Bridging Operator Semantic Inconsistencies: A Source-Level Cross-Framework Model Conversion Approach

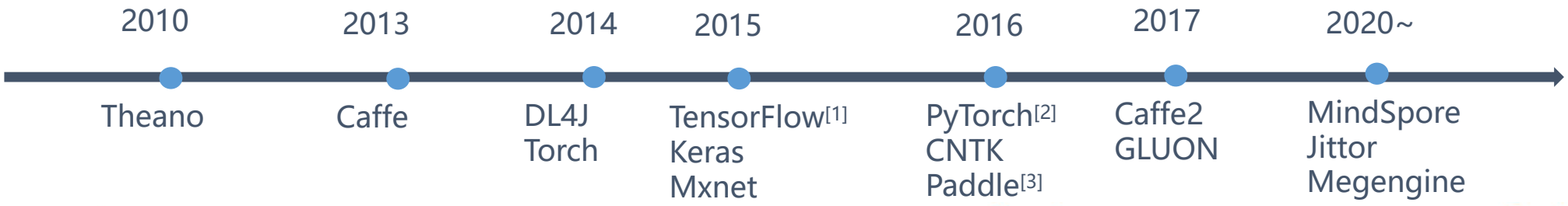
Xingpei Li[1], Yan Lei[2], Zhouyang Jia[1], Yuanliang Zhang[1], Haoran Liu[1], Liqian Chen[1], Wei Dong[1], Shanshan Li[1]

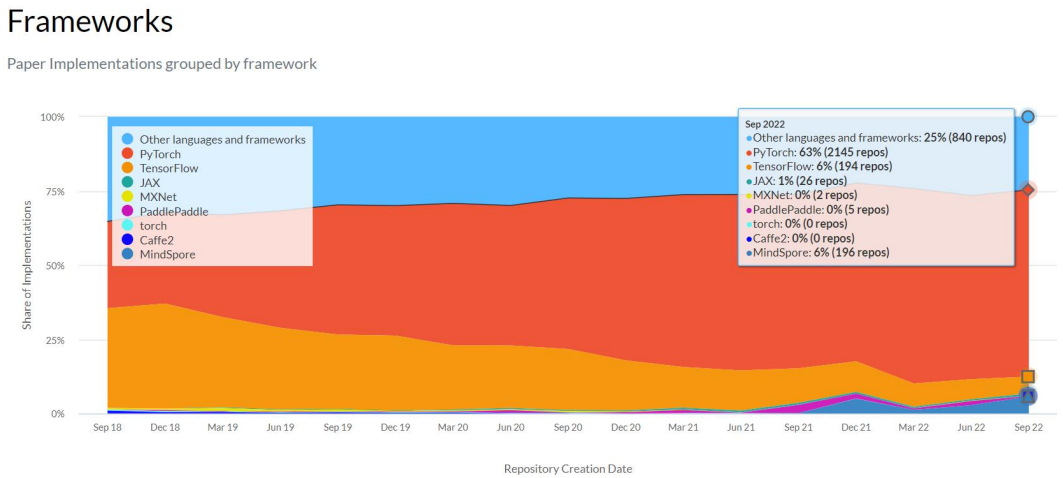
[1]*College of Computer Science and Technology*
National University of Defense Technology
Changsha, China

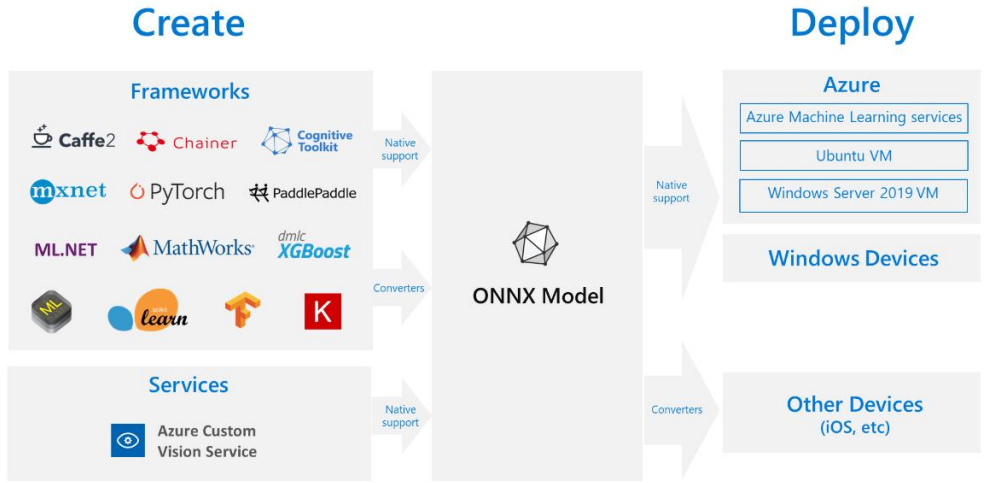
[2]*School of Big Data & Software Engineering*
Chongqing University
Chongqing, China

Background

- The rise of DL frameworks underscores the importance of model reuse





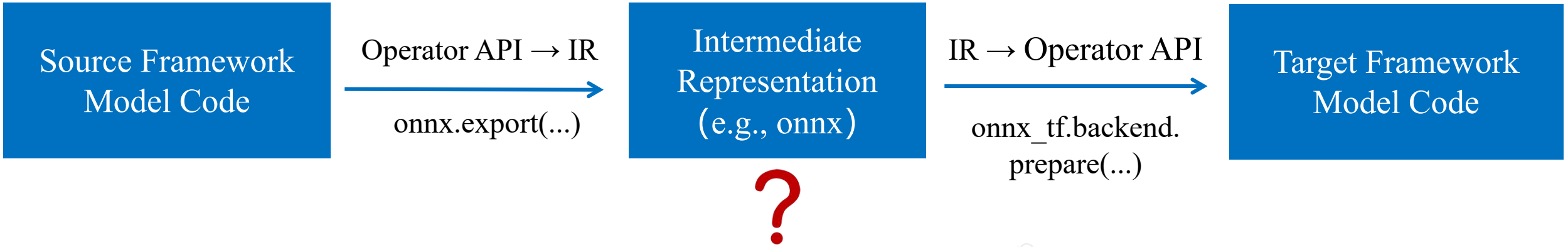


- High-quality cross-framework conversion is critical to ensure consistent model performance and interoperability

[1] Abadi M, Barham P, Chen J, et al. TensorFlow: a system for Large-Scale machine learning[C]//12th USENIX symposium on operating systems design and implementation (OSDI 16). 2016: 265-283.
[2] Paszke A, Gross S, Massa F, et al. Pytorch: An imperative style, high-performance deep learning library[J]. Advances in neural information processing systems, 2019, 32.
[3] Ma Y, Yu D, Wu T, et al. PaddlePaddle: An open-source deep learning platform from industrial practice[J]. Frontiers of Data and Domputing, 2019, 1(1): 105-115.

Background

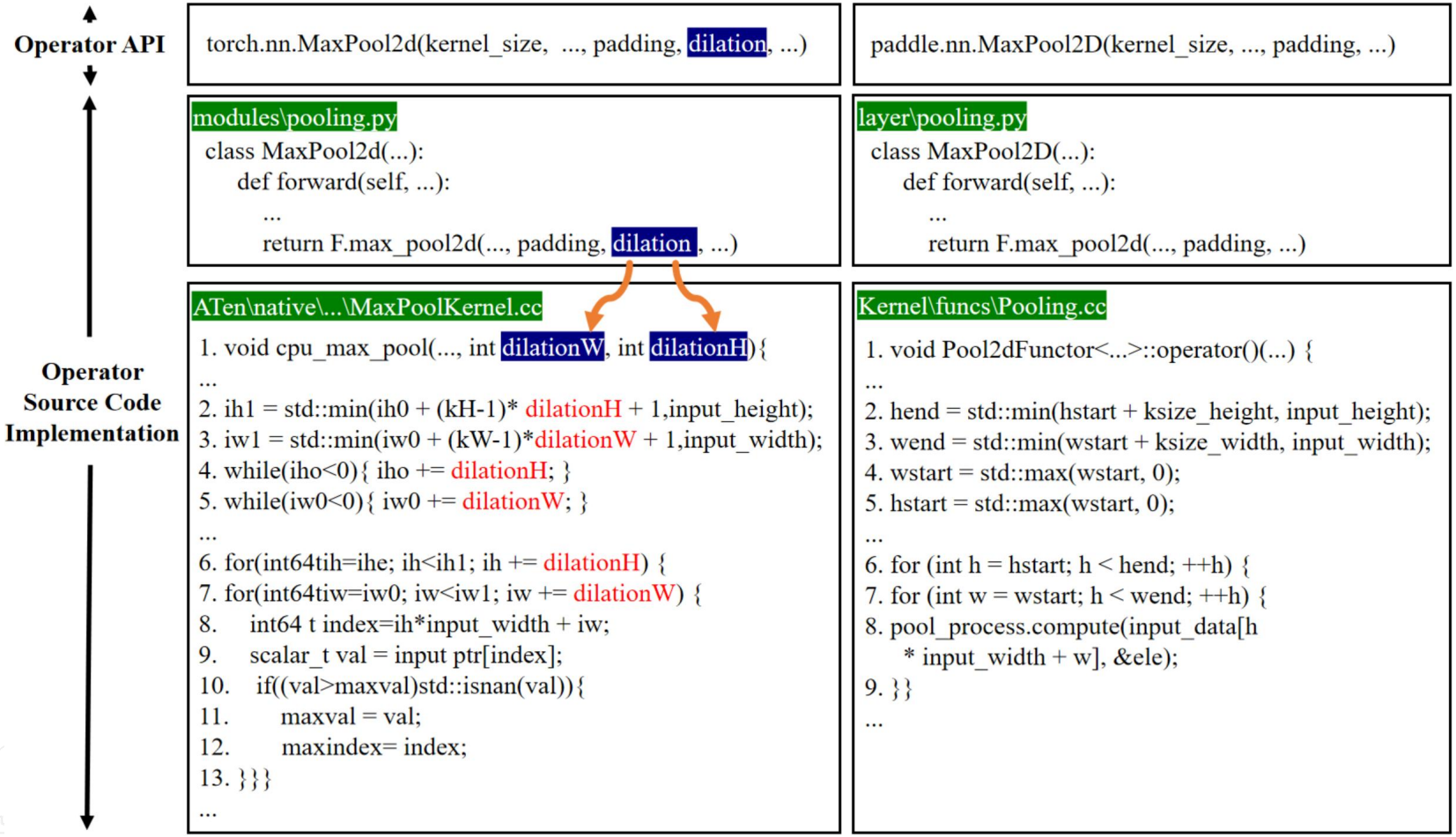
- Current converters can map API syntax but ignore operator implementation, causing semantic inconsistency
  - Transpiling operator names/parameters via graph structures[4,5]
  - Standardizing operator interfaces as cross-framework APIs[6,7,8]



[4] ONNX. 2017. Open Neural Network Exchange. https://onnx.ai/.
[5] Yu Liu, Cheng Chen, Ru Zhang, Tingting Qin, Xiang Ji, Haoxiang Lin, and Mao Yang. 2020. Enhancing the interoperability between deep learning frameworks by model conversion. In Proceedings of the 28th ACM joint meetingon European software engineering conference and symposium on the foundations of software engineering. 1320–1330.
[6] Daniel Lenton, Fabio Pardo, Fabian Falck, Stephen James, and Ronald Clark. 2021. Ivy: Templated deep learning for inter-framework portability. arXiv preprint arXiv:2102.02886 (2021).
[7] Baidu. 2022. PaConvert. https://github.com/PaddlePaddle/PaConvert.
[8] Linyuan Gong, Jiayi Wang, and Alvin Cheung. 2024. ADELT: transpilation between deep learning frameworks. In Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence. 6279–6287.

Motivation

- Semantic inconsistency potentially cause crash or performance issue
  - These inconsistencies are rooted in DL framework source code



Motivation

- Limitation of Related Works
  - **Graph-based Model Converters**[4,5]**:** Treating operators as black-box nodes, ignoring framework source code
  - **API-based Model Converters**[6,7,8]**:** Treating operators as uniform APIs, hiding framework source code difference

[4] ONNX. 2017. Open Neural Network Exchange. https://onnx.ai/.
[5] Yu Liu, Cheng Chen, Ru Zhang, Tingting Qin, Xiang Ji, Haoxiang Lin, and Mao Yang. 2020. Enhancing the interoperability between deep learning frameworks by model conversion. In Proceedings of the 28th ACM joint meetingon European software engineering conference and symposium on the foundations of software engineering. 1320–1330.
[6] Daniel Lenton, Fabio Pardo, Fabian Falck, Stephen James, and Ronald Clark. 2021. Ivy: Templated deep learning for inter-framework portability. arXiv preprint arXiv:2102.02886 (2021).
[7] Baidu. 2022. PaConvert. https://github.com/PaddlePaddle/PaConvert.
[8] Linyuan Gong, Jiayi Wang, and Alvin Cheung. 2024. ADELT: transpilation between deep learning frameworks. In Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence. 6279–6287.

Motivation

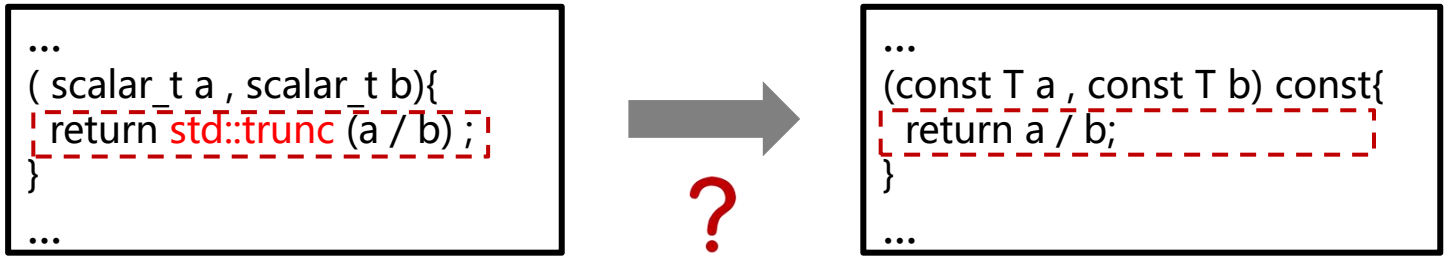
- Limitation of Related Works
  - **Graph-based Model Converters**[4,5]**:** Treating operators as black-box nodes, ignoring framework source code
  - **API-based Model Converters**[6,7,8]**:** Treating operators as uniform APIs, hiding framework source code difference

Modifying framework source code to bridge operator semantic inconsistencies during model conversion

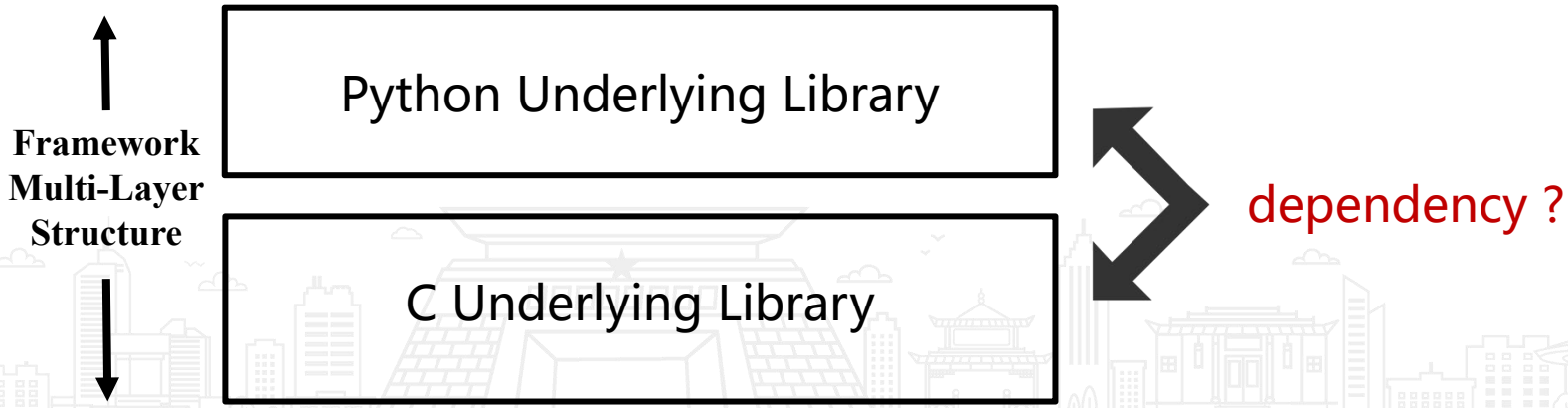
[4] ONNX. 2017. Open Neural Network Exchange. https://onnx.ai/.
[5] Yu Liu, Cheng Chen, Ru Zhang, Tingting Qin, Xiang Ji, Haoxiang Lin, and Mao Yang. 2020. Enhancing the interoperability between deep learning frameworks by model conversion. In Proceedings of the 28th ACM joint meetingon European software engineering conference and symposium on the foundations of software engineering. 1320–1330.
[6] Daniel Lenton, Fabio Pardo, Fabian Falck, Stephen James, and Ronald Clark. 2021. Ivy: Templated deep learning for inter-framework portability. arXiv preprint arXiv:2102.02886 (2021).
[7] Baidu. 2022. PaConvert. https://github.com/PaddlePaddle/PaConvert.
[8] Linyuan Gong, Jiayi Wang, and Alvin Cheung. 2024. ADELT: transpilation between deep learning frameworks. In Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence. 6279–6287.

Challenge

- Two key challenges:
  - Extracting relevant framework code and locating modification point between frameworks



  - Aligning framework code across different layers within the target framework



Contribution

- First empirical study on operator semantic inconsistencies in cross-framework conversion
- Source-level DL converter (ModelX) overcoming API mapping limitations via modifying framework source code
- PyTorch↔Paddle auto-conversion with superior reliability (52 multi-domain models) vs. ONNX/PaConvert

Empirical Study

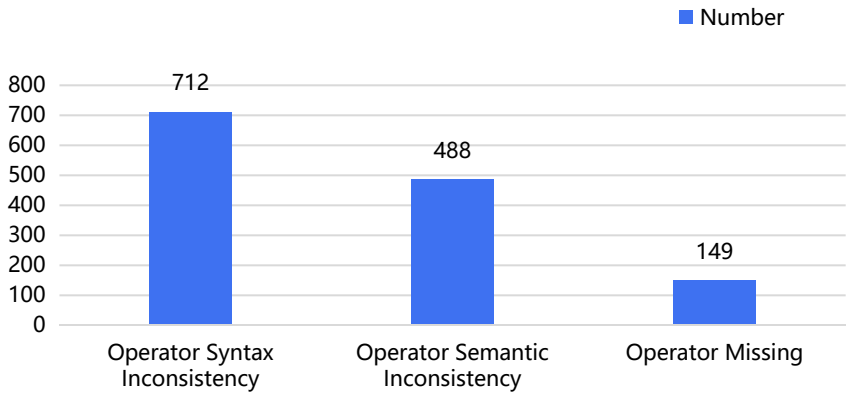
- Investigated operator inconsistencies in 1,349 PyTorch↔Paddle conversions

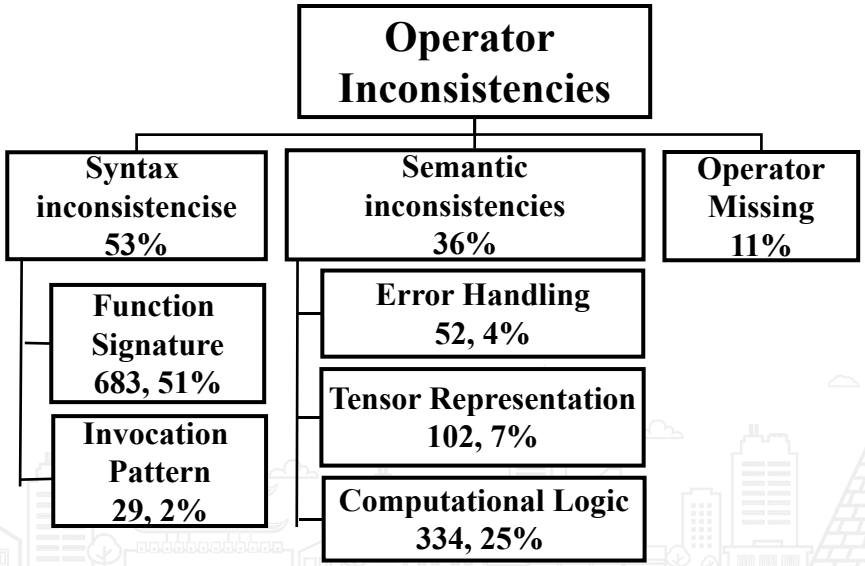
Empirical Study

- Investigated operator inconsistencies in 1,349 PyTorch↔Paddle conversions
- **Methodology Workflow：**
  - Manual API mapping analysis
  - Automated Framework code tracing for semantic inconsistencies
  - Operator mapping table (686 PyTorch operators; 663 Paddle operators)
  - Taxonomy of operator inconsistencies

Empirical Study

- 47% of operators alter semantics from divergent root causes






```python
# Syntax Inconsistency in Signature
torch.cat(tensor1, dim=0)
torch.from_dlpack(...)

# Syntax Inconsistency in Usage
import torch.optim as optim
import torch,optim.lr_scheduler as lr
Optimizer=optim.Adam(...)
Scheduler=lr.StepLR(Optimizer, ...)
```

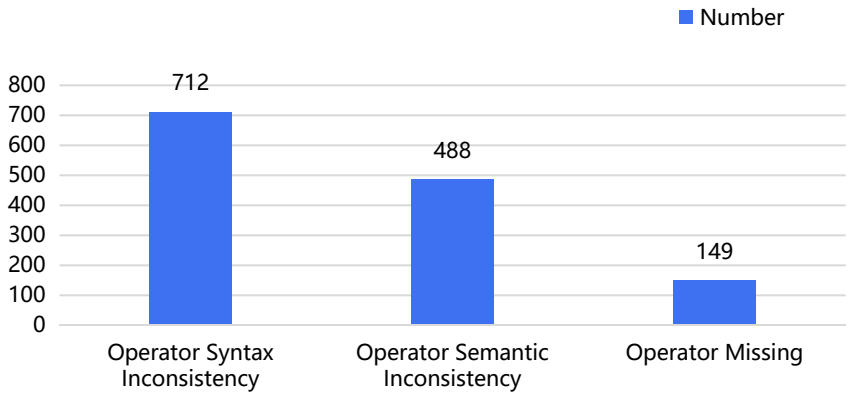
```python
# Syntax Inconsistencies in Signature
paddle.concat(tensor1, axis=0)
paddle.utils.dlpack.from_dlpack(...)

# Syntax Inconsistency in Usage
import paddle.optimizer as optimizer
import paddle.optimizer.lr as lr
Scheduler=lr.StepDecay(...)
Optimizer=optimizer.Adam(Scheduler, ...)
```

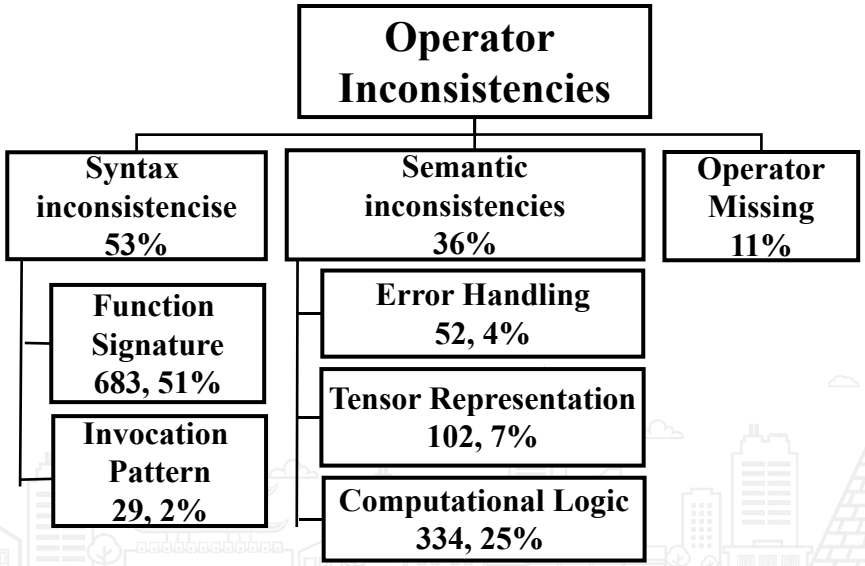
```python
// Semantic Inconsistency
torch.divide(a, b, rounding_mode=Trunc)

void div_trunc_kernel(auto& iter) {
...
cpu_kernel_vec(iter,
[](scalar_t a, scalar_t b){
   return std::trunc(a / b);},
[](...){
   return (a / b).trunc();});
});}
```

```python
// Semantic Inconsistency
paddle.divide(a, b)

struct DivideFunctor{
...
inline HOSTDEVICE T operator()(const T a
    , const T b) const{
   return a / b;
  }
...
};
```

Empirical Study

- 47% of operators alter semantics from divergent root causes
- Critical to resolve via type-specific alignment






```python
# Syntax Inconsistency in Signature
torch.cat(tensor1, dim=0)
torch.from_dlpack(...)

# Syntax Inconsistency in Usage
import torch.optim as optim
import torch,optim.lr_scheduler as lr
Optimizer=optim.Adam(...)
Scheduler=lr.StepLR(Optimizer, ...)
```

```python
# Syntax Inconsistencies in Signature
paddle.concat(tensor1, axis=0)
paddle.utils.dlpack.from_dlpack(...)

# Syntax Inconsistency in Usage
import paddle.optimizer as optimizer
import paddle.optimizer.lr as lr
Scheduler=lr.StepDecay(...)
Optimizer=optimizer.Adam(Scheduler, ...)
```

```python
// Semantic Inconsistency
torch.divide(a, b, rounding_mode=Trunc)

void div_trunc_kernel(auto& iter) {
...
cpu_kernel_vec(iter,
[](scalar_t a, scalar_t b){
   return std::trunc(a / b);},
[](...){
   return (a / b).trunc();});
});}
```

```python
// Semantic Inconsistency
paddle.divide(a, b)

struct DivideFunctor{
...
inline HOSTDEVICE T operator()(const T a
    , const T b) const{
   return a / b;
  }
...
};
```

Empirical Study

- Semantic-inconsistency code in layers without inter-dependencies

```
// operator API
torch.nn.Conv2d (...,dliation, ...)

//python underlying library
class MaxPool2d(...):
   def __init__(self, daliation, ...)
      self. dilation = dilation
      ...
      if isinstance(self. dilation , ...
          raise ValueError(f"...")
      ...

//C underlying library
void div_trunc_kernel(auto& iter) {
...
ih1=std::min(ih0+(kH-1)* dilationH +1,
input_height);
iw1 = std::min(iw0+(kW-1)* dilationW +1,
input_width);
while(iho<0){ iho+= dilationH ; }
while(iw0<0){ iw0+= dilationW ; }
...
for(ih=ihe; ih<ih1; ih+= dilationH ) {
for(iw=iw0; iw<iw1; iw+= dilationW ) {
...
```

```
// operator API
paddle.nn.MaxPool2D(...)

//python underlying library
class MaxPool2D(...):
    def __init__(self, ...)
        // no parameter initializing
        ...
        // no error handling

        ...

//C underlying library
void Pool2dFunctor(...) {
...
hend = std::min(hstart + ksize_height
, input_height);
wend = std::min(wstart + ksize_width
, input_width);
wstart = std::max(wstart, 0);
hstart = std::max(wstart, 0);
...
for (h=hstart; h<hend; ++h) {
for ( h=hstart; h<hend; ++h) {
...
```

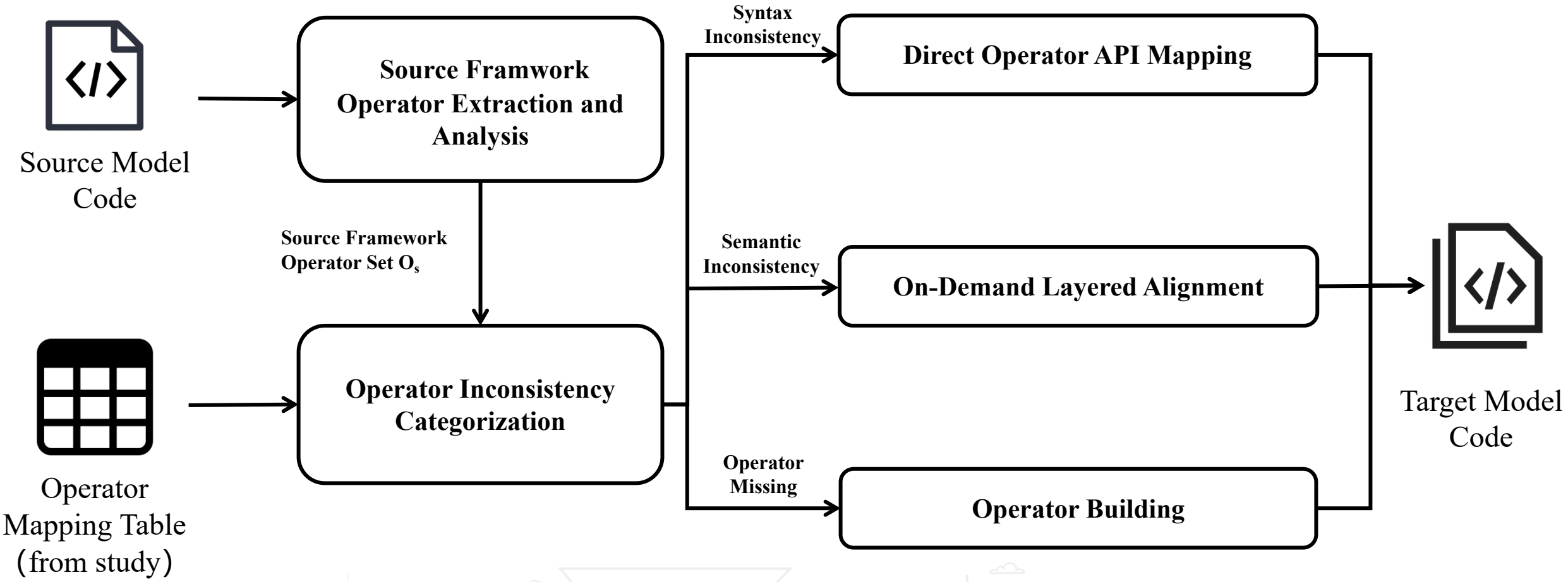
Empirical Study

- Semantic-inconsistency code in layers without inter-dependencies
- Allow per-layer isolation during resolution

```
// operator API
torch.nn.Conv2d (...,dliation, ...)

//python underlying library
class MaxPool2d(...):
   def __init__(self, daliation, ...)
      self. dilation = dilation
      ...
      if isinstance(self. dilation , ...
          raise ValueError(f"...")
      ...

//C underlying library
void div_trunc_kernel(auto& iter) {
...
ih1=std::min(ih0+(kH-1)* dilationH +1,
input_height);
iw1 = std::min(iw0+(kW-1)* dilationW +1,
input_width);
while(iho<0){ iho+= dilationH ; }
while(iw0<0){ iw0+= dilationW ; }
...
for(ih=ihe; ih<ih1; ih+= dilationH ) {
for(iw=iw0; iw<iw1; iw+= dilationW ) {
...
```

```
// operator API
paddle.nn.MaxPool2D(...)

//python underlying library
class MaxPool2D(...):
    def __init__(self, ...)
        // no parameter initializing
        ...
        // no error handling

        ...

//C underlying library
void Pool2dFunctor(...) {
...
hend = std::min(hstart + ksize_height
, input_height);
wend = std::min(wstart + ksize_width
, input_width);
wstart = std::max(wstart, 0);
hstart = std::max(wstart, 0);
...
for (h=hstart; h<hend; ++h) {
for ( h=hstart; h<hend; ++h) {
...
```

Desgin


Source Model Code
Source Framwork Operator Extraction and Analysis
Source Framework Operator Set O_s
Operator Mapping Table (from study)
Operator Inconsistency Categorization
Syntax Inconsistency
Direct Operator API Mapping
Semantic Inconsistency
On-Demand Layered Alignment
Operator Missing
Operator Building
Target Model Code

Desgin

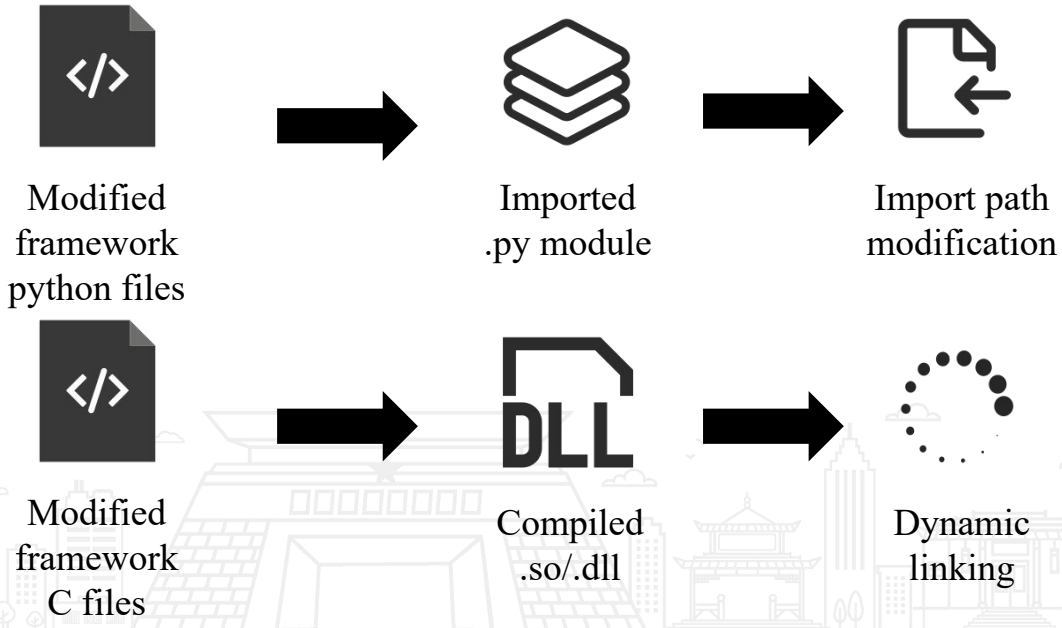
- **Operator Mapping Flow：**
  - Direct Operator API Mapping
    1. **Mapping API name and parameters**

Desgin

- **Operator Mapping Flow：**
  - Direct Operator API Mapping
  - On-Demand Layered Alignment

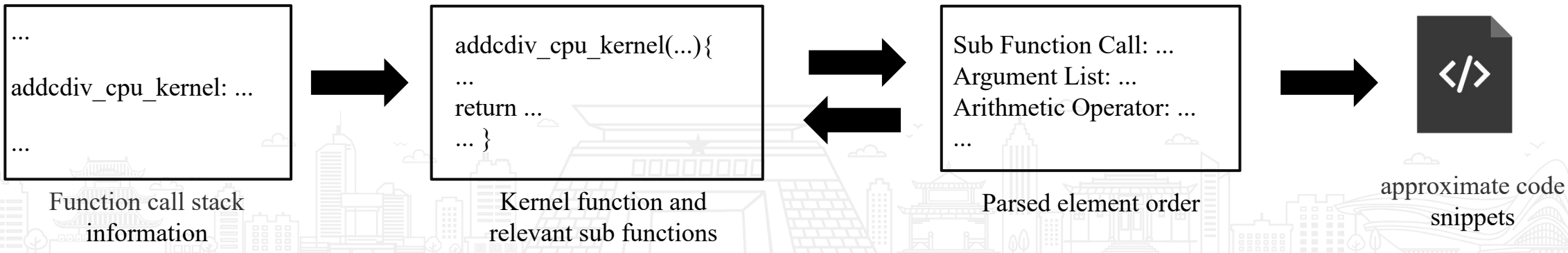
1. Mapping partial operator API

2. Aligning framework code to reconcile incompatible API parameters



Desgin

- **Operator Mapping Flow:**
  - Direct Operator API Mapping
  - On-Demand Layered Alignment
  - Operator Building
    1. **Analyzing function call stack**
    2. **Parsing function call**
    3. **Generating equivlant approximate code**



Evaluation

- **Research Question**
  - Performance vs. SOTA
    - Compare with ONNX, PaConvert, and LLMs(GPT-4o, GPT-3.5, DeepSeek-Coder)
  - Reliability & Equivalence
    - Test on 686 sampled PyTorch operators
    - Assess conversion success rate and error metrics (MAE/RMSE)
  - Robustness
    - Test 52 models in 3 domains: Vision (41), Text (3), Audio (8)

Evaluation

Performance vs. SOTA

Model type	Tool	Evaluation Latency (s)	Metrics Precision	Recall	F1 Score	Model type	Tool	Evaluation Latency (s)	Metrics Precision	Recall	F1 Score
AlexNet	ONNX	**116.01**	0.526	0.5256	0.5187	VGG	ONNX	116.54	0.6934	0.6893	0.6852
	PaConvert	119.52	0.526	0.5256	0.5187		PaConvert	117.12	0.6934	0.6893	0.6852
	ModelX	116.24	0.526	0.5256	0.5187		ModelX	**115.56**	0.6934	0.6893	0.6852
DenseNet	ONNX	Not supported				ShuffleNet	ONNX	Not supported			
	PaConvert	Not supported					PaConvert	Not supported			
	ModelX	**125.63**	**0.7507**	**0.7451**	**0.7423**		ModelX	**119.35**	**0.6793**	**0.6748**	**0.6708**
ResNet	ONNX	118.51	0.7418	0.737	0.7338	Inception3	ONNX	119.44	0.6801	0.6676	0.6635
	PaConvert	120.01	0.7417	0.7369	0.7337		PaConvert	Not supported			
	ModelX	**118.44**	**0.7642**	**0.7575**	**0.7614**		ModelX	**117.59**	**0.6921**	**0.6851**	**0.6812**

Note: (1) DenseNet, ResNet, ShuffleNet, and Inception3 **have operator semantic inconsistencies**, while AlexNet and VGG do not; (2) For each model type, latency and performance are averaged over **10 warm runs per instance**, then across instances (see Table 4).

- Evaluation metrics increased by approximately 2% on average
- Inference latency was lower (0.46% lower than ONNX; 1.50% lower than PaConvert)

Evaluation

- **Performance vs. SOTA**

Model type	LLM	Prompt Type	All Cases Passed	Error Type	Model type	LLM	Prompt Type	All Cases Passed	Error Type
AlexNet	ChatGPT-3.5	Original, COT	Success	-	ShuffleNet	ChatGPT-3.5	Original, COT	Failed	Syntax
	DeepSeek-Coder	Original, COT	Success	-		DeepSeek-Coder	Original, COT	Failed	Syntax
	ChatGPT-4o	Original, COT	Success	-		ChatGPT-4o	Original	Failed	Syntax
							COT	Success	-
DenseNet	ChatGPT-3.5	Original, COT	Failed	Syntax	VGG	ChatGPT-3.5	Original, COT	Failed	Syntax
	DeepSeek-Coder	Original, COT	Failed	Syntax		DeepSeek-Coder	Original, COT	Failed	Syntax
	ChatGPT-4o	Original, COT	Failed	Syntax		ChatGPT-4o	Original, COT	Success	-
ResNet	ChatGPT-3.5	Original, COT	Failed	Syntax	Inception3	ChatGPT-3.5	Original, COT	Failed	Syntax
	DeepSeek-Coder	Original, COT	Failed	Syntax		DeepSeek-Coder	Original, COT	Failed	Syntax
	ChatGPT-4o	Original, COT	Failed	Semantic		ChatGPT-4o	Original, COT	Failed	Syntax

- ChatGPT-4o performs optimally (9/18), outperforming the other two models
- LLMs cannot resolve semantic inconsistencies in operators – they can only modify syntactic interfaces, but cannot supplement underlying source code.

Evaluation

- **Reliability & Equivalence**

Operator Type	Number of Sampled Operators	Conversion Success Rate	Avg. Conversion Time (ms)	Metrics	
				Avg. MAE	Avg. RMSE
Tensor Ops	278	97.12% (270/278)	948.67	4.57×10^{-7}	2.13×10^{-6}
Layer Ops	190	98.95% (188/190)	924.14	3.82×10^{-6}	1.43×10^{-6}
Other Ops	218	76.14% (166/218)	1102.21	1.67×10^{-5}	4.36×10^{-5}

- Successfully converted 91% of PyTorch operators (624/686)
  - with a >95% success rate for critical categories (e.g., tensor operators and layer operators)
- The unsupported 9% primarily stems from:
  - Dependencies on framework-specific mechanisms
  - Prohibitively high migration costs

Evaluation

- **Robustness**

Fields	Datasets	Metrics	Frameworks		Metric Gap	ONNX Baseline	PaConvert Baseline
			PyTorch	Paddle			
Vision	CIFAR10	Precision	0.8222	0.7626	**0.0334**	0.0661	0.0680
		Recall	0.7698	0.7486			
		F1 Socre	0.7685	0.7490			
	CIFAR100	Precision	0.4897	0.4535	**0.0335**	0.0685	0.0713
		Recall	0.4603	0.4289			
		F1 Socre	0.4534	0.4204			
	FashionMNIST	Precision	0.9133	0.8956	**0.0188**	0.0490	0.0374
		Recall	0.9098	0.8918			
		F1 Socre	0.9093	0.8887			
Text	IMDB	Accruacy	0.7701	0.7824	**0.0261**	0.0572	0.0450
		Precision	0.7607	0.8065			
		Recall	0.7886	0.7431			
		F1 Socre	0.7742	0.7734			
Audio	Urbansound8K	Accuracy	0.3160	0.3042	**0.0097**	0.0380	0.0321
		Precision	0.2838	0.2770			
		Recall	0.3138	0.3243			

- ModelX consistently maintains the smallest performance gap across vision, text, and audio tasks, outperforming baseline tools.

Thanks!

Contact: lixingpei123@nudt.edu.cn