8.1	Minimum Requirements	14
3.8.2	Recommended Requirements	14
3.8.3	Key Dependencies	14
3.9	Citation	15
3.10	Support & Contact	15
3.11	License	15
3.12	Acknowledgments	15
4	Installation	16
5	Installation Guide	16

5.1	Prerequisites	16
5.2	Step-by-Step Installation	16
5.2.1	1. Clone the Repository	16
5.2.2	2. Create a Conda Environment (Recommended)	16
5.2.3	3. Install RDKit via Conda (Recommended Method)	16
5.2.4	4. Install Python Dependencies	17
5.2.5	5. Verify Installation	17
5.3	Platform-Specific Instructions	17
5.3.1	Linux	17
5.3.2	Windows	18
5.3.3	macOS	18
5.4	Docker Installation (Alternative)	18
5.5	Test Your Installation	18
5.5.1	Quick Test	18
5.6	Optional: GPU Support	18
5.6.1	NVIDIA GPU	18
5.6.2	Verify GPU	19
5.7	requirements.txt Contents	19
5.8	Updating K-talysticFlow	19
5.9	Uninstallation	19
5.10	Troubleshooting Installation	20
5.10.1	Issue: RDKit installation fails	20
5.10.2	Issue: TensorFlow errors	20
5.10.3	Issue: Memory errors during installation	20
5.10.4	Issue: Permission denied (Linux/Mac)	20
5.11	Next Steps	20
5.12	Need Help?	20
6	User Manual	21
7	User Manual	21
7.1	Table of Contents	21
7.2	Quick Start	21
7.2.1	First Run	21
7.3	Preparing Your Data	22
7.3.1	Data Format	22
7.3.2	File Requirements	22
7.3.3	Data Quality Guidelines	22
7.4	Running the Pipeline	22
7.4.1	Option 1: Complete Pipeline (Recommended for First Use)	22
7.4.2	Option 2: Step-by-Step Workflow	23
7.5	Advanced Options [8]	25
7.5.1	[1] Check Dependencies	25
7.5.2	[2] Test Parallel Processing	25
7.5.3	[3] Configure Workers	25
7.6	Understanding Outputs	25
7.6.1	Key Output Files	25
7.6.2	Understanding K-Prediction Score	26

7.7	Best Practices	26
7.7.1	Do's	26
7.7.2	Don'ts	26
7.8	Typical Workflows	26
7.8.1	Workflow A: New Project	26
7.8.2	Workflow B: Model Optimization	27
7.8.3	Workflow C: Batch Predictions	27
7.9	Time Estimates	27
7.10	Need Help?	27
8	Pipeline Steps	29
9	Pipeline Steps - Detailed Documentation	29
9.1	Table of Contents	29
9.2	Overview	29
9.3	Step 1: Data Preparation	29
9.3.1	Input	29
9.3.2	Process	30
9.3.3	Output	31
9.3.4	Configuration	31
9.3.5	Troubleshooting	31
9.4	Step 2: Featurization	31
9.4.1	Input	32
9.4.2	Process	32
9.4.3	Output	33
9.4.4	Configuration	34
9.4.5	Troubleshooting	34
9.5	Step 3: Model Training	34
9.5.1	Input	34
9.5.2	Process	34
9.5.3	Output	36
9.5.4	Configuration	37
9.5.5	Troubleshooting	37
9.6	Step 4: Model Evaluation	37
9.6.1	4.0 Main Evaluation	37
9.6.2	4.1 Cross-Validation	38
9.6.3	4.2 Enrichment Factor	39
9.6.4	4.3 Tanimoto Similarity	39
9.6.5	4.4 Learning Curve	39
9.7	Step 5-6: Prediction	40
9.7.1	Step 5: Featurization for Prediction	40
9.7.2	Step 6: Run Prediction	40
9.8	Complete Pipeline Flow	41
9.9	Related Pages	41
10	Parallel Processing	42
11	Parallel Processing Guide	42

11.1	What is Parallel Processing?	42
11.2	Performance Gains	42
11.2.1	Real-World Benchmarks	42
11.3	Configuration	42
11.3.1	Method 1: settings.py (Permanent)	42
11.3.2	Method 2: Runtime Configuration (Temporary)	43
11.4	Optimal Configuration Guide	43
11.4.1	Auto Mode (Recommended)	43
11.4.2	Fixed Core Count	44
11.4.3	Maximum Performance	44
11.4.4	Sequential Processing	44
11.5	Hardware-Specific Recommendations	44
11.5.1	Low-End Systems	44
11.5.2	Mid-Range Systems	44
11.5.3	High-End Systems	45
11.5.4	Server/HPC Systems	45
11.6	Testing Your Configuration	45
11.6.1	Run the Test Suite	45
11.7	Memory Considerations	46
11.7.1	PARALLEL_BATCH_SIZE	46
11.8	Which Scripts Use Parallelism?	46
11.9	Troubleshooting	46
11.9.1	Issue: No speedup observed	46
11.9.2	Issue: System becomes unresponsive	47
11.9.3	Issue: Out of memory errors	47
11.9.4	Issue: “joblib” import error	47
11.9.5	Issue: Slower than expected	47
11.10	Performance Monitoring	47
11.10.1	View Real-Time Status	47
11.10.2	Check Logs	48
11.11	Advanced: Custom Parallelization	48
11.12	Best Practices	48
11.12.1	Do’s	48
11.12.2	Don’ts	48
11.13	Related Pages	49
12	Output Analysis	50
13	Output Analysis: K-Prediction Score	50
13.1	K-Prediction Score: Mathematical Foundation	50
13.1.1	1. Score Function Definition	50
13.1.2	2. Output Function Properties (Softmax)	51
13.2	Score Interpretation and Usage	52
13.2.1	1. Probabilistic Interpretation	52
13.2.2	2. Decision Threshold Optimization	52
14	Configuration	54

15	Configuration Guide	54
15.1	Configuration File: <code>settings.py</code>	54
15.2	Configuration Sections	54
15.2.1	Section 1: Main Paths	54
15.2.2	Section 2: Basic Configurations	54
15.2.3	Section 5: Model Parameters	55
15.2.4	Section 6: Training Configurations	56
15.2.5	Section 7: Validation Configurations	57
15.2.6	Section 8: Data Validation Configurations	58
15.2.7	Section 12: Parallel Processing Configurations	58
15.3	Configuration Recipes	58
15.3.1	Recipe 1: Fast Testing	58
15.3.2	Recipe 2: Production Model	59
15.3.3	Recipe 3: Large Dataset (> 50K)	59
15.3.4	Recipe 4: Low Memory (8GB RAM)	59
15.4	Advanced Customization	60
15.4.1	Modifying Scripts	60
15.4.2	Custom Loss Functions	60
15.4.3	Custom Metrics	60
15.5	Configuration Best Practices	61
15.5.1	Do's	61
15.5.2	Don'ts	61
15.6	Applying Configuration Changes	61
15.6.1	Changes requiring re-run:	61
15.7	Monitoring Configuration Impact	61
15.7.1	Compare Model Versions	61
15.8	Related Pages	62
16	FAQ	63
17	Frequently Asked Questions (FAQ)	63
17.1	General Questions	63
17.1.1	What is K-talysticFlow?	63
17.1.2	Who should use K-talysticFlow?	63
17.1.3	Is K-talysticFlow free?	63
17.1.4	What makes K-talysticFlow different?	63
17.2	Data & Inputs	63
17.2.1	What input format does K-talysticFlow accept?	63
17.2.2	How much data do I need?	64
17.2.3	What is a good active/inactive ratio?	64
17.2.4	Can I use my own molecular descriptors?	64
17.2.5	What if I only have active compounds?	65
17.2.6	Can I predict multiple targets at once?	65
17.3	Model & Training	65
17.3.1	How long does training take?	65
17.3.2	Can I use my own neural network architecture?	65
17.3.3	What is a good ROC-AUC score?	66
17.3.4	My model overfits. What should I do?	66

17.3.5	Can I use a GPU?	66
17.4	Performance & Speed	67
17.4.1	How can I make it faster?	67
17.4.2	Why is parallel processing not helping?	67
17.4.3	How much RAM do I need?	67
17.5	Predictions & Screening	68
17.5.1	What is the K-Prediction Score?	68
17.5.2	How many compounds should I test experimentally?	68
17.5.3	Can I screen a million compounds?	68
17.5.4	Are predictions reliable for molecules very different from training set?	69
17.6	Technical Questions	69
17.6.1	What Python version do I need?	69
17.6.2	Can I run K-talysticFlow on Windows?	69
17.6.3	Do I need Conda or can I use pip?	69
17.6.4	Can I use K-talysticFlow in Jupyter Notebooks?	70
17.6.5	Is there a Docker image?	70
17.6.6	Can I use K-talysticFlow in a web application?	70
17.7	Scientific Questions	70
17.7.1	What type of molecular activity can I predict?	70
17.7.2	How does K-talysticFlow compare to other tools?	71
17.7.3	Can I publish results from K-talysticFlow?	71
17.7.4	What are the limitations of K-talysticFlow?	71
17.8	More Help	72
18	Troubleshooting	73
19	Troubleshooting Guide	73
19.1	Table of Contents	73
19.2	Installation Issues	73
19.2.1	Issue: <code>ModuleNotFoundError: No module named 'rdkit'</code>	73
19.2.2	Issue: <code>ImportError: DLL load failed (Windows)</code>	73
19.2.3	Issue: TensorFlow installation fails	73
19.2.4	Issue: <code>ImportError: cannot import name 'DeepChem'</code>	74
19.2.5	Issue: Permission denied when installing	74
19.3	Data Preparation Errors	74
19.3.1	Issue: <code>FileNotFoundError: 'actives.smi' not found</code>	74
19.3.2	Issue: <code>ValueError: Invalid SMILES: XYZ</code>	74
19.3.3	Issue: <code>Error: Insufficient data (< 50 molecules per class)</code>	75
19.3.4	Issue: Train/test split fails	75
19.4	Featurization Problems	75
19.4.1	Issue: <code>MemoryError</code> during featurization	75
19.4.2	Issue: Featurization extremely slow	75
19.4.3	Issue: <code>ValueError: Fingerprint size must be > 0</code>	76
19.4.4	Issue: RDKit WARNING: not removing hydrogen atom	76
19.5	Training Issues	76
19.5.1	Issue: Training stuck at 0% for long time	76
19.5.2	Issue: <code>ValueError: No training data found</code>	76
19.5.3	Issue: Training loss not decreasing	77

19.5.4	Issue: <code>CUDA out of memory</code> (GPU)	77
19.5.5	Issue: Training finishes but no model file	77
19.6	Memory Errors	77
19.6.1	Issue: <code>MemoryError: Unable to allocate array</code>	77
19.6.2	Issue: Python process killed suddenly	78
19.7	Parallel Processing Problems	78
19.7.1	Issue: No speedup from parallelism	78
19.7.2	Issue: <code>joblib</code> errors	79
19.7.3	Issue: Parallel test suite fails	79
19.7.4	Issue: Slower with parallelism enabled	79
19.8	Prediction Errors	79
19.8.1	Issue: <code>FileNotFoundError: Model checkpoint not found</code>	79
19.8.2	Issue: Predictions all the same value	80
19.8.3	Issue: <code>ValueError: Feature mismatch</code>	80
19.8.4	Issue: Predictions take too long	80
19.9	Performance Issues	80
19.9.1	Issue: Pipeline very slow overall	80
19.9.2	Issue: Specific script very slow	81
19.10	General Errors	81
19.10.1	Issue: <code>ImportError: cannot import name 'MultitaskClassifier'</code>	81
19.10.2	Issue: Control panel menu not displaying correctly	81
19.10.3	Issue: Logs not generated	82
19.10.4	Issue: <code>PermissionError: [Errno 13]</code>	82
19.10.5	Issue: Conflicting package versions	82
19.10.6	Issue: Script exits without error message	82
19.10.7	Issue: Results folder messy/corrupted	83
19.11	Getting More Help	83
19.11.1	Step 1: Check Logs	83
19.11.2	Step 2: Enable Debugging	83
19.11.3	Step 3: Run Dependency Checker	83
19.11.4	Step 4: Test Parallel Processing	83
19.11.5	Step 5: Minimal Reproducible Example	84
19.11.6	Step 6: Report Issue	84
19.12	Related Resources	84
20	Index	85
21	K-talysticFlow Wiki - Complete Index	85
21.1	All Wiki Pages	85
21.1.1	Main Pages	85
21.1.2	Getting Started (Essential)	85
21.1.3	Core Documentation	85
21.1.4	Advanced Topics	85
21.1.5	Help & Support	86
21.1.6	Meta Documentation	86
21.2	Quick Access by Category	86
21.2.1	Installation & Setup	86
21.2.2	Basic Usage	86

21.2.3	Results & Analysis	86
21.2.4	Performance	87
21.2.5	Customization	87
21.2.6	Troubleshooting	87
21.3	Reading Order Recommendations	87
21.3.1	For First-Time Users	87
21.3.2	For Regular Users	87
21.3.3	For Power Users	87
21.4	Search by Keyword	88
21.4.1	A-C	88
21.4.2	D-F	88
21.4.3	G-M	88
21.4.4	N-R	88
21.4.5	S-Z	89
21.5	Content Statistics	89
21.5.1	Total Documentation	89
21.5.2	Reading Time	89
21.5.3	Coverage	89
21.6	Learning Resources	89
21.6.1	Tutorials	89
21.6.2	Reference	90
21.6.3	Examples	90
21.7	Getting Help	90
21.7.1	Self-Service	90
21.7.2	Community Support	90
21.8	Updates	90
21.8.1	Version 1.0.0 (October 10, 2025)	90
21.8.2	Future Plans	90
21.9	Contributing	91
21.10	License	91
21.11	Complete Documentation Suite	91
21.11.1	Quick Links	91

1 About This Documentation

This comprehensive documentation covers all aspects of K-talysticFlow (KAST), an automated deep learning pipeline for molecular bioactivity prediction and virtual screening.

Project Information:

- **Name:** K-talysticFlow (KAST - K-atalystic Automated Screening Taskflow)
- **Version:** 1.0.0 (Stable Release)
- **Release Date:** October 10, 2025
- **Developer:** Késsia Souza Santos (@kelsouzs)
- **Institution:** Laboratory of Molecular Modeling, UEFS
- **Funding:** CNPq
- **License:** MIT License

Documentation Statistics:

- **Total Pages:** 12 main sections
- **Total Content:** ~48,800 words
- **Code Examples:** 150+
- **Tables:** 70+
- **Diagrams:** 25+

How to Use This Document:

- **Beginners:** Start with sections 1-3 (Introduction, Installation, User Manual)
- **Regular Users:** Focus on sections 4-7 (Pipeline, Performance, Analysis, Configuration)
- **Advanced Users:** Review sections 8-10 (FAQ, Troubleshooting, Advanced Topics)
- **Reference:** Use section 11-12 (Quick Reference, Index)

2 Home

3 KAST Wiki

3.1 K-atalytic 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

3.2 What is K-talysticFlow?

K-talysticFlow or **K-atalytic Automated Screening Taskflow** (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**, **RD-Kit**, and **TensorFlow**, it provides an end-to-end solution for computational drug discovery and virtual screening.

Developed at: [Laboratory of Molecular Modeling \(LMM-UEFS\)](#)

Funding: CNPq

Current Version: 1.0.0 (Stable Release - October 10, 2025)

3.3 Key Features

3.3.1 Core Capabilities

- **Fully Automated:** Interactive menu-driven interface for a seamless workflow
- **Deep Learning Model:** Multi-Layer Perceptron (MLP) trained on Morgan Fingerprints (ECFP)
- **Comprehensive Validation Suite:** Rigorous model assessment including:
 - ROC/AUC Analysis
 - Enrichment Factor Calculation
 - k-fold Cross-Validation with Scaffold Splitting
 - Tanimoto Similarity Analysis
 - Learning Curve Generation
- **Complete End-to-End Pipeline:** From raw SMILES data to actionable predictions
- **Cross-Platform:** Compatible with Windows and Linux
- **Analysis-Ready Outputs:** Clear reports, graphs, and CSV files

3.3.2 Advanced Features

- **Parallel Processing:** Multi-core support for 5-10x faster performance
 - Automatic CPU detection
 - Memory-efficient batch processing
 - Configurable worker allocation
- **K-Prediction Score:** Proprietary scoring system for ranking molecular activity
- **Comprehensive Logging:** Daily log rotation with detailed error tracking
- **Quality Assurance:** Built-in dependency checker and test suite

- **Flexible Configuration:** Centralized settings management in `settings.py`

3.4 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
Parallel Processing	Configuration and optimization guide
Output Analysis	How to interpret results and K-Prediction scores
Configuration Guide	Customize pipeline settings
FAQ	Frequently asked questions
Troubleshooting	Common issues and solutions
API Reference	Function and module documentation

3.5 Pipeline Overview

Diagram (see online documentation for interactive version)

```
graph TD
    A[ Raw SMILES Data] --> B[1 Data Preparation & Split]
    B --> C[2 Featurization]
    C --> D[3 Model Training]
    D --> E[4 Model Validation Suite]
    E --> F[5 Predictions on New Data]
    F --> G[ Ranked Hits & Analysis]

    subgraph "Validation Suite"
        E --> H[ Cross-Validation]
        E --> I[ Enrichment Factor]
        E --> J[ ROC Analysis & Metrics]
        E --> K[ Similarity Analysis]
        E --> L[ Learning Curve]
    end

    end

    style A fill:#e1f5ff
    style G fill:#d4edda
    style E fill:#fff3cd
```

3.6 Pipeline Steps

3.6.1 1 Data Preparation (1_preparation.py)

- Imports SMILES from active and inactive compound files
- Validates molecular structures
- Balances datasets
- Performs Scaffold Splitting for train/test sets
- Labels molecules (active=1, inactive=0)

Outputs: 01_train_set.csv, 01_test_set.csv

3.6.2 2 Featurization (2_featurization.py)

- Converts SMILES to Morgan Fingerprints (ECFP)
- Configurable radius (default: 3) and size (default: 2048 bits)
- **Parallel processing enabled** (5-10x speedup)
- Memory-efficient sparse matrix handling
- DeepChem format output

Outputs: featurized_datasets/train/, featurized_datasets/test/, 02_featurization_log.txt

3.6.3 3 Model Training (3_training.py)

- Trains DeepChem MultitaskClassifier (Neural Network)
- Architecture: [1000, 500] hidden layers
- Dropout: 0.25, Learning Rate: 0.001
- TensorFlow backend
- Automatic checkpoint saving
- Training metrics logging

Outputs: trained_model/checkpoint1.pt, training_metadata.json, 03_training_log.txt

3.6.4 4 Model Evaluation Suite

3.6.4.1 4.0 Main Evaluation (4_0_evaluation_main.py)

- ROC/AUC analysis
- Accuracy, Precision, Recall, F1-Score
- Confusion Matrix
- Performance metrics on test set

Outputs: 4_0_evaluation_report.txt, 4_0_test_predictions.csv

3.6.4.2 4.1 Cross-Validation (4_1_cross_validation.py)

- k-fold stratified cross-validation (default: 5 folds)
- Mean and standard deviation of metrics

- Scaffold-based splitting for chemical diversity

Outputs: 4_1_cross_validation_results.txt

3.6.4.3 4.2 Enrichment Factor (4_2_enrichment_factor.py)

- Evaluates screening performance
- Calculates EF at multiple cutoffs (1%, 2%, 5%, 10%)
- Validates virtual screening capability

Outputs: 4_2_enrichment_factor_results.txt

3.6.4.4 4.3 Tanimoto Similarity (4_3_tanimoto_similarity.py)

- Chemical space analysis
- Diversity metrics
- Similarity distribution plots
- **Parallel processing enabled** (3-5x speedup)

Outputs: 4_3_tanimoto_similarity_results.txt, similarity plots

3.6.4.5 4.4 Learning Curve (4_4_learning_curve.py)

- Training set size vs. performance
- Identifies optimal dataset size
- Diagnoses overfitting/underfitting
- **Parallel processing enabled** (4-8x speedup)

Outputs: 4_4_learning_curve_results.txt, learning curve plots

3.6.5 5 Prediction on New Molecules

3.6.5.1 5.0 Featurization (5_0_featurize_for_prediction.py)

- Converts new SMILES to fingerprints
- **Parallel processing enabled** (5-10x speedup)
- Prepares data for prediction

Outputs: prediction_featurized/, 5_0_featurization_report.txt

3.6.5.2 5.1 Run Prediction (5_1_run_prediction.py)

- Predicts activity for new molecules
- Calculates **K-Prediction Score** (proprietary ranking metric)
- Ranks molecules by predicted activity
- Exports results for analysis

Outputs: 5_1_new_molecule_predictions.csv

3.7 Getting Started

3.7.1 Quick Start Guide

1. **Install K-talysticFlow** - Set up your environment
2. **Prepare your data** - Place SMILES files in `data/` folder
3. **Run the pipeline** - Execute `python main.py` and follow the menu
4. **Analyze results** - Interpret your predictions

3.7.2 Recommended Workflow

1. Check environment

`python main.py` → Option `[8]` → `[1]` Check Dependencies

2. Run full pipeline

`python main.py` → Option `[7]` Run Complete Pipeline

3. Analyze outputs

Check `results/` folder for reports, plots, and predictions

3.8 System Requirements

3.8.1 Minimum Requirements

- **Python:** 3.9+
- **RAM:** 8 GB
- **CPU:** Dual-core processor
- **Disk:** 2 GB free space

3.8.2 Recommended Requirements

- **Python:** 3.10+
- **RAM:** 16 GB+
- **CPU:** Quad-core or better (for parallel processing)
- **Disk:** 5 GB+ free space
- **GPU:** Optional (TensorFlow GPU support)

3.8.3 Key Dependencies

- RDKit
 - DeepChem
 - TensorFlow
 - Scikit-learn
 - Pandas, NumPy
 - Matplotlib, Seaborn
 - Joblib (parallel processing)
 - TQDM (progress bars)
-

3.9 Citation

If you use K-talysticFlow in your research, please cite:

```
@software{kast2025,  
  author = {Santos, Késsia Souza},  
  title = {K-talysticFlow: Automated Deep Learning Pipeline for Molecular Screening},  
  year = {2025},  
  version = {1.0.0},  
  url = {https://github.com/kelsouzs/kast},  
  institution = {Laboratory of Molecular Modeling, UEFS}  
}
```

3.10 Support & Contact

- **GitHub Issues:** [Report bugs or request features](#)
 - **Email:** lmm@uefs.br
 - **LinkedIn:** [@kelsouzs](#)
 - **GitHub:** [@kelsouzs](#)
 - **Wiki:** Browse this documentation for detailed guides
-

3.11 License

This project is licensed under the **MIT License** - see the [LICENSE](#) file for details.

3.12 Acknowledgments

- **Funding:** CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico)
 - **Institution:** Laboratory of Molecular Modeling (LMM-UEFS)
 - **Community:** DeepChem, RDKit, and TensorFlow teams
-

Version: 1.0.0 | **Last Updated:** October 10, 2025 *Made with for the computational chemistry community* **Developer:** Késsia Souza Santos (@kelsouzs)

4 Installation

5 Installation Guide

This guide will walk you through setting up K-talysticFlow on your system.

5.1 Prerequisites

Before installing K-talysticFlow, ensure you have:

- **Python 3.9 or higher** installed
 - **Conda** (recommended for managing dependencies)
 - **Git** (for cloning the repository)
 - At least **8 GB RAM** (16 GB+ recommended)
 - **2-5 GB** free disk space
-

5.2 Step-by-Step Installation

5.2.1 1. Clone the Repository

```
# Clone from GitHub  
git clone https://github.com/kelsouzs/kast.git  
cd kast
```

Or download and extract the ZIP file from GitHub.

5.2.2 2. Create a Conda Environment (Recommended)

Using Conda ensures clean dependency management and avoids conflicts:

```
# Create environment with Python 3.10  
conda create -n kast python=3.10 -y  
  
# Activate the environment  
conda activate kast
```

Alternative: Using venv (if not using Conda)

```
python -m venv kast_env  
source kast_env/bin/activate # Linux/Mac  
kast_env\Scripts\activate # Windows
```

5.2.3 3. Install RDKit via Conda (Recommended Method)

RDKit is easiest to install through Conda:


```
conda install -c conda-forge rdkit -y
```

5.2.4 4. Install Python Dependencies

Install all required packages from requirements.txt:

```
pip install -r requirements.txt
```

This will install: - **DeepChem** (deep learning for chemistry) - **TensorFlow** (neural network backend) - **Scikit-learn** (machine learning utilities) - **Pandas**, **NumPy** (data manipulation) - **Matplotlib**, **Seaborn** (visualization) - **Joblib** (parallel processing) - **TQDM** (progress bars)

5.2.5 5. Verify Installation

Run the built-in environment checker:

```
python main.py
```

Then select: - **Option [8]** - Advanced Options - **Option [1]** - Check Environment & Dependencies

The checker will verify: - All required packages are installed - Correct versions - Import functionality - System compatibility

Expected Output:

```
=====
                        K-talysticFlow Dependency Checker
=====

Python version: 3.10.12 (OK)
RDKit: 2023.09.1 (OK)
DeepChem: 2.7.1 (OK)
TensorFlow: 2.15.0 (OK)
...
=====
All dependencies are correctly installed!
=====
```

5.3 Platform-Specific Instructions

5.3.1 Linux

```
# Install system dependencies (Ubuntu/Debian)
sudo apt-get update
sudo apt-get install python3-dev build-essential

# Then follow steps 2-5 above
```

5.3.2 Windows

```
# Use Anaconda Prompt or PowerShell  
# Ensure conda is in PATH  
  
# Then follow steps 2-5 above
```

5.3.3 macOS

```
# Install Xcode command line tools  
xcode-select --install  
  
# Then follow steps 2-5 above
```

5.4 Docker Installation (Alternative)

Coming soon! Docker image for easy deployment.

5.5 Test Your Installation

5.5.1 Quick Test

Run the parallel processing test suite:

```
python main.py  
# Select [8] Advanced Options → [2] Test Parallel Processing
```

This runs 6 comprehensive tests to verify: - Basic parallelism - Large dataset handling - Memory efficiency - Error handling - Performance benchmarks

5.6 Optional: GPU Support

To enable GPU acceleration for TensorFlow:

5.6.1 NVIDIA GPU

```
# Install CUDA and cuDNN (follow NVIDIA docs)  
# Then install TensorFlow GPU  
pip install tensorflow[and-cuda]
```

5.6.2 Verify GPU

```
import tensorflow as tf
print("GPUs Available:", tf.config.list_physical_devices('GPU'))
```

5.7 requirements.txt Contents

```
deepchem>=2.7.0
rdkit>=2022.9.5
tensorflow>=2.10.0
scikit-learn>=1.2.0
pandas>=1.5.0
numpy>=1.23.0
matplotlib>=3.6.0
seaborn>=0.12.0
joblib>=1.2.0
tqdm>=4.65.0
```

5.8 Updating K-talysticFlow

To update to the latest version:

```
# Pull latest changes
git pull origin main

# Update dependencies
pip install -r requirements.txt --upgrade
```

5.9 Uninstallation

To remove K-talysticFlow:

```
# Deactivate environment
conda deactivate

# Remove conda environment
conda env remove -n kast

# Delete repository folder
cd ..
rm -rf kast # Linux/Mac
rmdir /s kast # Windows
```

5.10 Troubleshooting Installation

5.10.1 Issue: RDKit installation fails

Solution: Use Conda instead of pip:

```
conda install -c conda-forge rdkit
```

5.10.2 Issue: TensorFlow errors

Solution: Install specific version:

```
pip install tensorflow==2.15.0
```

5.10.3 Issue: Memory errors during installation

Solution: Install packages one by one:

```
pip install deepchem  
pip install tensorflow  
# ... etc
```

5.10.4 Issue: Permission denied (Linux/Mac)

Solution: Use --user flag:

```
pip install -r requirements.txt --user
```

5.11 Next Steps

Once installation is complete:

1. **Read the User Manual** - Learn how to use K-talysticFlow
2. **Configure Settings** - Customize for your needs
3. **Run Your First Analysis** - Get started!

5.12 Need Help?

- Check [FAQ](#) for common questions
- See [Troubleshooting](#) for known issues
- **Open an issue** on GitHub
- **Contact:** lmm@uefs.br

← Back to Wiki Home

6 User Manual

7 User Manual

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

7.1 Table of Contents

1. Quick Start
 2. Preparing Your Data
 3. Running the Pipeline
 4. Menu Options
 5. Understanding Outputs
 6. Best Practices
-

7.2 Quick Start

7.2.1 First Run

1. **Activate your environment:**

```
conda activate kast
```

2. **Navigate to project directory:**

```
cd path/to/kast
```

3. **Launch the control panel:**

```
python main.py
```

4. **You'll see the main menu:**

```
=====
K-talysticFlow (KAST) Control Panel v1.0.0
=====
K-atalystic Automated Screening Taskflow
Automated Deep Learning for Molecular Screening
-----
```

- ```
[1] Data Preparation (Split Train/Test)
[2] Featurize Molecules (Generate Fingerprints)
[3] Train Model
[4] Evaluate Model (Multiple Options)
[5] Featurize New Molecules for Prediction
[6] Run Predictions on New Molecules
[7] Run Complete Pipeline (Steps 1-4)
[8] Advanced Options
[0] Exit
```

Parallel Processing: ENABLED (6 workers)

---

Enter your choice:

---

## 7.3 Preparing Your Data

### 7.3.1 Data Format

K-talysticFlow requires SMILES format files:

#### 7.3.1.1 Active Compounds (`actives.smi`)

CC(C)Cc1ccc(cc1)C(C)C(=O)O ibuprofen  
CN1C=NC2=C1C(=O)N(C(=O)N2C)C caffeine

#### 7.3.1.2 Inactive Compounds (`inactives.smi`)

CC(=O)OC1=CC=CC=C1C(=O)O aspirin  
CCCCCCCCCCCCCCCC hexadecane

### 7.3.2 File Requirements

- **Format:** `.smi` or `.smiles`
- **Structure:** SMILES [space] optional\_name
- **Location:** Place files in `data/` folder
- **Names:**
  - `actives.smi` for active compounds
  - `inactives.smi` for inactive compounds
  - `zinc_library.smi` (or any name) for prediction

### 7.3.3 Data Quality Guidelines

**Good practices:** - Use canonicalized SMILES - Remove duplicates - Validate structures - Balance active/inactive ratio (1:1 to 1:10) - Minimum 50 molecules per class - Maximum 100,000 molecules total

**Avoid:** - Invalid SMILES - Salts/mixtures (unless intended) - Very small molecules (< 5 atoms)  
- Very large molecules (> 200 atoms)

---

## 7.4 Running the Pipeline

### 7.4.1 Option 1: Complete Pipeline (Recommended for First Use)

**Menu Option [7]** - Runs all steps automatically:

1. Data Preparation
2. Featurization
3. Model Training

#### 4. Full Evaluation Suite

Enter your choice: 7

**What happens:** - Splits data into train/test sets (70/30) - Generates molecular fingerprints - Trains neural network model - Runs all validation tests - Time: ~10-30 minutes (depends on dataset size)

---

#### 7.4.2 Option 2: Step-by-Step Workflow

For more control, run each step individually:

Enter your choice: 1

**7.4.2.1 Step 1: Data Preparation [1] What it does:** - Reads `actives.smi` and `inactives.smi` - Validates SMILES structures - Balances dataset - Splits using Scaffold Splitting (70/30) - Labels: active=1, inactive=0

**Outputs:** - `results/01_train_set.csv` - `results/01_test_set.csv`

**Time:** 1-5 minutes

---

Enter your choice: 2

**7.4.2.2 Step 2: Featurization [2] What it does:** - Converts SMILES to Morgan Fingerprints (ECFP) - Radius: 3, Size: 2048 bits - Uses parallel processing (5-10x faster) - Creates DeepChem datasets

**Outputs:** - `results/featurized_datasets/train/` - `results/featurized_datasets/test/` - `results/02_featurization_log.txt`

**Time:** 2-10 minutes (parallel) | 10-60 minutes (sequential)

---

Enter your choice: 3

**7.4.2.3 Step 3: Model Training [3] What it does:** - Trains Multi-Layer Perceptron (MLP) - Architecture: Input  $\rightarrow$  1000  $\rightarrow$  500  $\rightarrow$  Output - 50 epochs, dropout 0.25 - TensorFlow backend

**Outputs:** - `results/trained_model/checkpoint1.pt` - `results/training_metadata.json` - `results/03_training_log.txt`

**Time:** 5-20 minutes

---

Enter your choice: 4

#### 7.4.2.4 Step 4: Model Evaluation [4] Submenu appears:

- [1] Main Evaluation Report (AUC, Accuracy, etc.)
- [2] Cross-Validation
- [3] Enrichment Factor
- [4] Tanimoto Similarity Analysis
- [5] Learning Curve Generation
- [6] Run All Evaluation Scripts
- [0] Back to Main Menu

**Recommended:** Choose [6] to run all evaluations

**Outputs:** - results/4\_0\_evaluation\_report.txt - results/4\_1\_cross\_validation\_results.txt  
- results/4\_2\_enrichment\_factor\_results.txt - results/4\_3\_tanimoto\_similarity\_results.txt  
- results/4\_4\_learning\_curve\_results.txt - Various plots in results/

**Time:** 10-40 minutes

---

#### 7.4.2.5 Step 5-6: Predictions on New Molecules

Enter your choice: 5

**7.4.2.5.1 Step 5: Featurize New Molecules [5] Requirements:** - Place your SMILES file in data/ folder - Update PREDICTION\_SMILES\_FILE in settings.py if needed

**What it does:** - Reads new molecules - Generates fingerprints (same parameters as training) - Prepares for prediction

**Outputs:** - results/prediction\_featurized/ - results/5\_0\_featurization\_report.txt

**Time:** 1-15 minutes (depends on dataset size)

---

Enter your choice: 6

**7.4.2.5.2 Step 6: Run Predictions [6] What it does:** - Loads trained model - Predicts activity for each molecule - Calculates **K-Prediction Score** - Ranks molecules by score

**Outputs:** - results/5\_1\_new\_molecule\_predictions.csv

**Time:** 1-5 minutes

**CSV Format:**

SMILES,Probability,K\_Prediction\_Score,Rank  
CC(C)Cc1ccc(cc1)C(C)C,0.89,89.0,1



```
CN1C=NC2=C1C(=O)N,0.45,45.0,2
...
```

---

## 7.5 Advanced Options [8]

```
Enter your choice: 8
```

**Submenu:**

- [1] Check Environment & Dependencies
- [2] Test Parallel Processing Compatibility
- [3] Configure Parallel Processing Workers
- [0] Back to Main Menu

### 7.5.1 [1] Check Dependencies

Verifies all required packages are installed correctly.

**Use when:** - First installation - After updating packages - Troubleshooting errors

---

### 7.5.2 [2] Test Parallel Processing

Runs 6 comprehensive tests: 1. Basic parallel execution 2. Large array processing 3. Memory efficiency 4. Error handling 5. Performance benchmark 6. Worker scaling

**Use when:** - Configuring optimal workers - Diagnosing performance issues - Verifying multi-core support

---

### 7.5.3 [3] Configure Workers

Adjust CPU core allocation on-the-fly without editing files.

**Example:**

```
Current: N_WORKERS = 6
Enter new value (None/auto, -1/all, or number): 4
Updated to 4 workers
```

---

## 7.6 Understanding Outputs

### 7.6.1 Key Output Files

| File             | Description                    |
|------------------|--------------------------------|
| 01_train_set.csv | Training molecules with labels |
| 01_test_set.csv  | Test molecules with labels     |

| File                             | Description                     |
|----------------------------------|---------------------------------|
| 4_0_evaluation_report.txt        | Main performance metrics        |
| 4_0_test_predictions.csv         | Test set predictions            |
| 5_1_new_molecule_predictions.csv | <b>Final ranked predictions</b> |

## 7.6.2 Understanding K-Prediction Score

**K-Prediction Score = Probability  $\times$  100**

- **Score 90-100:** Very likely active (high confidence)
- **Score 70-89:** Likely active (medium-high confidence)
- **Score 50-69:** Possibly active (medium confidence)
- **Score 30-49:** Possibly inactive (medium-low confidence)
- **Score 0-29:** Likely inactive (low confidence)

**Interpretation:** - Focus on top-ranked molecules (highest scores) - Scores  $> 70$  are good candidates for experimental validation - Consider enrichment factor when prioritizing hits

See **Output Analysis** for detailed interpretation.

---

## 7.7 Best Practices

### 7.7.1 Do's

1. **Always run validation suite** before predictions
2. **Check evaluation metrics** (AUC  $> 0.7$  is good)
3. **Use balanced datasets** (similar active/inactive counts)
4. **Enable parallel processing** for large datasets
5. **Review logs** in results/logs/ for errors
6. **Backup your model** in results/trained\_model/
7. **Document your workflow** and parameter changes

### 7.7.2 Don'ts

1. **Don't skip validation steps** - you need to know model quality
  2. **Don't use very imbalanced data** (e.g., 1:100 ratio)
  3. **Don't ignore low AUC scores** ( $< 0.6$  = poor model)
  4. **Don't modify trained model files** manually
  5. **Don't delete featurized datasets** if retraining
  6. **Don't run multiple instances** on same results folder
- 

## 7.8 Typical Workflows

### 7.8.1 Workflow A: New Project

1. Prepare data files  $\rightarrow$  Place in data/
2. Run Option [7]  $\rightarrow$  Complete Pipeline

3. Check results → Review metrics
  4. If AUC > 0.7 → Proceed to predictions
  5. Run Options [5] + [6] → Predict new molecules
  6. Analyze outputs → Select top hits
- 

### 7.8.2 Workflow B: Model Optimization

1. Run initial pipeline → Option [7]
  2. Check learning curve → Option [4] [5]
  3. Adjust parameters in settings.py → If needed
  4. Retrain → Option [3]
  5. Re-evaluate → Option [4] [6]
  6. Compare metrics → Iterate if necessary
- 

### 7.8.3 Workflow C: Batch Predictions

1. Train model once → Options [1] [2] [3]
  2. Validate thoroughly → Option [4] [6]
  3. For each new library:
    - a. Place .smi file in data/
    - b. Update settings.py
    - c. Run Options [5] + [6]
    - d. Collect predictions
- 

## 7.9 Time Estimates

| Task          | Small Dataset    | Medium Dataset | Large Dataset |
|---------------|------------------|----------------|---------------|
|               | (< 1K molecules) | (1K-10K)       | (10K-100K)    |
| Preparation   | < 1 min          | 1-2 min        | 2-5 min       |
| Featurization | 1-2 min          | 5-10 min       | 10-30 min     |
| Training      | 2-5 min          | 5-10 min       | 10-20 min     |
| Evaluation    | 5-10 min         | 10-20 min      | 20-40 min     |
| Prediction    | < 1 min          | 1-5 min        | 5-15 min      |

*Times assume parallel processing enabled with 4-8 cores*

---

## 7.10 Need Help?

- [FAQ](#) - Frequently asked questions
- [Troubleshooting](#) - Common issues
- [Configuration Guide](#) - Customize settings

- **GitHub Issues** - Report bugs
- **Email:** kelsouzs.uefs@gmail.com

---

← Back to Wiki Home | Next: Pipeline Steps →

## 8 Pipeline Steps

## 9 Pipeline Steps - Detailed Documentation

Complete technical documentation of each K-talysticFlow pipeline step.

---

### 9.1 Table of Contents

1. [Overview](#)
  2. Step 1: Data Preparation
  3. Step 2: Featurization
  4. Step 3: Model Training
  5. Step 4: Model Evaluation
  6. Step 5-6: Prediction
- 

### 9.2 Overview

K-talysticFlow pipeline consists of 6 main steps:

# Diagram (see online documentation for interactive version)

```
graph LR
 A[1. Preparation] --> B[2. Featurization]
 B --> C[3. Training]
 C --> D[4. Evaluation]
 D --> E[5. Feat. Prediction]
 E --> F[6. Run Prediction]
```

---

### 9.3 Step 1: Data Preparation

**Script:** bin/1\_preparation.py

**Menu Option:** [1]

**Purpose:** Clean, validate, and split molecular data

#### 9.3.1 Input

- data/actives.smi - Active compounds
- data/inactives.smi - Inactive compounds

**Format:**

SMILES [space] optional\_name

CC(C)Cc1ccc(cc1)C(C)C(O)=O ibuprofen

---

### 9.3.2 Process

```
Read SMILES files
actives = pd.read_csv('actives.smi', sep='\t', header=None)
inactives = pd.read_csv('inactives.smi', sep='\t', header=None)
```

#### 9.3.2.1 1.1 Data Loading

---

**9.3.2.2 1.2 SMILES Validation** Checks performed: - Valid SMILES syntax (RDKit parsing) - Length constraints (5-200 characters) - Duplicate removal - Sanitization

**Example validation:**

```
from rdkit import Chem

def validate_smiles(smiles):
 mol = Chem.MolFromSmiles(smiles)
 if mol is None:
 return False # Invalid
 return True
```

---

**9.3.2.3 1.3 Data Balancing** Balances active/inactive ratio to prevent class imbalance:

**Strategies:** - If actives > inactives: Undersample actives - If inactives > actives: Undersample inactives - **Target:** 1:1 ratio (configurable)

---

**9.3.2.4 1.4 Labeling** Assigns binary labels: - **Actives:** active = 1 - **Inactives:** active = 0

**Output DataFrame:**

|   | smiles                                  | active |
|---|-----------------------------------------|--------|
| 0 | <chem>CC(C)Cc1ccc(cc1)C(C)C(O)=O</chem> | 1      |
| 1 | <chem>CCCCCCCCCCCCCCCC</chem>           | 0      |

---

**9.3.2.5 1.5 Scaffold Splitting** Algorithm: DeepChem's ScaffoldSplitter

**Purpose:** - Ensures train/test sets have different molecular scaffolds - Better test of generalization - More realistic than random splitting

**How it works:** 1. Generate Bemis-Murcko scaffold for each molecule 2. Group molecules by scaffold 3. Split scaffolds (not individual molecules) into train/test

**Visual:**

Training Set Scaffolds: [A, B, C]  
Test Set Scaffolds: [D, E]

### Parameters:

```
TEST_SET_FRACTION = 0.3 # 30% test, 70% train
RANDOM_STATE = 42 # Reproducibility
```

---

### 9.3.3 Output

Files created: - results/01\_train\_set.csv - results/01\_test\_set.csv

#### Format:

```
smiles,active
CC(C)Cc1ccc(cc1)C(C)C(O)=O,1
CN1C=NC2=C1C(=O)N(C)C(=O)N2C,1
CCCCCCCCCCCCCCCC,0
```

#### Statistics logged:

```
Total molecules: 10,000
Active compounds: 5,000
Inactive compounds: 5,000
Training set: 7,000 (3,500 active, 3,500 inactive)
Test set: 3,000 (1,500 active, 1,500 inactive)
```

---

### 9.3.4 Configuration

Key settings in settings.py:

```
TEST_SET_FRACTION = 0.3
RANDOM_STATE = 42
MIN_MOLECULES_PER_CLASS = 50
MAX_MOLECULES_TOTAL = 100000
MIN_SMILES_LENGTH = 5
MAX_SMILES_LENGTH = 200
```

---

### 9.3.5 Troubleshooting

**Issue:** “Insufficient molecules”

```
MIN_MOLECULES_PER_CLASS = 20 # Lower threshold
```

**Issue:** “Invalid SMILES” - Check SMILES syntax - Remove invalid entries manually

---

## 9.4 Step 2: Featurization

**Script:** bin/2\_featurization.py

**Menu Option:** [2]

**Purpose:** Convert SMILES to numerical fingerprints

#### 9.4.1 Input

- results/01\_train\_set.csv
  - results/01\_test\_set.csv
- 

#### 9.4.2 Process

**9.4.2.1 2.1 Morgan Fingerprint Generation Algorithm:** Extended Connectivity Fingerprints (ECFP)

**Parameters:**

```
FP_RADIUS = 3 # ECFP6 (radius × 2)
FP_SIZE = 2048 # Number of bits
```

**How it works:** 1. For each atom, identify circular substructures up to radius 2. Hash each substructure to a bit position 3. Set corresponding bits to 1 in 2048-bit vector

**Example:**

SMILES: CCO (ethanol)

Substructures at radius 3:

- [CH3]-C
- C-[CH2]-O
- [CH2]-[OH]

→ Hashed to bits: [45, 234, 567, ...]

→ Fingerprint: [0,0,0,...,1(bit 45),...,1(bit 234),...,1(bit 567),...]

---

**9.4.2.2 2.2 Parallel Processing** If enabled (ENABLE\_PARALLEL\_PROCESSING = True):

```
from joblib import Parallel, delayed

def generate_fp(smiles):
 mol = Chem.MolFromSmiles(smiles)
 fp = AllChem.GetMorganFingerprintAsBitVect(
 mol, radius=FP_RADIUS, nBits=FP_SIZE
)
 return np.array(fp)

Parallel execution
fingerprints = Parallel(n_jobs=N_WORKERS)(
 delayed(generate_fp)(smiles) for smiles in smiles_list
)
```

**Speedup:** 5-10x faster for large datasets

---



#### 9.4.2.3 2.3 DeepChem Dataset Creation

Converts numpy arrays to DeepChem format:

```
dataset = dc.data.NumpyDataset(
 X=fingerprints, # Feature matrix (N × 2048)
 y=labels, # Labels (N × 1)
 ids=smiles_list # Molecule IDs
)
```

---

#### 9.4.2.4 2.4 Sparse Matrix Optimization

For memory efficiency:

```
from scipy.sparse import csr_matrix

Convert to sparse matrix (most bits are 0)
X_sparse = csr_matrix(fingerprints)
```

Memory saved: ~80-90% for typical fingerprints

---

### 9.4.3 Output

Directory structure:

```
results/featurized_datasets/
 train/
 metadata.csv.gz
 shard-0-ids.npy
 shard-0-w.npy
 shard-0-X.npy # Fingerprints
 shard-0-y.npy # Labels
 tasks.json
 test/
 (same structure)
```

Log file: results/02\_featurization\_log.txt

Content:

Featurization Log

=====

Date: 2025-10-10 14:30:22

Fingerprint Type: Morgan (ECFP)

Radius: 3

Size: 2048 bits

Training Set:

Molecules: 7,000

Time: 45.3 seconds

Parallel: Yes (6 workers)

Test Set:  
Molecules: 3,000  
Time: 19.2 seconds

---

#### 9.4.4 Configuration

```
FP_SIZE = 2048
FP_RADIUS = 3
ENABLE_PARALLEL_PROCESSING = True
N_WORKERS = None
PARALLEL_BATCH_SIZE = 100000
```

---

#### 9.4.5 Troubleshooting

**Issue:** Memory error

```
PARALLEL_BATCH_SIZE = 50000
ENABLE_PARALLEL_PROCESSING = False
```

**Issue:** Too slow

```
ENABLE_PARALLEL_PROCESSING = True
N_WORKERS = -1
```

---

### 9.5 Step 3: Model Training

**Script:** bin/3\_training.py

**Menu Option:** [3]

**Purpose:** Train deep neural network classifier

#### 9.5.1 Input

- results/featurized\_datasets/train/
  - results/featurized\_datasets/test/ (for validation)
- 

#### 9.5.2 Process

##### 9.5.2.1 3.1 Model Architecture **Type:** Multi-Layer Perceptron (MLP)

**Default architecture:**

Input (2048) → Dense(1000) → Dropout(0.25) → Dense(500) → Dropout(0.25) → Output(1)

**Visual:**

```

[2048 bits]
↓
[1000 neurons, ReLU]
↓
[Dropout 25%]
↓
[500 neurons, ReLU]
↓
[Dropout 25%]
↓
[1 neuron, Sigmoid] → Probability

```

---

```

MODEL_PARAMS = {
 'n_tasks': 1, # Binary classification
 'layer_sizes': [1000, 500], # Hidden layer sizes
 'dropouts': 0.25, # Regularization
 'learning_rate': 0.001, # Adam optimizer
 'mode': 'classification',
 'nb_epoch': 50 # Training epochs
}

```

### 9.5.2.2 3.2 Model Configuration

---

### 9.5.2.3 3.3 Loss Function Binary Cross-Entropy:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Where: -  $y_i$  = true label (0 or 1) -  $\hat{y}_i$  = predicted probability -  $N$  = number of samples

---

```

model = dc.models.MultitaskClassifier(**MODEL_PARAMS)

for epoch in range(NB_EPOCH):
 # Train on batches
 loss = model.fit(train_dataset, nb_epoch=1)

 # Validate
 val_loss = model.evaluate(test_dataset)

 # Log progress
 print(f"Epoch {epoch+1}/{NB_EPOCH}: Loss={loss:.4f}")

```

### 9.5.2.4 3.4 Training Loop

---

**9.5.2.5 3.5 Optimization** **Optimizer:** Adam - Adaptive learning rate - Momentum-based - Good for noisy gradients

**Regularization:** - **Dropout:** Randomly disables 25% of neurons during training - **Early stopping:** (optional, not default)

---

**9.5.2.6 3.6 Checkpointing** Saves model after training:

```
model.save_checkpoint(model_dir='results/trained_model/')
```

---

## 9.5.3 Output

Files created:

1. **Model checkpoint:**
  - results/trained\_model/checkpoint1.pt
  - Contains trained weights
2. **Metadata:**
  - results/training\_metadata.json

```
{
 "model_type": "MultitaskClassifier",
 "layer_sizes": [1000, 500],
 "dropouts": 0.25,
 "learning_rate": 0.001,
 "nb_epoch": 50,
 "training_date": "2025-10-10",
 "fingerprint_size": 2048,
 "fingerprint_radius": 3
}
```

3. **Training log:**
  - results/03\_training\_log.txt

Training Log

=====

Architecture: [1000, 500]

Epochs: 50

Epoch 1/50: Loss=0.6234

Epoch 2/50: Loss=0.5423

...

Epoch 50/50: Loss=0.1245

Training completed in 8.3 minutes

---

### 9.5.4 Configuration

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

Customization examples:

Deeper network:

```
'layer_sizes': [2048, 1024, 512, 256]
```

Prevent overfitting:

```
'dropouts': 0.5,
'nb_epoch': 30
```

---

### 9.5.5 Troubleshooting

**Issue:** Loss not decreasing

```
'learning_rate': 0.0001 # Reduce
```

**Issue:** Overfitting

```
'dropouts': 0.5, # Increase
'nb_epoch': 30 # Reduce
```

**Issue:** Training too slow

```
'nb_epoch': 30 # Reduce epochs
'layer_sizes': [512, 256] # Simpler model
```

---

## 9.6 Step 4: Model Evaluation

**Scripts:** bin/4\_\*.py

**Menu Option:** [4]

**Purpose:** Comprehensive model validation

### 9.6.1 4.0 Main Evaluation

**Script:** bin/4\_0\_evaluation\_main.py

### 9.6.1.1 Metrics Calculated

1. **ROC-AUC** (Receiver Operating Characteristic)
2. **Accuracy**
3. **Precision** (Positive Predictive Value)
4. **Recall** (Sensitivity)
5. **F1-Score**
6. **Confusion Matrix**

### 9.6.1.2 Output results/4\_0\_evaluation\_report.txt:

Model Evaluation Report

=====

Test Set Performance:

ROC-AUC: 0.8523

Accuracy: 82.34%

Precision: 0.78

Recall: 0.85

F1-Score: 0.81

Confusion Matrix:

|                 | Predicted Negative | Predicted Positive |
|-----------------|--------------------|--------------------|
| Actual Negative | 850                | 120                |
| Actual Positive | 80                 | 450                |

---

## 9.6.2 4.1 Cross-Validation

Script: bin/4\_1\_cross\_validation.py

### 9.6.2.1 Process k-fold stratified cross-validation (default k=5):

```
for fold in range(N_FOLDS):
 # Split data
 train_fold, val_fold = split(dataset, fold)

 # Train model
 model.fit(train_fold)

 # Evaluate
 auc_fold = evaluate(val_fold)
```

### 9.6.2.2 Output results/4\_1\_cross\_validation\_results.txt:

5-Fold Cross-Validation Results

=====

Fold 1: AUC=0.83, Acc=0.78

Fold 2: AUC=0.85, Acc=0.81  
Fold 3: AUC=0.82, Acc=0.79  
Fold 4: AUC=0.84, Acc=0.80  
Fold 5: AUC=0.86, Acc=0.82

Mean AUC: 0.84 ± 0.015  
Mean Accuracy: 0.80 ± 0.015

---

### 9.6.3 4.2 Enrichment Factor

Script: bin/4\_2\_enrichment\_factor.py

#### 9.6.3.1 Formula

$$EF_x\% = \frac{\text{Actives in top } x\%}{\text{Total actives} \times x\%}$$

#### 9.6.3.2 Output results/4\_2\_enrichment\_factor\_results.txt:

Enrichment Factor Analysis  
=====

EF @ 1%: 15.2  
EF @ 2%: 12.8  
EF @ 5%: 8.5  
EF @ 10%: 5.2

Interpretation:

- Testing top 1% yields 15.2× more actives than random

---

### 9.6.4 4.3 Tanimoto Similarity

Script: bin/4\_3\_tanimoto\_similarity.py

#### 9.6.4.1 Calculation

$$Tanimoto(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

#### 9.6.4.2 Output

- results/4\_3\_tanimoto\_similarity\_results.txt
  - Similarity distribution plots
- 

### 9.6.5 4.4 Learning Curve

Script: bin/4\_4\_learning\_curve.py

#### 9.6.5.1 Process Train models with increasing data sizes:

```
sizes = [500, 1000, 2000, 5000, 10000]
for size in sizes:
 train_subset = dataset[:size]
 model.fit(train_subset)
 auc = evaluate(test_set)
```

#### 9.6.5.2 Output

- results/4\_4\_learning\_curve\_results.txt
  - Learning curve plot (learning\_curve.png)
- 

### 9.7 Step 5-6: Prediction

#### 9.7.1 Step 5: Featurization for Prediction

Script: bin/5\_0\_featurize\_for\_prediction.py

Menu Option: [5]

Process: Same as Step 2, but for new molecules

Input: data/zinc\_library.smi (or other library)

Output: results/prediction\_featurized/

---

#### 9.7.2 Step 6: Run Prediction

Script: bin/5\_1\_run\_prediction.py

Menu Option: [6]

```
Load model
model = dc.models.MultitaskClassifier()
model.restore(checkpoint='results/trained_model/')

Predict
predictions = model.predict(new_dataset)

Calculate K-Prediction Score
k_scores = predictions[:, 1] * 100

Rank
ranked = sort_by_score(k_scores)
```

##### 9.7.2.1 Process



### 9.7.2.2 Output results/5\_1\_new\_molecule\_predictions.csv:

```
SMILES,Probability,K_Prediction_Score,Rank
CC(C)Cc1ccc(cc1)C(C)C(O)=O,0.9523,95.23,1
CN1C=NC2=C1C(=O)N(C)C(=O)N2C,0.8845,88.45,2
CCCCCCCCCCCCCCC,0.1523,15.23,150
```

---

## 9.8 Complete Pipeline Flow

```
Raw Data (SMILES)
 ↓ [1_preparation.py]
Train/Test Split (CSV)
 ↓ [2_featurization.py]
Fingerprints (DeepChem format)
 ↓ [3_training.py]
Trained Model (checkpoint)
 ↓ [4_*.py]
Validation Reports
 ↓ [5_0_featurize_for_prediction.py]
New Molecule Fingerprints
 ↓ [5_1_run_prediction.py]
Ranked Predictions (CSV)
```

---

## 9.9 Related Pages

- [User Manual](#) - How to run pipeline
  - [Configuration](#) - Customize parameters
  - [Output Analysis](#) - Interpret results
  - [Troubleshooting](#) - Fix issues
- 

← Back to Wiki Home

## 10 Parallel Processing

## 11 Parallel Processing Guide

Complete guide to configuring and optimizing parallel processing in K-talysticFlow.

---

### 11.1 What is Parallel Processing?

Parallel processing allows K-talysticFlow to use multiple CPU cores simultaneously, dramatically reducing computation time for intensive tasks like:

- **Featurization** (5-10x faster)
  - **Tanimoto Similarity** (3-5x faster)
  - **Learning Curves** (4-8x faster)
  - **Predictions** (5-10x faster)
- 

### 11.2 Performance Gains

#### 11.2.1 Real-World Benchmarks

| Dataset Size      | Sequential Time | Parallel Time (6 cores) | Speedup |
|-------------------|-----------------|-------------------------|---------|
| 1,000 molecules   | 2 min           | 1 min                   | 2x      |
| 10,000 molecules  | 20 min          | 4 min                   | 5x      |
| 50,000 molecules  | 90 min          | 12 min                  | 7.5x    |
| 100,000 molecules | 180 min         | 20 min                  | 9x      |

*Benchmarks: Intel i7-8700 (6 cores, 12 threads), 16GB RAM*

---

### 11.3 Configuration

#### 11.3.1 Method 1: settings.py (Permanent)

Edit `settings.py` - Section 12:

```
Section 12: PARALLEL PROCESSING CONFIGURATIONS
=====

Enable/disable parallelism globally
ENABLE_PARALLEL_PROCESSING = True

Number of CPU cores to use
Options:
None = auto-detect (cpu_count - 1) RECOMMENDED
-1 = use ALL cores
1 = disable parallelism
```

```
N = use exactly N cores (e.g., 4, 6, 8)
N_WORKERS = None

Batch size for memory-efficient processing
Larger = faster but more RAM
PARALLEL_BATCH_SIZE = 100000

Minimum dataset size to trigger parallelism
Below this threshold, runs sequentially
PARALLEL_MIN_THRESHOLD = 10000
```

---

### 11.3.2 Method 2: Runtime Configuration (Temporary)

Use the control panel for on-the-fly changes:

```
python main.py
Select [8] Advanced Options
Select [3] Configure Parallel Processing Workers
```

#### Example:

Current Configuration:

```
ENABLE_PARALLEL_PROCESSING = True
N_WORKERS = None (Auto: 7 cores detected)
PARALLEL_BATCH_SIZE = 100000
PARALLEL_MIN_THRESHOLD = 10000
```

Enter new N\_WORKERS value: 4

Runtime configuration updated: N\_WORKERS = 4

**Note:** Runtime changes are temporary (session only)

---

## 11.4 Optimal Configuration Guide

### 11.4.1 Auto Mode (Recommended)

```
N_WORKERS = None
```

**Pros:** - Automatically detects optimal cores - Leaves 1 core free for system - Safe for all hardware - Best for general use

**When to use:** Default for most users

---

### 11.4.2 Fixed Core Count

```
N_WORKERS = 4 # Use exactly 4 cores
```

**Pros:** - Predictable resource usage - Good for shared systems - Consistent performance

**When to use:** - Shared workstations - Need consistent resource allocation - Troubleshooting performance

---

### 11.4.3 Maximum Performance

```
N_WORKERS = -1 # Use ALL cores
```

**Pros:** - Maximum speed - Best for dedicated machines

**Cons:** - May slow down system responsiveness - Not recommended during multitasking

**When to use:** - Dedicated analysis machine - Batch processing overnight - Maximum speed priority

---

### 11.4.4 Sequential Processing

```
ENABLE_PARALLEL_PROCESSING = False
OR
N_WORKERS = 1
```

**Pros:** - Lowest memory usage - Easiest debugging - Compatible with all systems

**Cons:** - 5-10x slower

**When to use:** - Low-end hardware - Memory constraints - Debugging issues - Very small datasets (< 1,000 molecules)

---

## 11.5 Hardware-Specific Recommendations

### 11.5.1 Low-End Systems

*Dual-core CPU, 8GB RAM*

```
ENABLE_PARALLEL_PROCESSING = False
N_WORKERS = 1
PARALLEL_BATCH_SIZE = 50000
```

---

### 11.5.2 Mid-Range Systems

*Quad-core CPU, 16GB RAM*

```
ENABLE_PARALLEL_PROCESSING = True
N_WORKERS = None # Auto: will use 3 cores
PARALLEL_BATCH_SIZE = 100000
```

---

### 11.5.3 High-End Systems

*8+ core CPU, 32GB+ RAM*

```
ENABLE_PARALLEL_PROCESSING = True
N_WORKERS = None # Auto: will use cpu_count-1
PARALLEL_BATCH_SIZE = 200000
```

---

### 11.5.4 Server/HPC Systems

*16+ cores, 64GB+ RAM*

```
ENABLE_PARALLEL_PROCESSING = True
N_WORKERS = -1 # Use all cores
PARALLEL_BATCH_SIZE = 500000
```

---

## 11.6 Testing Your Configuration

### 11.6.1 Run the Test Suite

```
python main.py
[8] Advanced Options → [2] Test Parallel Processing
```

The suite runs 6 tests:

- 11.6.1.1 **Test 1: Basic Parallel Execution**    Verifies joblib Parallel functionality
- 11.6.1.2 **Test 2: Large Array Processing**    Tests 1M element computation (real-world scale)
- 11.6.1.3 **Test 3: Memory Efficiency**    Ensures batch processing works correctly
- 11.6.1.4 **Test 4: Error Handling**    Tests recovery from worker failures
- 11.6.1.5 **Test 5: Performance Benchmark**    Compares sequential vs parallel (10K operations)
- 11.6.1.6 **Test 6: Worker Scaling**    Tests performance with 1, 2, 4, and max workers

**Interpretation:** - All tests **PASSED** - Optimal configuration - **Some tests FAILED** - Reduce workers or batch size - **Test 5 speedup < 2x** - Consider disabling parallelism

---

## 11.7 Memory Considerations

### 11.7.1 PARALLEL\_BATCH\_SIZE

Controls how many molecules are processed in each batch.

**Trade-off:** - **Larger batches** = Faster but more RAM - **Smaller batches** = Slower but safer

**Guidelines:**

| Available RAM | Recommended Batch Size | Max Dataset     |
|---------------|------------------------|-----------------|
| 8 GB          | 50,000                 | ~100K molecules |
| 16 GB         | 100,000                | ~500K molecules |
| 32 GB         | 200,000                | ~1M molecules   |
| 64 GB+        | 500,000                | Unlimited       |

**Signs batch size is too large:** - Out of memory errors - System freezing - Swap usage spikes

**Solution:** Reduce by 50%

---

## 11.8 Which Scripts Use Parallelism?

| Script                          | Parallelism | Speedup | Bottleneck     |
|---------------------------------|-------------|---------|----------------|
| 1_preparation.py                | No          | N/A     | I/O bound      |
| 2_featurization.py              | Yes         | 5-10x   | CPU bound      |
| 3_training.py                   | Partial*    | Varies  | GPU/CPU bound  |
| 4_0_evaluation_main.py          | No          | N/A     | Fast already   |
| 4_1_cross_validation.py         | No          | N/A     | Model overhead |
| 4_2_enrichment_factor.py        | No          | N/A     | Fast already   |
| 4_3_tanimoto_similarity.py      | Yes         | 3-5x    | CPU bound      |
| 4_4_learning_curve.py           | Yes         | 4-8x    | CPU bound      |
| 5_0_featurize_for_prediction.py | Yes         | 5-10x   | CPU bound      |
| 5_1_run_prediction.py           | Partial*    | Varies  | Model overhead |

\*TensorFlow uses internal parallelism (separate from joblib)

---

## 11.9 Troubleshooting

### 11.9.1 Issue: No speedup observed

**Possible causes:** 1. Dataset too small (< 10,000 molecules) 2. PARALLEL\_MIN\_THRESHOLD not reached 3. I/O bottleneck (slow disk) 4. Only 1-2 cores available

**Solutions:** - Check dataset size - Lower PARALLEL\_MIN\_THRESHOLD - Use SSD storage - Verify CPU core count

---

### 11.9.2 Issue: System becomes unresponsive

**Cause:** Too many workers consuming all CPU

**Solution:**

```
N_WORKERS = None # Auto mode (leaves 1 core free)
OR
N_WORKERS = cpu_count // 2 # Use half of cores
```

---

### 11.9.3 Issue: Out of memory errors

**Cause:** Batch size too large

**Solution:**

```
PARALLEL_BATCH_SIZE = 50000 # Reduce by 50%
OR
N_WORKERS = 2 # Reduce workers
```

---

### 11.9.4 Issue: “joblib” import error

**Solution:**

```
pip install joblib
```

---

### 11.9.5 Issue: Slower than expected

**Check:** 1. Disk speed (use SSD if possible) 2. RAM usage (swap = slow) 3. Background processes  
4. CPU temperature (throttling?)

**Benchmark:**

```
python bin/test_parallel_compatibility.py
```

---

## 11.10 Performance Monitoring

### 11.10.1 View Real-Time Status

The control panel shows current configuration:

Parallel Processing: ENABLED (6 workers)

### 11.10.2 Check Logs

```
View featurization log
cat results/02_featurization_log.txt

Look for:
"Using parallel processing with X workers"
"Processed Y molecules in Z seconds"
```

---

## 11.11 Advanced: Custom Parallelization

For developers extending K-talysticFlow:

```
from joblib import Parallel, delayed
import settings as cfg

def process_molecule(smiles):
 # Your processing logic
 return result

def parallel_process(smiles_list):
 if cfg.ENABLE_PARALLEL_PROCESSING and len(smiles_list) >= cfg.PARALLEL_MIN_THRESHOLD:
 # Parallel mode
 results = Parallel(n_jobs=cfg.N_WORKERS)(
 delayed(process_molecule)(smiles)
 for smiles in smiles_list
)
 else:
 # Sequential mode
 results = [process_molecule(smiles) for smiles in smiles_list]

 return results
```

---

## 11.12 Best Practices

### 11.12.1 Do's

1. Use **auto mode** (`N_WORKERS = None`) for most cases
2. **Test your configuration** before big runs
3. **Monitor memory usage** during first runs
4. **Adjust batch size** based on RAM
5. **Run overnight** for maximum performance (use -1 workers)
6. **Benchmark** with `test_parallel_compatibility.py`

### 11.12.2 Don'ts

1. **Don't use all cores** during multitasking



2. **Don't set batch size too high** (RAM limits)
  3. **Don't parallelize small datasets** (< 1,000 molecules)
  4. **Don't forget to save** settings after optimization
  5. **Don't ignore memory warnings**
- 

### 11.13 Related Pages

- [Configuration Guide](#) - All settings explained
  - [Installation](#) - Install joblib
  - [Troubleshooting](#) - Performance issues
  - [User Manual](#) - General usage
- 

← [Back to Wiki Home](#)

## 12 Output Analysis

### 13 Output Analysis: K-Prediction Score

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

---

#### 13.1 K-Prediction Score: Mathematical Foundation

##### 13.1.1 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.

##### 13.1.1.1 Fundamental Equation

K-Prediction Score =  $\text{Softmax}(\mathbf{f\_MLP}(\mathbf{x}))$

**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)

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

 Where z_final is calculated as:
 z_final = h · W_final + b_final
 h = ReLU(h · W + b)
 h = ReLU(x · W + b)

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

 # Layer 1: Input -> Hidden Layer 1
 z = W @ morgan_fingerprint + b
 h = ReLU(z) # Output with 1000 dimensions

 # Layer 2: Hidden Layer 1 -> Hidden Layer 2
 z = W @ h + b
 h = ReLU(z) # Output with 500 dimensions
```

```

Output Layer: Generating Logits
z_final = W_final @ h + 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

```

### 13.1.1.2 Detailed Mathematical Implementation

#### 13.1.2 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.

```

def softmax_properties_analysis():
 """
 Softmax Function: $\text{Softmax}(z) = \exp(z) / \sum \exp(z)$

 Important properties:
 - $\sum \text{Softmax}(z) = 1.0$ (valid probability distribution)
 - Softmax is monotonic: if $z > z$, then $\text{Softmax}(z) > \text{Softmax}(z)$
 - 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

```

#### 13.1.2.1 Mathematical Characteristics

#### 13.1.2.2 Sensitivity Analysis

- The **logits** ( $z_{\text{final}}$ ) 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**

---

## 13.2 Score Interpretation and Usage

### 13.2.1 1. Probabilistic Interpretation

```
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
```

#### 13.2.1.1 Calibration and Practical Meaning

#### 13.2.1.2 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"

### 13.2.2 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.

```
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: $\text{argmax}_t [\text{Sensitivity}(t) + \text{Specificity}(t) - 1]$ (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
```

### 13.2.2.1 Mathematical Implementation of Optimal Threshold

---

Next: [FAQ](#) | [Troubleshooting](#)

## 14 Configuration

## 15 Configuration Guide

Complete guide to customizing K-talysticFlow settings.

---

### 15.1 Configuration File: `settings.py`

All K-talysticFlow configurations are centralized in `settings.py` at the project root.

```
settings.py
import os
from pathlib import Path

... configuration sections ...
```

---

### 15.2 Configuration Sections

#### 15.2.1 Section 1: Main Paths

```
PROJECT_ROOT = Path(__file__).parent.resolve()
DATA_RAW_DIR = PROJECT_ROOT / 'data'
RESULTS_DIR = PROJECT_ROOT / 'results'
ACTIVE_SMILES_FILE = DATA_RAW_DIR / 'actives.smi'
INACTIVE_SMILES_FILE = DATA_RAW_DIR / 'inactives.smi'
```

**When to modify:** - Using different folder structure - Files named differently

**Example:**

```
ACTIVE_SMILES_FILE = DATA_RAW_DIR / 'my_actives.smiles'
INACTIVE_SMILES_FILE = DATA_RAW_DIR / 'my_inactives.smiles'
```

---

#### 15.2.2 Section 2: Basic Configurations

```
TEST_SET_FRACTION = 0.3
RANDOM_STATE = 42
FP_SIZE = 2048
FP_RADIUS = 3
```

##### 15.2.2.1 `TEST_SET_FRACTION` Default: 0.3 (30% test, 70% train)

**Options:** - 0.2 → 20% test, 80% train (more training data) - 0.3 → 30% test, 70% train  
**RECOMMENDED** - 0.4 → 40% test, 60% train (more test data)

**When to change:** - Small dataset → Use 0.2 (more training data) - Large dataset → Use 0.3-0.4 (better validation)

---

#### 15.2.2.2 RANDOM\_STATE Default: 42

**Purpose:** Reproducibility (same splits every time)

**Options:** - Any integer (e.g., 0, 123, 999) - None → Different splits each run

**When to change:** - Want different train/test splits - Testing model robustness

---

#### 15.2.2.3 FP\_SIZE (Fingerprint Size) Default: 2048 bits

**Options:** - 512 → Smaller, faster, less info - 1024 → Balanced - 2048 → Standard **RECOMMENDED** - 4096 → Larger, more info, slower

**Impact:** - Larger → More information but slower and more memory - Smaller → Faster but may lose information

**When to change:** - Memory constraints → Use 512 or 1024 - Large dataset → Try 4096 for better performance

---

#### 15.2.2.4 FP\_RADIUS Default: 3

**Options:** - 2 → ECFP4 (smaller substructures) - 3 → ECFP6 **RECOMMENDED** - 4 → ECFP8 (larger substructures)

**Impact:** - Larger radius → Captures larger molecular patterns - Smaller radius → More focused on local features

**Recommendation:** Start with 3, adjust based on molecule size

---

### 15.2.3 Section 5: Model Parameters

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

**15.2.3.1 layer\_sizes (Neural Network Architecture) Default:** [1000, 500] (2 hidden layers)

**Options:**

```
Smaller/faster model
'layer_sizes': [512, 256]

Larger/more complex model
'layer_sizes': [2048, 1024, 512]

Very deep model
'layer_sizes': [1024, 512, 256, 128]
```

**Guidelines:** - Small dataset (< 1K): [512, 256] - Medium dataset (1K-10K): [1000, 500] - Large dataset (> 10K): [2048, 1024, 512]

---

**15.2.3.2 dropouts (Regularization) Default:** 0.25 (25% dropout)

**Options:** - 0.1 → Light regularization - 0.25 → Moderate **RECOMMENDED** - 0.5 → Strong regularization (prevents overfitting)

**When to change:** - **Overfitting** (train AUC » test AUC) → Increase to 0.5 - **Underfitting** (both low) → Decrease to 0.1

---

**15.2.3.3 learning\_rate Default:** 0.001

**Options:** - 0.0001 → Slow, stable learning - 0.001 → Standard **RECOMMENDED** - 0.01 → Fast, may be unstable

**When to change:** - **Loss not decreasing** → Reduce to 0.0001 - **Very slow training** → Increase to 0.01 (with caution)

---

**15.2.3.4 nb\_epoch (Training Epochs) Default:** 50

**Options:** - 30 → Faster training - 50 → Standard **RECOMMENDED** - 100 → More training (may overfit)

**When to change:** - **Quick testing** → 20-30 epochs - **Production model** → 50-100 epochs - **Overfitting** → Reduce to 30

---

## 15.2.4 Section 6: Training Configurations

```
NB_EPOCH_TRAIN = 50
NB_EPOCH_CV = 30
```



```
NB_EPOCH_LC = 20
CLASSIFICATION_THRESHOLD = 0.5
```

#### 15.2.4.1 NB\_EPOCH\_\* (Epochs for Different Stages)

- **NB\_EPOCH\_TRAIN**: Main training (50)
- **NB\_EPOCH\_CV**: Cross-validation (30) - faster
- **NB\_EPOCH\_LC**: Learning curve (20) - even faster

**Why different values?** - CV and LC run multiple times → Use fewer epochs to save time - Still get valid comparisons

---

#### 15.2.4.2 CLASSIFICATION\_THRESHOLD Default: 0.5 (50% probability cutoff)

**Options:** - 0.3 → More predictions as “active” (higher recall, lower precision) - 0.5 → Balanced  
**RECOMMENDED** - 0.7 → Fewer predictions as “active” (lower recall, higher precision)

**When to change:** - **Need high recall** (don’t miss actives) → 0.3-0.4 - **Need high precision** (few false positives) → 0.6-0.7

---

### 15.2.5 Section 7: Validation Configurations

```
N_FOLDS_CV = 5
EF_FRACTIONS_PERCENT = [1.0, 2.0, 5.0, 10.0]
ENRICHMENT_FACTORS = [0.01, 0.05, 0.1]
TANIMOTO_SAMPLE_SIZE = 1000
```

#### 15.2.5.1 N\_FOLDS\_CV (Cross-Validation Folds) Default: 5

**Options:** - 3 → Faster, less reliable - 5 → Balanced **RECOMMENDED** - 10 → More reliable, slower

**When to change:** - **Quick testing** → 3 folds - **Publication** → 10 folds for robustness

---

#### 15.2.5.2 EF\_FRACTIONS\_PERCENT (Enrichment Factor Cutoffs) Default: [1.0, 2.0, 5.0, 10.0] (top 1%, 2%, 5%, 10%)

**When to change:** - **Large library** → Add 0.5% or 0.1% - **Small library** → Use only [5.0, 10.0]

---

#### 15.2.5.3 TANIMOTO\_SAMPLE\_SIZE Default: 1000 (sample 1000 molecules)

**Options:** - 500 → Faster - 1000 → Balanced - 5000 → More accurate, slower

**When to change:** - **Large dataset (> 50K)** → Use 5000 for better statistics - **Quick analysis** → Use 500

---

## 15.2.6 Section 8: Data Validation Configurations

```
MIN_MOLECULES_PER_CLASS = 50
MAX_MOLECULES_TOTAL = 100000
MIN_SMILES_LENGTH = 5
MAX_SMILES_LENGTH = 200
```

**15.2.6.1 MIN\_MOLECULES\_PER\_CLASS Default:** 50 (minimum 50 actives and 50 inactives)

**Options:** - 20 → Lower threshold (less reliable) - 50 → Recommended minimum - 100 → Better for robust models

---

**15.2.6.2 MAX\_MOLECULES\_TOTAL Default:** 100000 (100K molecules max)

**Purpose:** Memory protection

**When to change:** - **More RAM (32GB+)** → Increase to 500000 - **Less RAM (8GB)** → Decrease to 50000

---

**15.2.6.3 MIN/MAX\_SMILES\_LENGTH Default:** 5 to 200 characters

**Purpose:** Filter out very small or very large molecules

**When to change:** - **Peptides/polymers** → Increase MAX to 500 - **Fragments only** → Decrease MIN to 3

---

## 15.2.7 Section 12: Parallel Processing Configurations

```
ENABLE_PARALLEL_PROCESSING = True
N_WORKERS = None
PARALLEL_BATCH_SIZE = 100000
PARALLEL_MIN_THRESHOLD = 10000
```

See [Parallel Processing Guide](#) for complete documentation.

---

## 15.3 Configuration Recipes

### 15.3.1 Recipe 1: Fast Testing

```
Quick runs for testing
TEST_SET_FRACTION = 0.2
```

```
FP_SIZE = 1024
FP_RADIUS = 2
MODEL_PARAMS = {
 'layer_sizes': [512, 256],
 'nb_epoch': 20
}
N_FOLDS_CV = 3
```

---

### 15.3.2 Recipe 2: Production Model

```
High-quality model for publication
TEST_SET_FRACTION = 0.3
FP_SIZE = 2048
FP_RADIUS = 3
MODEL_PARAMS = {
 'layer_sizes': [1000, 500],
 'nb_epoch': 100,
 'dropouts': 0.3
}
N_FOLDS_CV = 10
TANIMOTO_SAMPLE_SIZE = 5000
```

---

### 15.3.3 Recipe 3: Large Dataset (> 50K)

```
Optimized for large datasets
FP_SIZE = 2048
MODEL_PARAMS = {
 'layer_sizes': [2048, 1024, 512],
 'nb_epoch': 50
}
ENABLE_PARALLEL_PROCESSING = True
N_WORKERS = -1
PARALLEL_BATCH_SIZE = 200000
```

---

### 15.3.4 Recipe 4: Low Memory (8GB RAM)

```
Minimize memory usage
FP_SIZE = 1024
FP_RADIUS = 2
MODEL_PARAMS = {
 'layer_sizes': [512, 256],
}
```

```
ENABLE_PARALLEL_PROCESSING = False
PARALLEL_BATCH_SIZE = 50000
MAX_MOLECULES_TOTAL = 50000
```

---

## 15.4 Advanced Customization

### 15.4.1 Modifying Scripts

For advanced users who want to modify pipeline behavior:

**Edit individual scripts in bin/ folder:**

```
bin/2_featurization.py
Find and modify featurization parameters

bin/3_training.py
Customize training loop, callbacks, etc.
```

**Recommendation:** Create backup before modifying:

```
cp bin/3_training.py bin/3_training.py.backup
```

---

### 15.4.2 Custom Loss Functions

Edit bin/3\_training.py to use custom loss:

```
Find MultitaskClassifier initialization
model = dc.models.MultitaskClassifier(
 # ... existing params ...
 # Add custom loss (advanced)
)
```

---

### 15.4.3 Custom Metrics

Add to evaluation scripts:

```
bin/4_0_evaluation_main.py
from sklearn.metrics import matthews_corrcoef

Add after existing metrics
mcc = matthews_corrcoef(y_true, y_pred)
print(f"Matthews Correlation Coefficient: {mcc:.4f}")
```

---

## 15.5 Configuration Best Practices

### 15.5.1 Do's

1. **Start with defaults** before customizing
2. **Document changes** in comments
3. **Backup settings.py** before major changes
4. **Test on small dataset** first
5. **Keep RANDOM\_STATE consistent** for reproducibility
6. **Match FP parameters** between training and prediction

### 15.5.2 Don'ts

1. **Don't change settings mid-pipeline** (except N\_WORKERS)
  2. **Don't use very small epochs** (< 10) for final models
  3. **Don't set N\_WORKERS too high** (leaves no CPU for system)
  4. **Don't ignore memory errors** (reduce batch size instead)
  5. **Don't modify settings during training** (restart pipeline)
- 

## 15.6 Applying Configuration Changes

### 15.6.1 Changes requiring re-run:

---

| Changed Setting    | Re-run From Step     |
|--------------------|----------------------|
| FP_SIZE, FP_RADIUS | [2] Featurization    |
| TEST_SET_FRACTION  | [1] Preparation      |
| MODEL_PARAMS       | [3] Training         |
| N_FOLDS_CV         | [4] Cross-validation |
| N_WORKERS          | No re-run needed     |

---

## 15.7 Monitoring Configuration Impact

### 15.7.1 Compare Model Versions

```
Save results with descriptive names
mv results results_config_v1
Modify settings
Re-run pipeline
mv results results_config_v2
Compare metrics
diff results_config_v1/4_0_evaluation_report.txt \
 results_config_v2/4_0_evaluation_report.txt
```

---

## 15.8 Related Pages

- [Parallel Processing Guide](#) - N\_WORKERS configuration
- [User Manual](#) - Using configured settings
- [Output Analysis](#) - Evaluating configuration impact
- [Troubleshooting](#) - Configuration-related issues

---

← [Back to Wiki Home](#)

## 16 FAQ

### 17 Frequently Asked Questions (FAQ)

Common questions and answers about K-talysticFlow.

---

#### 17.1 General Questions

##### 17.1.1 What is K-talysticFlow?

K-talysticFlow (KAST) is an automated deep learning pipeline for predicting molecular bioactivity. It helps researchers identify promising drug candidates through virtual screening using neural networks trained on molecular fingerprints.

---

##### 17.1.2 Who should use K-talysticFlow?

**Ideal for:** - Computational chemists - Drug discovery researchers - Graduate students in cheminformatics - Medicinal chemists performing virtual screening - Data scientists working with molecular data

**Not suitable for:** - Complete beginners without chemistry background - Projects requiring quantum mechanics (use DFT software instead) - Protein-ligand docking (use AutoDock, Vina, etc.)

---

##### 17.1.3 Is K-talysticFlow free?

Yes! K-talysticFlow is **open-source** and licensed under the **MIT License**. You can use it freely for: - Academic research - Commercial projects - Educational purposes

---

##### 17.1.4 What makes K-talysticFlow different?

1. **Fully automated** - No coding required for standard workflows
  2. **Interactive menu** - User-friendly control panel
  3. **Comprehensive validation** - 5 evaluation modules built-in
  4. **K-Prediction Score** - Proprietary ranking system
  5. **Parallel processing** - 5-10x faster than sequential
  6. **Production-ready** - Logging, error handling, reproducibility
- 

#### 17.2 Data & Inputs

##### 17.2.1 What input format does K-talysticFlow accept?

SMILES format (.smi or .smiles files)

Format:

SMILES [space] optional\_name

**Example:**

```
CC(C)Cc1ccc(cc1)C(C)C(=O)O ibuprofen
CN1C=NC2=C1C(=O)N(C)C(=O)N2C caffeine
```

---

### 17.2.2 How much data do I need?

**Minimum:** - 50 active compounds - 50 inactive compounds - Total: 100+ molecules

**Recommended:** - 500+ actives - 500+ inactives - Total: 1,000-10,000 molecules

**Optimal:** - 5,000+ actives - 5,000+ inactives - Total: 10,000+ molecules

**Rule of thumb:** More data = better model (up to ~100K molecules)

---

### 17.2.3 What is a good active/inactive ratio?

**Best:** 1:1 (balanced)

3,000 actives : 3,000 inactives

**Acceptable:** 1:2 to 1:10

1,000 actives : 5,000 inactives

**Poor:** > 1:10

100 actives : 5,000 inactives

**Note:** K-talysticFlow automatically balances datasets in `1_preparation.py`

---

### 17.2.4 Can I use my own molecular descriptors?

Currently, K-talysticFlow uses **Morgan Fingerprints (ECFP)** exclusively.

**Customization available:** - Radius (default: 3) - Size (default: 2048 bits)

Edit in `settings.py`:

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

**Future versions** may support custom descriptors.

---



### 17.2.5 What if I only have active compounds?

You need **both actives and inactives** to train a binary classifier.

**Options:** 1. **Generate decoys** using tools like: - [DUD-E](#) (Drug-like decoys) - [NRLiSt BDB](#) - Random compounds from ZINC 2. **Use negative examples** from literature 3. **Experimental inactives** from screening data

---

### 17.2.6 Can I predict multiple targets at once?

Currently, K-talysticFlow supports **single-target** prediction (binary classification: active/inactive for one target).

**Workaround:** Train separate models for each target.

---

## 17.3 Model & Training

### 17.3.1 How long does training take?

Typical times:

| Dataset Size      | Training Time |
|-------------------|---------------|
| 1,000 molecules   | 2-5 minutes   |
| 10,000 molecules  | 5-10 minutes  |
| 50,000 molecules  | 10-20 minutes |
| 100,000 molecules | 20-40 minutes |

*Times for default 50 epochs on mid-range CPU*

---

### 17.3.2 Can I use my own neural network architecture?

Yes! Edit settings.py:

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

**Example - Deeper network:**

```
'layer_sizes': [2048, 1024, 512, 256]
```

**Example - Smaller network:**

```
'layer_sizes': [512, 256]
```

---

### 17.3.3 What is a good ROC-AUC score?

| AUC Score | Performance    |
|-----------|----------------|
| 0.9 - 1.0 | Excellent      |
| 0.8 - 0.9 | Very Good      |
| 0.7 - 0.8 | Good           |
| 0.6 - 0.7 | Fair           |
| < 0.6     | Poor (retrain) |

**For publication:** AUC > 0.75 is generally acceptable.

---

### 17.3.4 My model overfits. What should I do?

**Overfitting signs:** - Training AUC » Test AUC (gap > 0.15) - High training accuracy, low test accuracy

**Solutions:**

1. **Increase dropout:**

```
'dropouts': 0.5 # Increase from 0.25
```

2. **Reduce epochs:**

```
'nb_epoch': 30 # Decrease from 50
```

3. **Get more training data**

4. **Simplify architecture:**

```
'layer_sizes': [512, 256] # Simpler than [1000, 500]
```

5. **Use data augmentation** (future feature)

---

### 17.3.5 Can I use a GPU?

Yes! TensorFlow automatically uses GPU if available.

**To enable:**

```
pip install tensorflow[and-cuda]
```

**Verify:**

```
import tensorflow as tf
print(tf.config.list_physical_devices('GPU'))
```

**Note:** GPU helps most for very large datasets (>50K molecules)

---

## 17.4 Performance & Speed

### 17.4.1 How can I make it faster?

1. **Enable parallel processing:**

```
ENABLE_PARALLEL_PROCESSING = True
N_WORKERS = None # Auto mode
```

2. **Increase batch size** (if you have RAM):

```
PARALLEL_BATCH_SIZE = 200000
```

3. **Use SSD storage** instead of HDD
  4. **Close background programs**
  5. **Use GPU** for training (TensorFlow)
- 

### 17.4.2 Why is parallel processing not helping?

Possible reasons:

1. **Dataset too small** (< 10,000 molecules)
    - Below `PARALLEL_MIN_THRESHOLD`
    - Solution: Lower threshold or disable parallelism
  2. **Only 1-2 CPU cores**
    - Not enough workers for speedup
    - Solution: Upgrade hardware or use cloud computing
  3. **I/O bottleneck** (slow disk)
    - Reading/writing is the slowest part
    - Solution: Use SSD
  4. **Script doesn't support parallelism**
    - Some scripts are sequential by design
    - See [Parallel Processing Guide](#)
- 

### 17.4.3 How much RAM do I need?

**Minimum:** 8 GB (small datasets < 10K)

**Recommended:** 16 GB (datasets 10K-50K)

**Optimal:** 32 GB+ (datasets > 50K)

RAM usage formula (rough):

$$\text{RAM (GB)} = (\text{Dataset size} \times \text{Fingerprint size} \times 8) / 1\text{e}9 + (\text{Batch size} \times 2) / 1\text{e}6$$

Example: 50K molecules, 2048-bit FP, 100K batch  
 $(50000 \times 2048 \times 8) / 1\text{e}9 + (100000 \times 2) / 1\text{e}6$   
0.82 GB + 0.2 GB = 1 GB

*Actual usage varies with parallel processing overhead*

---

## 17.5 Predictions & Screening

### 17.5.1 What is the K-Prediction Score?

**K-Prediction Score = Probability  $\times$  100**

It's a 0-100 scale for easy interpretation: - **90-100**: Very likely active (high priority) - **70-89**: Likely active (good candidates) - **50-69**: Possibly active (medium priority) - **Below 50**: Likely inactive

See [Output Analysis](#) for details.

---

### 17.5.2 How many compounds should I test experimentally?

Depends on your budget and model quality:

**High-quality model (AUC > 0.85, EF@1% > 10)**: - Test **top 1-2%** of predictions - K-Score > 80

**Good model (AUC 0.75-0.85, EF@1% = 5-10)**: - Test **top 5%** of predictions - K-Score > 60

**Fair model (AUC 0.65-0.75, EF@1% < 5)**: - Test **top 10%** or validate with secondary assay - K-Score > 50

---

### 17.5.3 Can I screen a million compounds?

**Yes!** K-talysticFlow can handle large libraries.

**Tips for huge libraries:**

1. **Enable parallel processing:**

```
ENABLE_PARALLEL_PROCESSING = True
N_WORKERS = -1 # Use all cores
```

2. **Increase batch size:**

```
PARALLEL_BATCH_SIZE = 500000
```

3. **Run overnight** or on HPC cluster

4. **Split library** into chunks if memory issues

**Time estimate:** - 1M molecules    1-3 hours (parallel, 8 cores)

---

#### 17.5.4 Are predictions reliable for molecules very different from training set?

**No!** Predictions are most reliable for molecules **similar to training data**.

**Check Tanimoto similarity:** - **Tanimoto** > **0.5** to training set → Reliable predictions    -  
**Tanimoto 0.3-0.5** → Moderate confidence    - **Tanimoto** < **0.3** → Unreliable (out of applicability domain)

**Recommendation:** Run `4_3_tanimoto_similarity.py` to assess chemical space coverage.

---

### 17.6 Technical Questions

#### 17.6.1 What Python version do I need?

**Required:** Python 3.9+

**Recommended:** Python 3.10 or 3.11

**Not supported:** Python 3.8 or earlier

---

#### 17.6.2 Can I run K-talysticFlow on Windows?

**Yes!** K-talysticFlow is fully compatible with: - Windows 10/11 - Linux (Ubuntu, CentOS, etc.)  
- macOS

---

#### 17.6.3 Do I need Conda or can I use pip?

**Recommended:** Conda (for RDKit)

**Alternative:** pip + system RDKit

**Why Conda?** - RDKit installation is much easier - Better dependency management - Isolated environments

---

#### 17.6.4 Can I use K-talysticFlow in Jupyter Notebooks?

Yes! You can import and use modules:

```
import sys
sys.path.append('/path/to/kast')

from bin import preparation, featurization
import settings as cfg

Run pipeline steps programmatically
```

**But:** The interactive menu (main.py) is designed for terminal use.

---

#### 17.6.5 Is there a Docker image?

Not yet, but coming soon!

**Current workaround:**

```
FROM continuumio/miniconda3
RUN conda install -c conda-forge rdkit
RUN pip install deepchem tensorflow scikit-learn
... etc
```

---

#### 17.6.6 Can I use K-talysticFlow in a web application?

Yes! You can integrate it as a backend:

```
from bin.run_prediction import predict_activity

API endpoint
@app.route('/predict', methods=['POST'])
def predict():
 smiles_list = request.json['smiles']
 predictions = predict_activity(smiles_list)
 return jsonify(predictions)
```

**Note:** Requires additional web framework (Flask, FastAPI, etc.)

---

### 17.7 Scientific Questions

#### 17.7.1 What type of molecular activity can I predict?

K-talysticFlow is designed for **binary classification**: - Active vs Inactive - Toxic vs Non-toxic - Binder vs Non-binder - Hit vs Non-hit

**Examples:** - Enzyme inhibitors - Receptor agonists/antagonists - Antimicrobial agents - Cytotoxicity - Blood-brain barrier permeability (BBB+/BBB-)

**Not suitable for:** - Regression (e.g., IC50, Ki values) - use different tools - Multi-class classification (A vs B vs C) - Time-series predictions

---

### 17.7.2 How does K-talysticFlow compare to other tools?

| Feature                    | KAST          | DeepChem  | AutoML | Commercial Tools |
|----------------------------|---------------|-----------|--------|------------------|
| <b>Ease of use</b>         |               |           |        |                  |
| <b>Automation</b>          | Full          | Partial   | Full   | Full             |
| <b>Validation suite</b>    | Comprehensive | Basic     | Good   | Excellent        |
| <b>Parallel processing</b> |               |           |        |                  |
| <b>Cost</b>                | Free          | Free      | Varies | \$\$\$\$         |
| <b>Customization</b>       | High          | Very High | Low    | Medium           |
| <b>Support</b>             | Community     | Community | Vendor | Vendor           |

**When to use KAST:** - Need full automation + flexibility - Academic research - Limited budget - Want comprehensive validation

**When to use alternatives:** - Need advanced features (e.g., 3D descriptors) - Large-scale enterprise deployment - Prefer GUI over CLI

---

### 17.7.3 Can I publish results from K-talysticFlow?

**Yes!** K-talysticFlow is suitable for academic publication.

**Please cite:**

```
@software{kast2025,
 author = {Santos, Késsia Souza},
 title = {K-talysticFlow: Automated Deep Learning Pipeline for Molecular Screening},
 year = {2025},
 version = {1.0.0},
 url = {https://github.com/kelsouzs/kast}
}
```

---

### 17.7.4 What are the limitations of K-talysticFlow?

**Current limitations:**

1. **Binary classification only** (not regression)
2. **2D fingerprints only** (no 3D descriptors yet)
3. **Single target** (not multi-task learning)
4. **No explicit chemistry rules** (not rule-based)

## 5. Requires both active and inactive data

**Future enhancements planned:** - Regression models - 3D descriptors - Multi-target prediction  
- Explainability (SHAP, attention)

---

## 17.8 More Help

- [User Manual](#) - Complete usage guide
- [Installation](#) - Setup instructions
- [Troubleshooting](#) - Fix common issues
- [Output Analysis](#) - Interpret results

**Still have questions?** - Email: [kelsouzs.uefs@gmail.com](mailto:kelsouzs.uefs@gmail.com) - GitHub Issues: [Report a problem](#)

---

← [Back to Wiki Home](#)



## 18 Troubleshooting

### 19 Troubleshooting Guide

Solutions to common issues and errors in K-talysticFlow.

---

#### 19.1 Table of Contents

1. Installation Issues
  2. Data Preparation Errors
  3. Featurization Problems
  4. Training Issues
  5. Memory Errors
  6. Parallel Processing Problems
  7. Prediction Errors
  8. Performance Issues
  9. General Errors
- 

#### 19.2 Installation Issues

##### 19.2.1 Issue: ModuleNotFoundError: No module named 'rdkit'

**Cause:** RDKit not installed

**Solution:**

```
Using Conda (RECOMMENDED)
conda install -c conda-forge rdkit

OR using pip (may fail on some systems)
pip install rdkit-pypi
```

---

##### 19.2.2 Issue: ImportError: DLL load failed (Windows)

**Cause:** Missing Visual C++ redistributables

**Solution:** 1. Download [Microsoft Visual C++ Redistributable](#) 2. Install and restart 3. Reinstall Python packages

---

##### 19.2.3 Issue: TensorFlow installation fails

**Solution:**

```
Install specific version
pip install tensorflow==2.15.0
```

```
If still fails (CPU only)
pip install tensorflow-cpu
```

---

**19.2.4 Issue:** ImportError: cannot import name 'DeepChem'

**Solution:**

```
pip uninstall deepchem
pip install deepchem==2.7.1
```

---

**19.2.5 Issue:** Permission denied when installing

**Solution (Linux/Mac):**

```
pip install -r requirements.txt --user
```

**Solution (Windows):** Run command prompt as Administrator

---

## 19.3 Data Preparation Errors

**19.3.1 Issue:** FileNotFoundError: 'actives.smi' not found

**Cause:** SMILES files not in data/ folder

**Solution:** 1. Create data/ folder in project root 2. Place actives.smi and inactives.smi there  
3. Check file names (case-sensitive on Linux)

```
Check structure
ls data/
Should show:
actives.smi
inactives.smi
```

---

**19.3.2 Issue:** ValueError: Invalid SMILES: XYZ

**Cause:** Malformed SMILES string

**Solution:** 1. Validate SMILES using RDKit:

```
from rdkit import Chem
mol = Chem.MolFromSmiles('YOUR_SMILES')
if mol is None:
 print("Invalid SMILES")
```

2. Remove invalid entries from .smi files
3. Use canonicalized SMILES

---

### 19.3.3 Issue: Error: Insufficient data (< 50 molecules per class)

**Cause:** Too few compounds

**Solution:** - **Minimum:** 50 actives + 50 inactives - Add more compounds to dataset - Or adjust threshold in `settings.py`:

```
MIN_MOLECULES_PER_CLASS = 20 # Lower threshold
```

---

### 19.3.4 Issue: Train/test split fails

**Cause:** Scaffold splitting issues

**Solution:** Try random splitting instead:

Edit `bin/1_preparation.py`:

```
Find this line:
splitter = dc.splits.ScaffoldSplitter()

Change to:
splitter = dc.splits.RandomSplitter()
```

---

## 19.4 Featurization Problems

### 19.4.1 Issue: MemoryError during featurization

**Cause:** Dataset too large for available RAM

**Solution:**

1. Reduce batch size:

```
PARALLEL_BATCH_SIZE = 50000 # Reduce from 100000
```

2. Disable parallelism:

```
ENABLE_PARALLEL_PROCESSING = False
```

3. Process in chunks manually
- 

### 19.4.2 Issue: Featurization extremely slow

**Cause:** Large dataset with sequential processing

**Solution:**

1. Enable parallel processing:

```
ENABLE_PARALLEL_PROCESSING = True
N_WORKERS = None # Auto mode
```

## 2. Increase workers:

```
N_WORKERS = 6 # Or your CPU core count - 1
```

---

### 19.4.3 Issue: ValueError: Fingerprint size must be > 0

**Cause:** Invalid fingerprint configuration

**Solution:** Check `settings.py`:

```
FP_SIZE = 2048 # Must be > 0
FP_RADIUS = 3 # Must be > 0
```

---

### 19.4.4 Issue: RDKit WARNING: not removing hydrogen atom

**Cause:** RDKit warnings (usually harmless)

**Solution:** These are warnings, not errors. To suppress:

```
from rdkit import RDLogger
RDLogger.DisableLog('rdApp.*')
```

Already handled in K-talysticFlow code.

---

## 19.5 Training Issues

### 19.5.1 Issue: Training stuck at 0% for long time

**Cause:** Very large dataset or slow initialization

**Solution:** - **Normal behavior** for first epoch (TensorFlow initialization) - Wait 2-5 minutes before concluding it's stuck - Check CPU/GPU usage in Task Manager

---

### 19.5.2 Issue: ValueError: No training data found

**Cause:** Featurization step not completed

**Solution:** 1. Run featurization first:

```
python main.py
Select [2] Featurize Molecules
```

2. Check `results/featurized_datasets/train/` exists

---

### 19.5.3 Issue: Training loss not decreasing

Causes & Solutions:

#### 1. Learning rate too high:

```
'learning_rate': 0.0001 # Reduce from 0.001
```

2. Random labels (data quality issue): - Verify label correctness - Check if actives and inactives are truly different

#### 3. Model too simple:

```
'layer_sizes': [2048, 1024, 512] # Increase complexity
```

---

### 19.5.4 Issue: CUDA out of memory (GPU)

Solution:

1. Reduce batch size (TensorFlow internal):
  - Edit DeepChem model parameters (advanced)
2. Switch to CPU:

```
export CUDA_VISIBLE_DEVICES="" # Linux/Mac
set CUDA_VISIBLE_DEVICES= # Windows
```

#### 3. Use smaller model:

```
'layer_sizes': [512, 256]
```

---

### 19.5.5 Issue: Training finishes but no model file

Cause: Error during checkpoint saving

Solution: 1. Check results/trained\_model/ folder exists 2. Check write permissions 3. Review results/03\_training\_log.txt for errors

---

## 19.6 Memory Errors

### 19.6.1 Issue: MemoryError: Unable to allocate array

Cause: Insufficient RAM

Solutions:

#### 1. Reduce batch size:

```
PARALLEL_BATCH_SIZE = 25000 # Reduce significantly
```

#### 2. Disable parallelism:

```
ENABLE_PARALLEL_PROCESSING = False
```

### 3. Reduce workers:

```
N_WORKERS = 2 # Use fewer cores
```

### 4. Close background programs

### 5. Use swap/pagefile (slower but works)

### 6. Upgrade RAM (8GB → 16GB)

---

## 19.6.2 Issue: Python process killed suddenly

**Cause:** Out-of-memory (OOM) killer (Linux)

**Solution:**

```
Check logs
dmesg | grep -i "killed process"

If OOM killer was triggered:
1. Reduce memory usage (see above)
2. Add swap space
sudo fallocate -l 8G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
```

---

## 19.7 Parallel Processing Problems

### 19.7.1 Issue: No speedup from parallelism

**Causes:**

#### 1. Dataset too small (< 10K):

```
PARALLEL_MIN_THRESHOLD = 1000 # Lower threshold
```

#### 2. Only 1-2 cores available:

```
Check cores
python -c "import os; print(os.cpu_count())"
```

#### 3. I/O bottleneck (slow disk): - Use SSD instead of HDD - Can't fix with parallelism

#### 4. Script doesn't support parallelism: - See [Parallel Processing Guide](#) for supported scripts

---

### 19.7.2 Issue: joblib errors

#### Error:

AttributeError: module 'joblib' has no attribute 'Parallel'

#### Solution:

```
pip uninstall joblib
pip install joblib==1.3.0
```

---

### 19.7.3 Issue: Parallel test suite fails

#### Solution:

```
Run test suite
python bin/test_parallel_compatibility.py

If Test 1 fails: joblib issue
pip install --upgrade joblib

If Test 2 fails: reduce workers
N_WORKERS = 2

If Test 3 fails: memory issue
PARALLEL_BATCH_SIZE = 25000
```

---

### 19.7.4 Issue: Slower with parallelism enabled

**Cause:** Overhead exceeds benefit (small dataset)

**Solution:** Disable for small datasets:

```
PARALLEL_MIN_THRESHOLD = 50000 # Only parallelize large datasets
```

---

## 19.8 Prediction Errors

### 19.8.1 Issue: FileNotFoundError: Model checkpoint not found

**Cause:** Model not trained yet

**Solution:** 1. Train model first:

```
python main.py
Select [3] Train Model
```

2. Verify results/trained\_model/checkpoint1.pt exists

---

### 19.8.2 Issue: Predictions all the same value

Causes:

1. **Poor model (AUC < 0.6):** - Retrain with better data - Check evaluation metrics
  2. **Invalid input data:** - Verify SMILES are correct - Check featurization completed
  3. **Wrong model loaded:** - Check `training_metadata.json` matches current settings
- 

### 19.8.3 Issue: ValueError: Feature mismatch

**Cause:** Prediction fingerprints don't match training

**Solution:** Ensure same parameters for prediction:

```
settings.py must match training config
FP_SIZE = 2048 # Same as training
FP_RADIUS = 3 # Same as training
```

**If changed after training:** Re-featurize and re-predict:

```
python main.py
[5] Featurize for Prediction
[6] Run Prediction
```

---

### 19.8.4 Issue: Predictions take too long

**Solution:**

1. **Enable parallelism:**

```
ENABLE_PARALLEL_PROCESSING = True
```

2. **Increase workers:**

```
N_WORKERS = -1
```

3. **Check dataset size:**

```
wc -l data/zinc_library.smi
If > 100K, expect 10-30 min even with parallelism
```

---

## 19.9 Performance Issues

### 19.9.1 Issue: Pipeline very slow overall

**Checklist:**

Parallel processing enabled?



```
ENABLE_PARALLEL_PROCESSING = True
```

**Using SSD or HDD?** - HDD: 5-10x slower I/O - Solution: Use SSD

**Sufficient RAM?** - Check usage in Task Manager - Close background programs

**CPU usage low?** - May indicate I/O bottleneck - Can't fix with CPU optimization

**Antivirus scanning files?** - Exclude K-talysticFlow folder

---

### 19.9.2 Issue: Specific script very slow

**Script-specific solutions:**

**1\_preparation.py (slow):** - Normal for large datasets - Scaffold splitting is intensive - Expected: 1-5 min for 10K molecules

**2\_featurization.py (slow):** - Enable parallelism (5-10x faster) - Use more workers

**3\_training.py (slow):** - Reduce epochs - Use GPU - Simplify model

**4\_4\_learning\_curve.py (slow):** - Most time-consuming evaluation - Enable parallelism - Reduce training sizes in script (advanced)

---

## 19.10 General Errors

**19.10.1 Issue: ImportError: cannot import name 'MultitaskClassifier'**

**Cause:** DeepChem version mismatch

**Solution:**

```
pip install deepchem==2.7.1
```

---

**19.10.2 Issue: Control panel menu not displaying correctly**

**Cause:** Terminal encoding issues

**Solution:**

**Windows:**

```
chcp 65001 # Set UTF-8 encoding
```

**Linux/Mac:**

```
export LANG=en_US.UTF-8
```

---

### 19.10.3 Issue: Logs not generated

**Cause:** Logging directory doesn't exist

**Solution:**

```
mkdir -p results/logs
```

Or let scripts create automatically (check write permissions).

---

### 19.10.4 Issue: PermissionError: [Errno 13]

**Cause:** Insufficient write permissions

**Solution:**

**Linux/Mac:**

```
chmod -R 755 results/
```

**Windows:** Right-click folder → Properties → Security → Edit → Grant Full Control

---

### 19.10.5 Issue: Conflicting package versions

**Error:**

ERROR: pip's dependency resolver does not currently take into account all the packages that are

**Solution:**

```
Create fresh environment
conda create -n kast_clean python=3.10 -y
conda activate kast_clean
conda install -c conda-forge rdkit -y
pip install -r requirements.txt
```

### 19.10.6 Issue: Script exits without error message

**Solution:** Check log files:

```
Main log
cat results/logs/kast_YYYYMMDD.log

Script-specific logs
cat results/02_featurization_log.txt
cat results/03_training_log.txt
etc.
```

---

### 19.10.7 Issue: Results folder messy/corrupted

Solution:

```
Backup current results
mv results results_backup_$(date +%Y%m%d)

Create fresh results folder
mkdir results
mkdir results/logs
mkdir results/featurized_datasets
mkdir results/trained_model

Re-run pipeline
python main.py
```

---

## 19.11 Getting More Help

### 19.11.1 Step 1: Check Logs

```
Recent errors
tail -n 50 results/logs/kast_*.log

Specific script log
cat results/03_training_log.txt
```

---

### 19.11.2 Step 2: Enable Debugging

Edit settings.py:

```
DEBUG = True
VERBOSE = True
```

---

### 19.11.3 Step 3: Run Dependency Checker

```
python main.py
[8] Advanced Options → [1] Check Dependencies
```

---

### 19.11.4 Step 4: Test Parallel Processing

```
python main.py
[8] Advanced Options → [2] Test Parallel Processing
```

---

### 19.11.5 Step 5: Minimal Reproducible Example

Create small test dataset:

```
Create minimal data
head -n 100 data/actives.smi > data/actives_test.smi
head -n 100 data/inactives.smi > data/inactives_test.smi

Update settings.py temporarily
ACTIVE_SMILES_FILE = 'data/actives_test.smi'
INACTIVE_SMILES_FILE = 'data/inactives_test.smi'

Run pipeline
python main.py
```

---

### 19.11.6 Step 6: Report Issue

If problem persists, open a GitHub issue with:

1. **Error message** (full traceback)
2. **Log files** (attach relevant logs)
3. **System info:**

```
python --version
pip list | grep -E 'deepchem|rdkit|tensorflow'
uname -a # Linux/Mac
systeminfo # Windows
```

4. **Steps to reproduce**

**GitHub Issues:** <https://github.com/kelsouzs/kast/issues>

---

## 19.12 Related Resources

- [FAQ](#) - Common questions
  - [Installation](#) - Setup guide
  - [User Manual](#) - Usage instructions
  - [Parallel Processing](#) - Performance optimization
- 

← Back to Wiki Home **Still stuck?** Contact: [kelsouzs.uefs@gmail.com](mailto:kelsouzs.uefs@gmail.com)

20 Index

21 K-talysticFlow Wiki - Complete Index

Version: 1.0.0 | Last Updated: October 10, 2025

21.1 All Wiki Pages

21.1.1 Main Pages

| # | Page                           | Description                         | Read Time |
|---|--------------------------------|-------------------------------------|-----------|
| 1 | <a href="#">Home</a>           | Wiki homepage and project overview  | 5 min     |
| 2 | <a href="#">README</a>         | Documentation index and quick links | 3 min     |
| 3 | <a href="#">Navigation Map</a> | Visual guide to find information    | 10 min    |

21.1.2 Getting Started (Essential)

| # | Page                         | Description                      | Read Time |
|---|------------------------------|----------------------------------|-----------|
| 4 | <a href="#">Installation</a> | Complete setup for all platforms | 20 min    |
| 5 | <a href="#">User Manual</a>  | Step-by-step usage guide         | 40 min    |

21.1.3 Core Documentation

| # | Page                            | Description                             | Read Time |
|---|---------------------------------|-----------------------------------------|-----------|
| 6 | <a href="#">Pipeline Steps</a>  | Technical details of each pipeline step | 60 min    |
| 7 | <a href="#">Output Analysis</a> | Results interpretation and metrics      | 45 min    |
| 8 | <a href="#">Configuration</a>   | Settings customization guide            | 50 min    |

21.1.4 Advanced Topics

| # | Page                                | Description                    | Read Time |
|---|-------------------------------------|--------------------------------|-----------|
| 9 | <a href="#">Parallel Processing</a> | Performance optimization guide | 30 min    |

#### 21.1.5 Help & Support

| #  | Page                            | Description                     | Read Time |
|----|---------------------------------|---------------------------------|-----------|
| 10 | <a href="#">FAQ</a>             | 40+ frequently asked questions  | 30 min    |
| 11 | <a href="#">Troubleshooting</a> | 50+ common issues and solutions | Variable  |

#### 21.1.6 Meta Documentation

| #  | Page                                | Description                   | Read Time |
|----|-------------------------------------|-------------------------------|-----------|
| 12 | <a href="#">Wiki Update Summary</a> | Update history and statistics | 10 min    |
| 13 | <a href="#">Index</a>               | This page - Complete index    | 5 min     |

### 21.2 Quick Access by Category

#### 21.2.1 Installation & Setup

- [Installation Guide](#)
- [System Requirements](#)
- [Conda Setup](#)
- [Verification](#)

#### 21.2.2 Basic Usage

- [Quick Start](#)
- [Data Preparation](#)
- [Running Pipeline](#)
- [Understanding Outputs](#)

#### 21.2.3 Results & Analysis

- [K-Prediction Score](#)
- [ROC-AUC](#)
- [Enrichment Factor](#)
- [All Metrics](#)

#### 21.2.4 Performance

- [Parallel Processing Setup](#)
- [Hardware Optimization](#)
- [Performance Benchmarks](#)
- [Testing Configuration](#)

#### 21.2.5 Customization

- [All Settings](#)
- [Model Parameters](#)
- [Fingerprint Settings](#)
- [Configuration Recipes](#)

#### 21.2.6 Troubleshooting

- [Installation Issues](#)
  - [Memory Errors](#)
  - [Performance Issues](#)
  - [General Errors](#)
- 

### 21.3 Reading Order Recommendations

#### 21.3.1 For First-Time Users

1. Home (5 min)
  2. Installation (20 min)
  3. User Manual - Quick Start (10 min)
  4. Output Analysis - K-Score (15 min)
- Total: ~50 minutes

#### 21.3.2 For Regular Users

1. User Manual - Complete (40 min)
  2. Pipeline Steps - Overview (30 min)
  3. Output Analysis - All Metrics (45 min)
  4. Configuration - Basics (30 min)
- Total: ~2.5 hours

#### 21.3.3 For Power Users

1. Pipeline Steps - Full (60 min)
  2. Parallel Processing (30 min)
  3. Configuration - Advanced (50 min)
  4. All other pages as reference
- Total: ~3+ hours
-

## 21.4 Search by Keyword

### 21.4.1 A-C

- **Accuracy** → [Output Analysis](#)
- **Active/Inactive** → [User Manual](#)
- **AUC** → [Output Analysis](#)
- **Batch Size** → [Parallel Processing](#)
- **Configuration** → [Configuration](#)
- **Cross-Validation** → [Output Analysis](#)

### 21.4.2 D-F

- **Data Preparation** → [Pipeline Steps](#)
- **DeepChem** → [Pipeline Steps](#)
- **Dropout** → [Configuration](#)
- **Enrichment Factor** → [Output Analysis](#)
- **Epochs** → [Configuration](#)
- **Errors** → [Troubleshooting](#)
- **F1-Score** → [Output Analysis](#)
- **FAQ** → [FAQ](#)
- **Featurization** → [Pipeline Steps](#)
- **Fingerprints** → [Pipeline Steps](#)

### 21.4.3 G-M

- **GPU** → [FAQ](#)
- **Installation** → [Installation](#)
- **K-Prediction Score** → [Output Analysis](#)
- **Learning Curve** → [Output Analysis](#)
- **Memory** → [Troubleshooting](#)
- **Metrics** → [Output Analysis](#)
- **Model Training** → [Pipeline Steps](#)
- **Morgan Fingerprints** → [Pipeline Steps](#)

### 21.4.4 N-R

- **N\_WORKERS** → [Parallel Processing](#)
- **Neural Network** → [Pipeline Steps](#)
- **Overfitting** → [FAQ](#)
- **Parallel Processing** → [Parallel Processing](#)
- **Performance** → [Parallel Processing](#)
- **Precision** → [Output Analysis](#)
- **Predictions** → [Pipeline Steps](#)
- **Quick Start** → [User Manual](#)
- **RAM** → [FAQ](#)
- **Recall** → [Output Analysis](#)
- **ROC** → [Output Analysis](#)



### 21.4.5 S-Z

- **Scaffold Splitting** → [Pipeline Steps](#)
  - **Settings** → [Configuration](#)
  - **SMILES** → [User Manual](#)
  - **Speedup** → [Parallel Processing](#)
  - **Tanimoto** → [Output Analysis](#)
  - **Training** → [Pipeline Steps](#)
  - **Troubleshooting** → [Troubleshooting](#)
  - **Validation** → [Output Analysis](#)
- 

## 21.5 Content Statistics

### 21.5.1 Total Documentation

- **Pages:** 13
- **Words:** ~35,000+
- **Code Examples:** 120+
- **Tables:** 60+
- **Diagrams:** 20+
- **Links (Internal):** 200+
- **Links (External):** 25+

### 21.5.2 Reading Time

- **Quick Start Path:** ~50 min
- **Regular User Path:** ~2.5 hours
- **Complete Read:** ~5-6 hours
- **Reference Use:** As needed

### 21.5.3 Coverage

- Installation (all platforms)
  - Usage (all features)
  - Configuration (all settings)
  - Troubleshooting (50+ issues)
  - FAQ (40+ questions)
  - Examples (100+ code snippets)
- 

## 21.6 Learning Resources

### 21.6.1 Tutorials

- [First Run Tutorial](#)
- [Step-by-Step Workflow](#)
- [Parallel Processing Setup](#)

### 21.6.2 Reference

- [Complete Pipeline Documentation](#)
- [All Settings Reference](#)
- [All Metrics Explained](#)

### 21.6.3 Examples

- [Configuration Recipes](#)
  - [Workflow Examples](#)
  - [Troubleshooting Examples](#)
- 

## 21.7 Getting Help

### 21.7.1 Self-Service

1. **Search this index** for keywords
2. **Check FAQ** for common questions
3. **Review Troubleshooting** for errors
4. **Read relevant section** in detail

### 21.7.2 Community Support

- **GitHub Issues:** [Report bugs](#)
  - **Email:** [kelsouzs.uefs@gmail.com](mailto:kelsouzs.uefs@gmail.com)
  - **LinkedIn:** [@kelsouzs](#)
- 

## 21.8 Updates

### 21.8.1 Version 1.0.0 (October 10, 2025)

- Complete Wiki created
- All features documented
- 13 comprehensive pages
- 35,000+ words
- Ready for production

### 21.8.2 Future Plans

- Video tutorials
  - Docker documentation
  - Case studies
  - API reference
-

## 21.9 Contributing

Want to improve the documentation?

1. **Report issues** - Typos, errors, unclear sections
2. **Suggest additions** - Missing topics, better examples
3. **Submit PRs** - Fix and improve

See [Wiki Update Summary](#) for guidelines.

---

## 21.10 License

This documentation is part of K-talysticFlow and is licensed under the MIT License.

---

## 21.11 Complete Documentation Suite

**13 Pages** | **35,000+ Words** | **120+ Examples**

### 21.11.1 Quick Links

[Home](#) | [Install](#) | [Manual](#) | [Analysis](#)

[Performance](#) | [Config](#) | [FAQ](#) | [Troubleshoot](#)

---

**Version:** 1.0.0 | **Updated:** October 10, 2025 *Made with for the computational chemistry community* **Developer:** Késsia Souza Santos (@kelsouzs)