Dependency Chain Optimization Using Machine Learning

Executive Summary

This project implements an advanced system for optimizing dependency chains in software projects using a combination of graph-based analysis, machine learning, and reinforcement learning. The system consists of two main components:

1. A critical node classifier that identifies important dependencies
2. A reinforcement learning agent that generates optimized dependency chains

System Architecture

Critical Node Classification

The system begins with a graph-based representation of dependencies using *networkx* where:

- Nodes represent artifacts, releases, and added values
- Edges represent relationships between nodes (dependencies, added values, artifact-release relationships)
- Node features include both topological metrics and semantic information

Graph statistics:
- Number of nodes: 442275
- Number of edges: 499760

Relationship Types Distribution

Type	Count	Percentage
dependency	332,259	66.5%
addedValues	121,617	24.3%
relationship_AR	46,124	9.2%
Total	**500,000**	**100%**

Scope Distribution

Scope	Count	Percentage
compile	203,469	61.2%

test	75,891	22.8%
runtime	26,815	8.1%
provided	25,985	7.8%
implementation	39	0.012%
api	19	0.006%
system	16	0.005%
optional	15	0.005%
import	4	0.001%
external	2	0.001%
runtme	2	0.001%
integration-test	1	< 0.001%
runtimeOnly	1	< 0.001%
Total	**332,259**	**100%**

Node Type Distribution

Type	Count	Percentage
POPULARITY_1_YEAR	40,152	33.0%
CVE	39,859	32.8%
FRESHNESS	39,855	32.7%
SPEED	1,751	1.5%
Total	**121,617**	**100%**

Feature Engineering

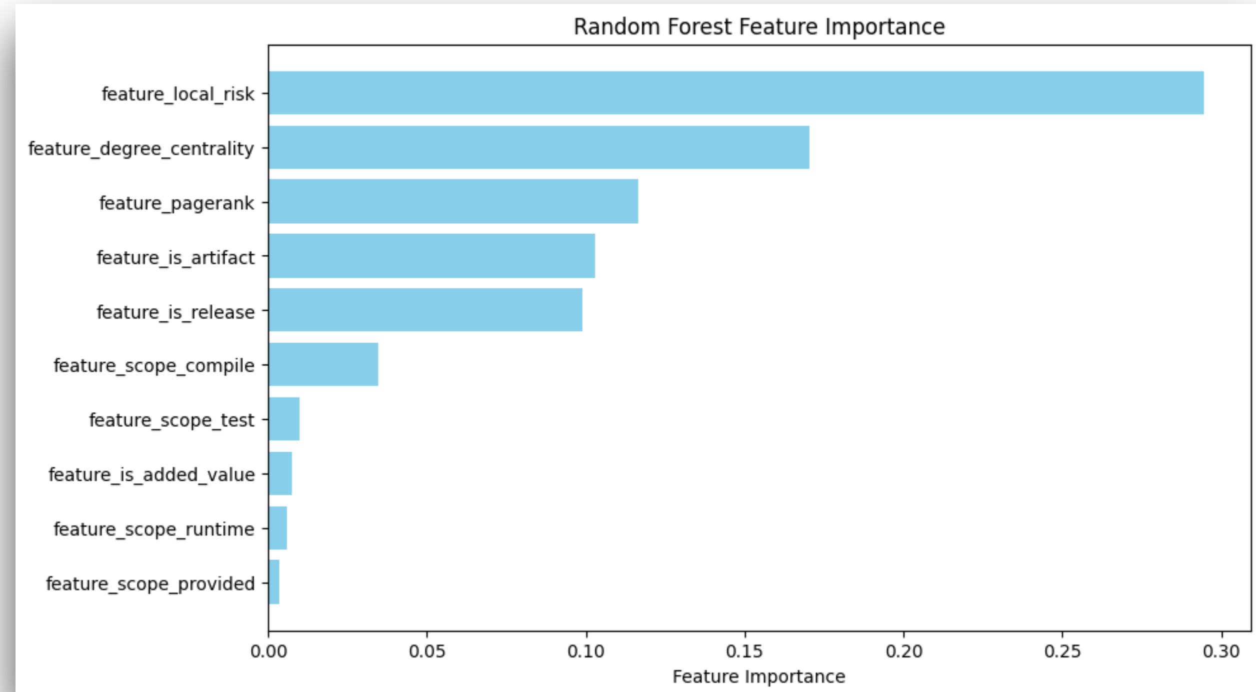
The system extracts several types of features:

1) Topological Features:
- Degree centrality
- Betweenness centrality (using k=1000)
- PageRank
- Clustering coefficient
- Local risk ratio

The local risk implemented here is a simply ratio of successors to predecessors

2) Semantic Features:
- Node types (Artifact, Release, AddedValue)
- Dependency scopes (compile, runtime, test, etc.)
- Quality metrics (CVE, Freshness, Speed, Popularity)

All these features are stored as node_features which are extremely useful in predicting critical nodes, more so than embeddings as seen in the diagram below



Node2Vec

We use the following parameters to initialize Node2Vec -> *(self, dimensions=128, walk_length=30, num_walks=200)* and the following to fit it with our training data -> *(window=10, min_count=1)*

The dimensions, walk length and number of walks can all be increaased for better performance.

Pipeline

The training set (353820 nodes) chain goes as follows:
Extract topological and semantic features -> Fit Transform using Node2Vec -> Identify critical Nodes

The testing set (88455 nodes) chain goes as follows:

Extract topological and semantic features -> Transform using Node2Vec -> Identify critical Nodes

Identification of Critical Nodes

```
score = (
    features['degree_centrality'] * 0.2 +
    features['betweenness_centrality'] * 0.3 +
    features['pagerank'] * 0.2 +
    features['local_risk'] * 0.1 +
    (features['is_artifact'] * 0.1) +
    (features['scope_compile'] > 0) * 0.05 +
    (features['type_CVE'] > 0) * 0.05
)
```

The above score function is used to assign as value to each node. After each nodes value has been determined, the top 5% of nodes are classified as critical.

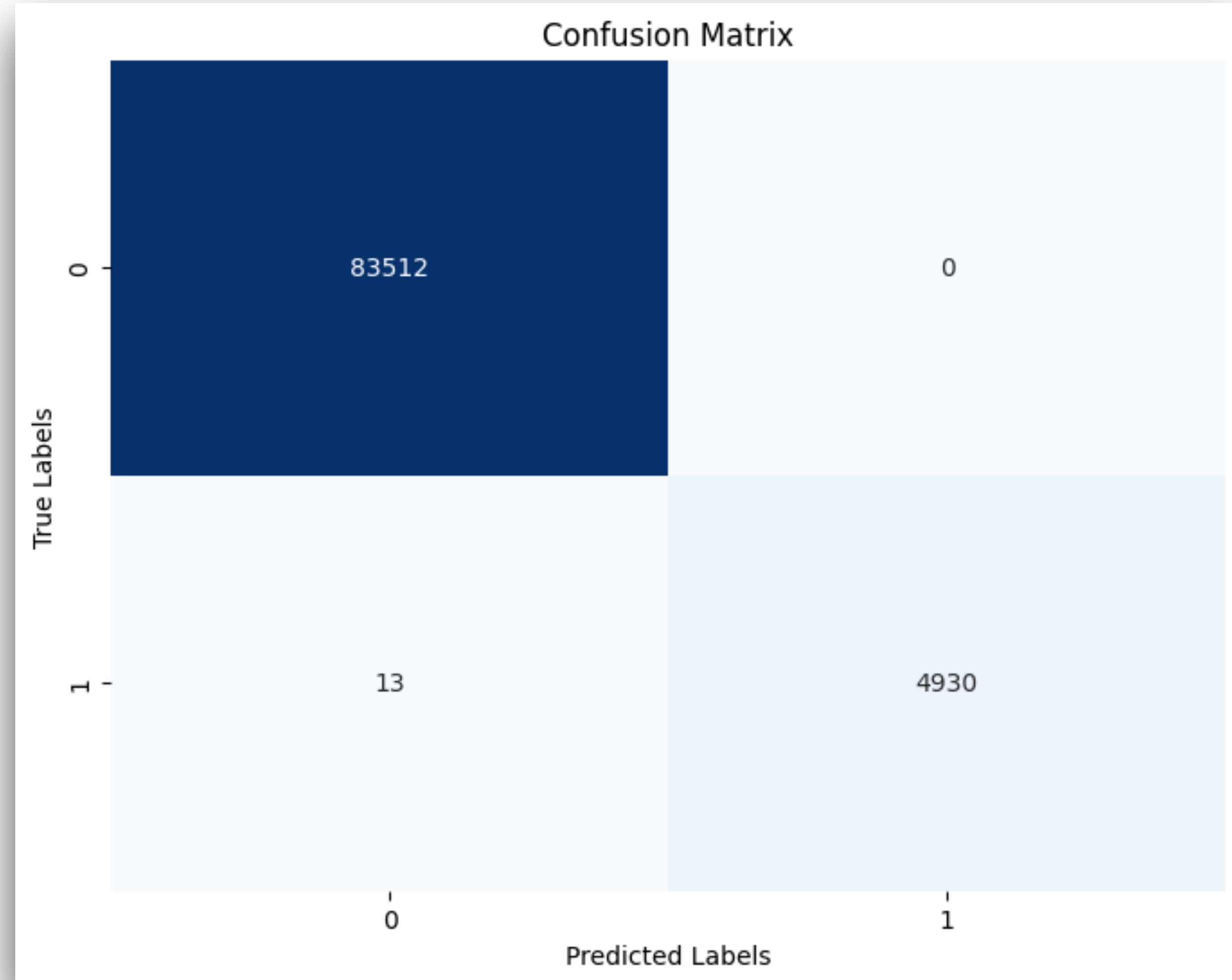
Random Forest

We prepare the train data for our classifier by appending generated embedding (128 features) to our custom features (25). Our labels are 0 and 1 depending on whether our node is non-critical or critical as decided by our threshold. For test data, only features are passed. Both the datasets are kept separate throughout the whole process to prevent data leakage.

Classification Results

Class	Precision	Recall	F1-Score	Support
0 (Non-Critical)	1.00	1.00	1.00	83,512
1 (Critical)	1.00	1.00	1.00	4,943
Accuracy			1.00	88,455
Macro Avg	1.00	1.00	1.00	88,455
Weighted Avg	1.00	1.00	1.00	88,455

Confusion Matrix



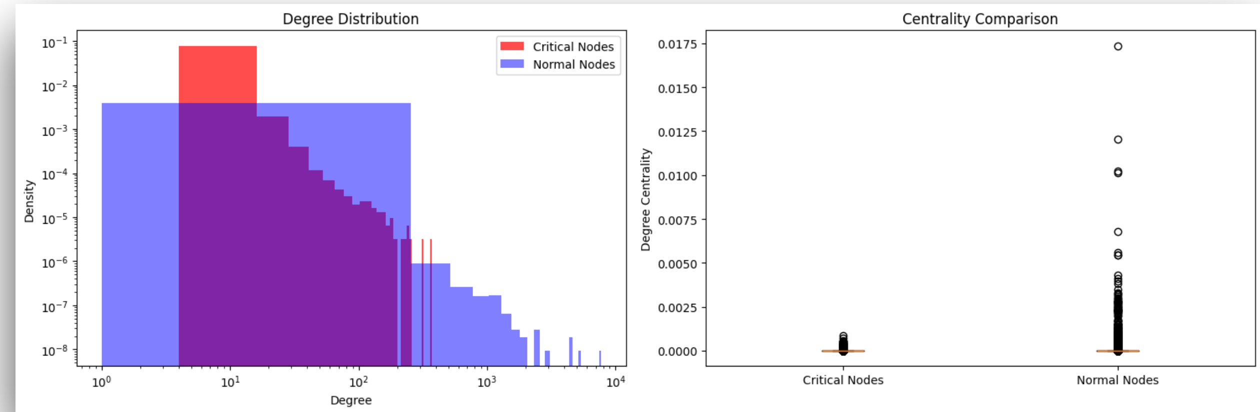
The classifier is extremely robust.

Analysis

Network Analysis:
- Total nodes: 442275
- Critical nodes: 25072
- Non-critical nodes: 417203

Average degree:
- Critical nodes: 6.57
- Normal nodes: 2.00

Both non-critical and critical nodes have 0 average degree centrality and others too for that matter.
Here is a degree distribution:



Degree Centrality:
Mean of top-10 Critical Degree: 7.500878651200133
Mean of top-10 Normal Degree: 4.809857513206448

Betweenness Centrality:
Mean of top-10 Critical Betweenness: 16.30465888033559
Mean of top-10 Normal Betweenness: 16.238758888599698

PageRank:
Mean of top-10 Critical Betweenness: 11.09994490413303
Mean of top-10 Normal Betweenness: 6.2296692282643615

All of the above values are the negative logarithms with natural base for the actual centralities, since they are extremely small given the network size.

Network Metrics

Feature	Mean	Median	Max	Min
degree_centrality	5.11E-06	2.26E-06	1.74E-02	2.26E-06
betweenness_centrality	3.97E-12	0.00	5.11E-07	0.00
pagerank	2.26E-06	1.68E-06	4.56E-03	1.68E-06
clustering_coefficient	0.00	0.00	0.00	0.00
local_risk	1.10	1.00	371.00	0.00

Node Type Indicators

Feature	Mean	Median	Max	Min
is_artifact	0.100	0.00	1.00	0.00
is_release	0.625	1.00	1.00	0.00
is_added_value	0.275	0.00	1.00	0.00

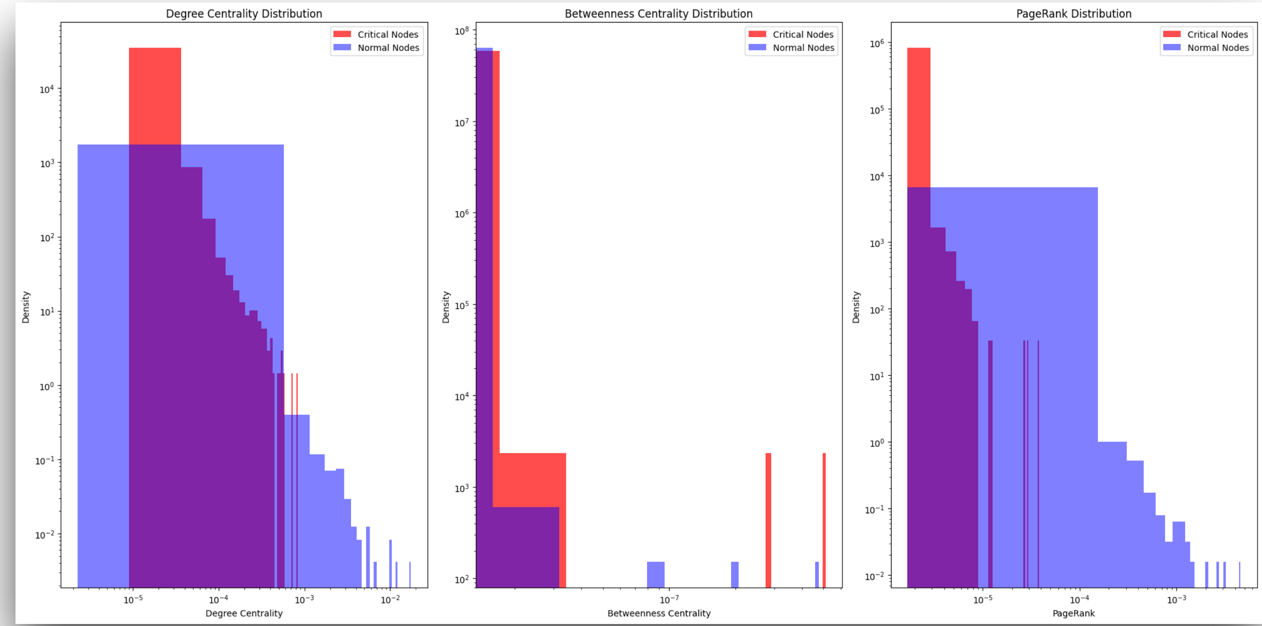
Scope Features

Feature	Mean	Median	Max	Min
scope_compile	0.460	0.00	4410.00	0.00
scope_runtime	0.061	0.00	983.00	0.00
scope_test	0.171	0.00	7030.00	0.00
scope_provided	0.059	0.00	1282.00	0.00
scope_implementation	8.82E-05	0.00	5.00	0.00
scope_runtimeOnly	2.26E-06	0.00	1.00	0.00
scope_system	3.62E-05	0.00	9.00	0.00
scope_optional	3.39E-05	0.00	6.00	0.00
scope_import	9.04E-06	0.00	2.00	0.00
scope_api	4.30E-05	0.00	2.00	0.00
scope_integration-test	2.26E-06	0.00	1.00	0.00
scope_runtme	4.52E-06	0.00	1.00	0.00
scope_external	4.52E-06	0.00	1.00	0.00

Type Features

Feature	Mean	Median	Max	Min
type_POPULARITY_1_YEAR	0.091	0.00	1.00	0.00
type_CVE	0.090	0.00	1.00	0.00
type_FRESHNESS	0.090	0.00	1.00	0.00

Finally, here is the distribution for centralities



Reinforcement Learning Environment

The project implements a custom OpenAI Gym environment (DependencyChainEnv) that:

1. Manages state representation of dependency chains
2. Implements action space for node selection
3. Provides reward function based on multiple objectives:
   - Chain validity
   - Security score
   - Performance metrics
   - Freshness indicators
   - Critical node distribution

State Space

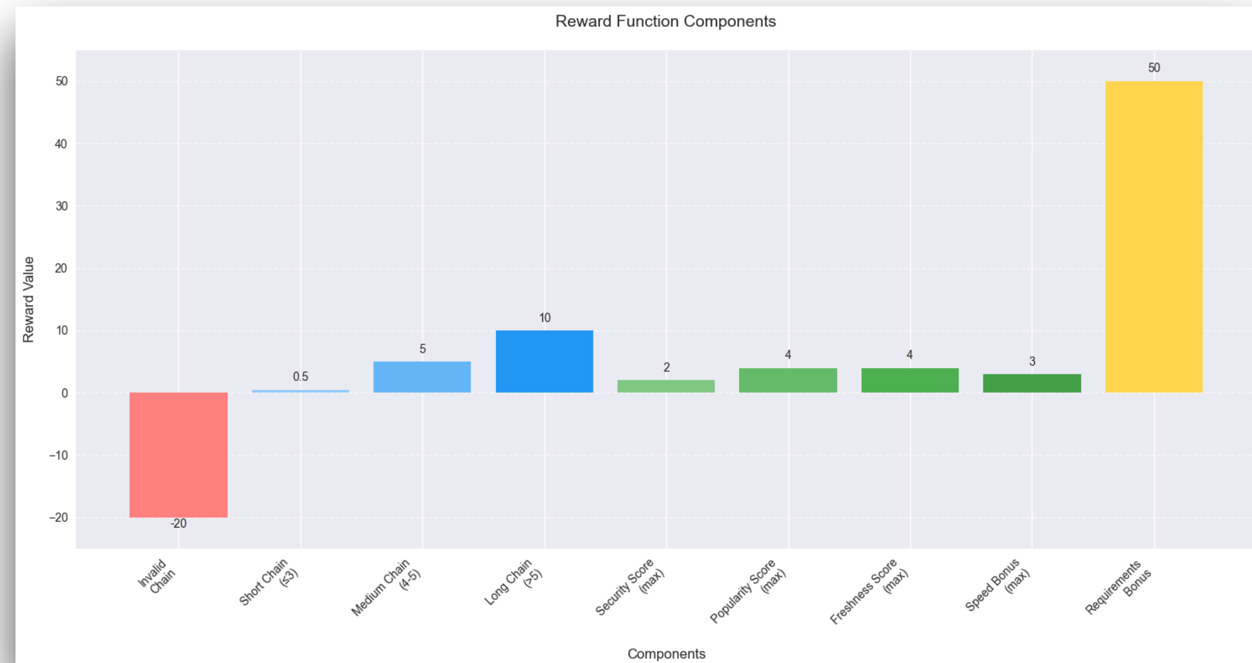
The environment represents states using a combination of:
- Current node features
- Chain metrics
- Historical context
- Quality indicators

Action Space

Actions correspond to selecting the next node in the dependency chain, with constraints ensuring:
- Valid dependencies
- No cycles
- Proper scope progression

Reward Function

The reward function incorporates multiple objectives.



Some of these rewards are static, but some, like security score, popularity score, freshness score and speed bonus are multiplied with the *metric_value* found in the code.

```
local_risks = [self.node_features[node]['local_risk'] for node in chain]
avg_local_risk = np.mean(local_risks)
critical_count = sum(self.critical_nodes[node] for node in chain)
critical_ratio = critical_count / len(chain)
cve_count = sum(self.node_features[node]['type_CVE'] for node in chain)
security_score = np.exp(-cve_count)

perf_scores = [
    1.0 if self.node_features[node]['type_SPEED'] else 0.5
    for node in chain
]
performance_score = np.mean(perf_scores)

freshness_scores = [
    1.0 if self.node_features[node]['type_FRESHNESS'] else 0.5
    for node in chain
]
freshness_score = np.mean(freshness_scores)
```

Training Process

The system uses a curriculum learning approach with three stages:
Stage 1: Short chains (max length 5)
- Focus on basic chain validity
- Relaxed quality requirements

Stage 2: Medium chains (max length 10)
- Increased emphasis on security
- Introduction of performance constraints

Stage 3: Long chains (max length 15)
- Full quality requirements
- Optimization for all metrics

The project uses an Actor-Critic architecture with several key components:

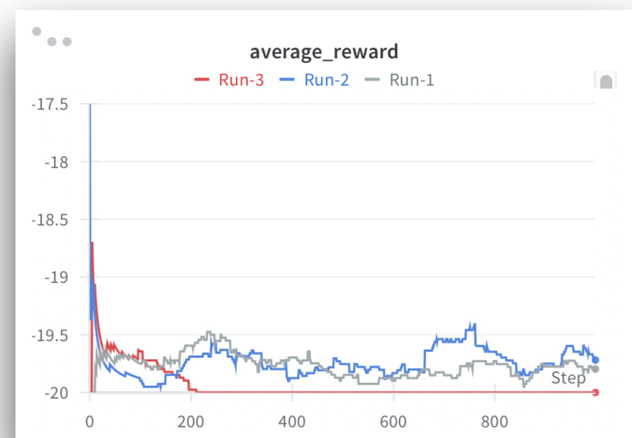
Feature Processing Networks:

Scope network (13 → 32 → 64)
Relationship network (3 → 16 → 32)
Quality network (4 → 16 → 32)

Main Network:

Combined feature processing
Attention mechanism
Separate actor and critic heads

Results and Evaluation



The system was evaluated on multiple metrics:

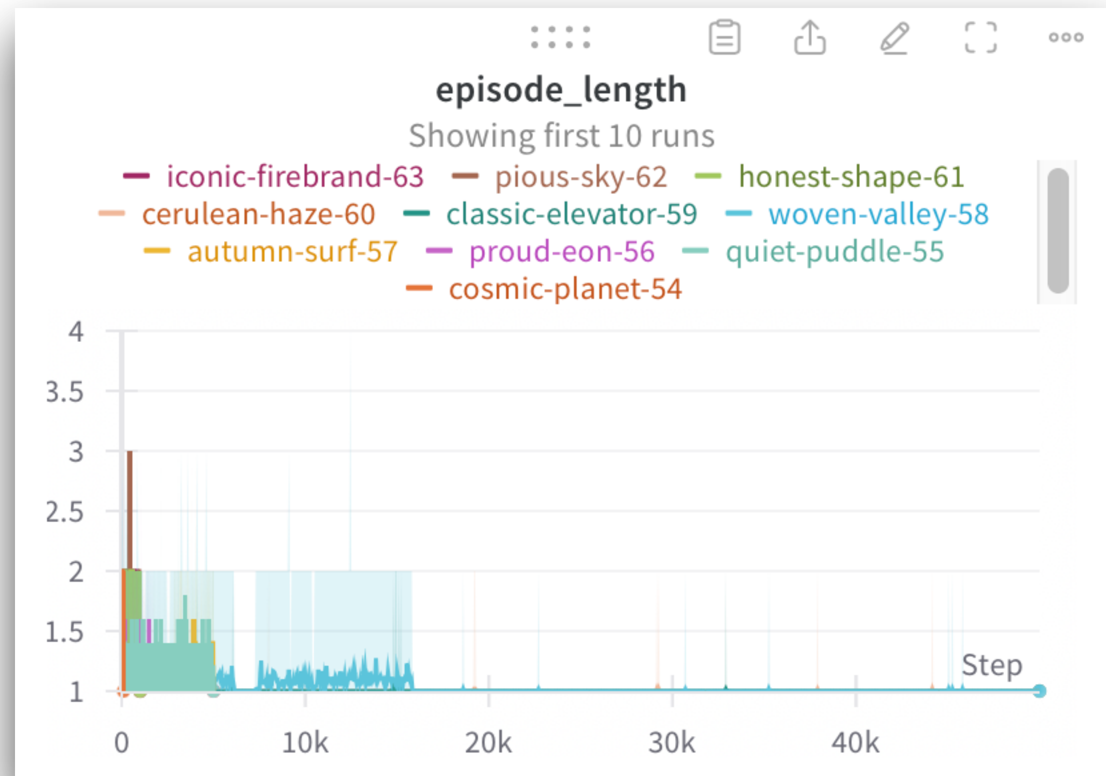
Chain Quality:
- Average security score: 1.0
- Average performance score: 0.5
- Average freshness score: 0.5

Chain Characteristics:
- Average chain length: 1.15
- Valid chain ratio: 100%
- Critical node distribution: Balanced

An example of generated Chain:

Generating sample optimized chain:
Generated chain: 3089324 -> 3089329

Now, while these two nodes (net.exoego:aws-sdk-scalajs-facade-guardduty_sjs1_2.12) -> (net.exoego:aws-sdk-scalajs-facade-guardduty_sjs1_2.12:0.30.0-v2.715.0) are connected, extensive analysis will be required to verify, whether the connection was actually meaningly. Furthermore, the model only demonstrated ability to create a chain of length 4 at max.



Future Improvements

The first improvement would be to get the model working for a real life scenario. Apart from this, we can have the following improvements:

Model Enhancements:
- Add transformer-based feature processing
- Explore multi-agent approaches

Feature Engineering:
- Include temporal dependency patterns
- Add compatibility scores
- Incorporate usage statistics

Training Optimizations:
- Add prioritized sampling
- Explore meta-learning approaches

Conclusion

The project successfully serves as a proof of concept for the feasibility of using machine learning for dependency chain optimization. The combination of graph-based analysis and reinforcement learning provides a flexible framework for generating high-quality dependency chains while balancing multiple objectives.