

Formalización enfocada a operaciones de Tensores en PyTorch

FECHA

2025-10-07

ESTUDIANTE

Jorge Cruces

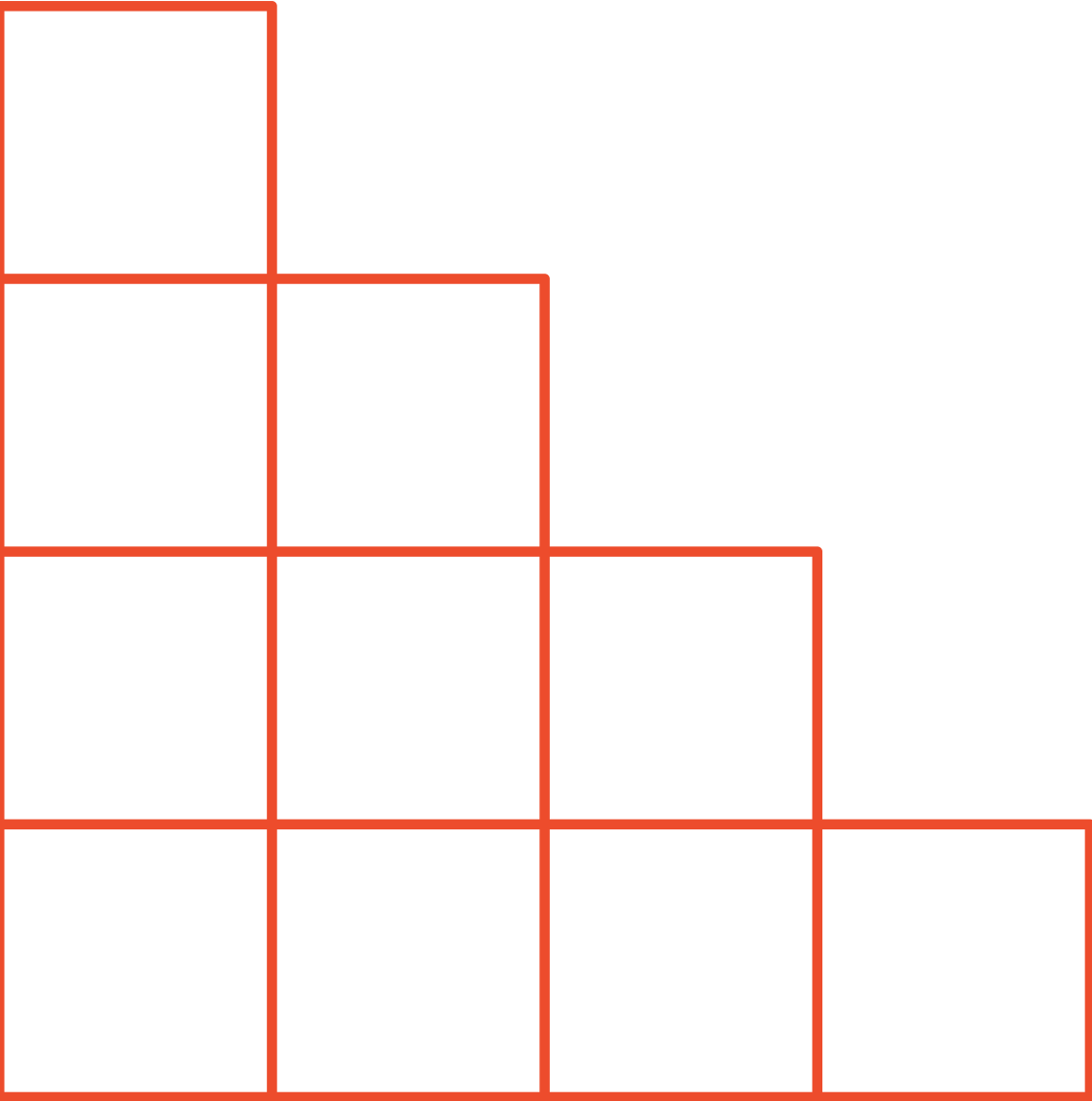
PROFESORES GUÍAS

Matías Toro
Éric Tanter

COMISIÓN

Luis Mateu
Valentin Barriere

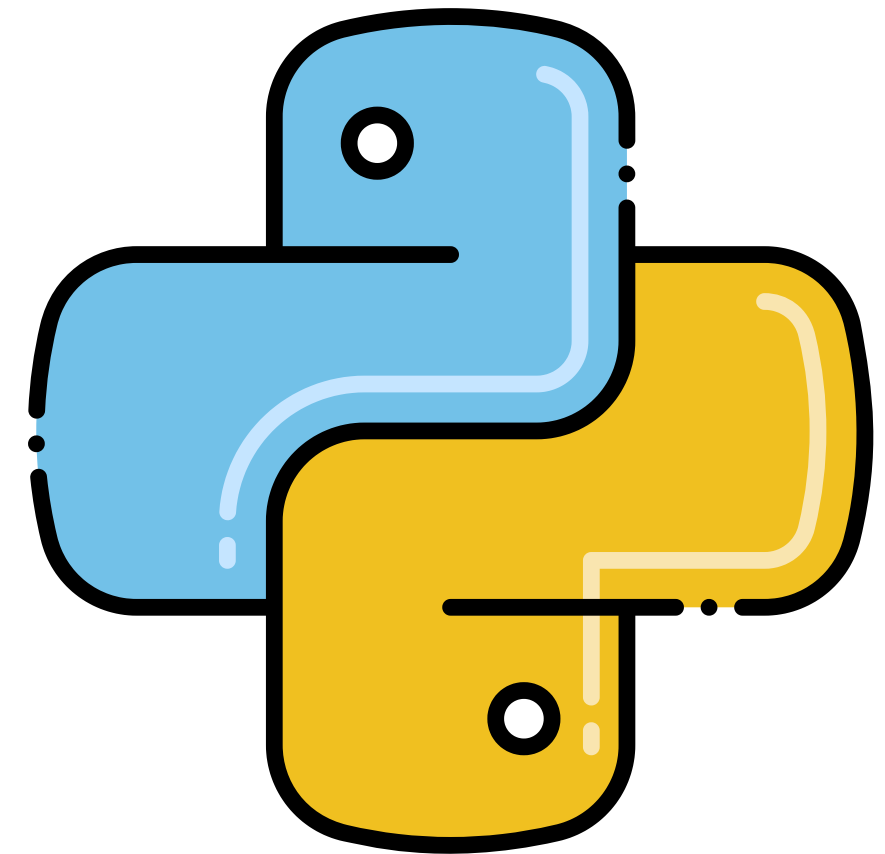
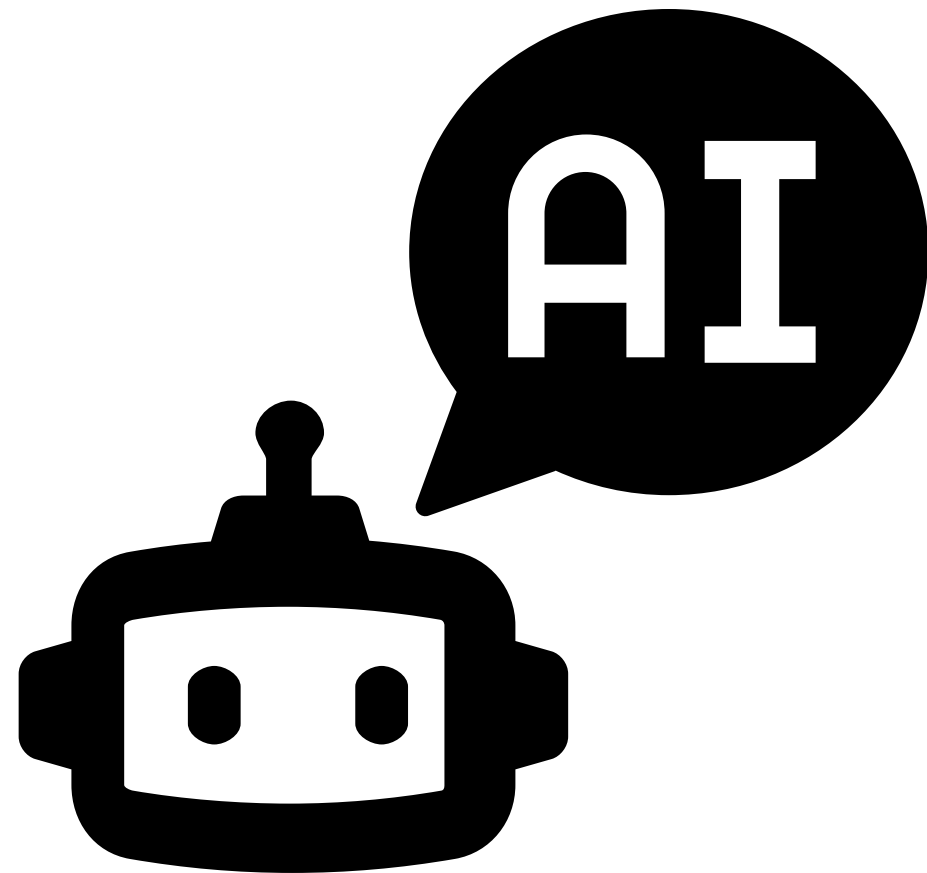




03	Introducción
07	Objetivos y Metodologia
09	Recolección de Operaciones y resultados
15	Literatura y herramientas existentes
19	Gramática
30	Evaluación
35	Conclusión

Avance de la IA y la importancia de Python

03/36



Frameworks actuales

04/36



Tensorflow

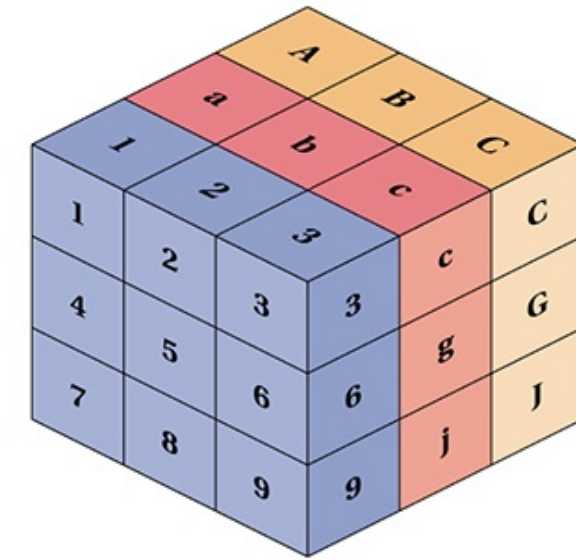


PyTorch

Tensores

Un tensor es una estructura matemática que generaliza los conceptos de escalares, vectores y matrices a dimensiones superiores.

En el ámbito de este trabajo, un **tensor es simplemente un arreglo multidimensional de datos**.



05/36



1. $S \rightarrow a$
2. $S \rightarrow SS$
3. $aSa \rightarrow b$

Una gramática formal es un **conjunto de reglas** que describe cómo se pueden generar cadenas de un lenguaje.

Gramática Formal

SMT Solvers

SMT Solvers (Satisfiability Modulo Theories Solvers) son herramientas que permiten verificar si una fórmula lógica es **satisfacible** bajo ciertas teorías matemática.



Ejemplo de un programa en PyTorch

06/36

```
# 2 frases, cada una con 4 palabras
data = torch.randn(2, 4, 3) # [2 frases, 4 palabras, 3 dimensiones]
labels = torch.tensor([1, 0]) # Etiquetas: positiva (1), negativa (0)

class SentimentClassifier(nn.Module):
    def __init__(self):
        super(SentimentClassifier, self).__init__()
        self.fc1 = nn.Linear(4 * 3, 8) # Aplana: 4 palabras * 3 dim = 12
        self.fc2 = nn.Linear(8, 2)      # Salida: 2 clases (positiva/negativa)

    def forward(self, x):
        batch_size = x.size(0)
        x = x.reshape(batch_size, 4 * 8) # Reshape: [batch, 4, 3] -> [batch, 12]
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

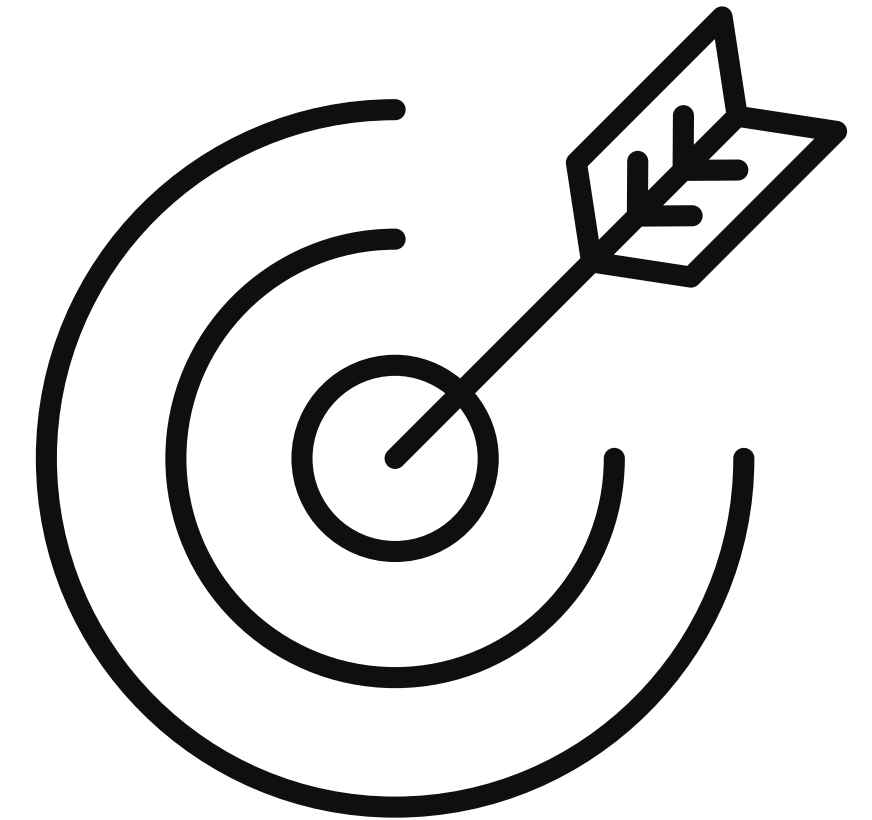
RuntimeError: shape '[2, 32]' is invalid for input of size 24

Objetivo General

Desarrollar una **formalización** para las operaciones tensoriales más utilizadas en proyectos de deep learning basados en **PyTorch**, con el fin de verificar automáticamente la coherencia dimensional en programas que combinan múltiples operaciones tensoriales.

Objetivos Especificos

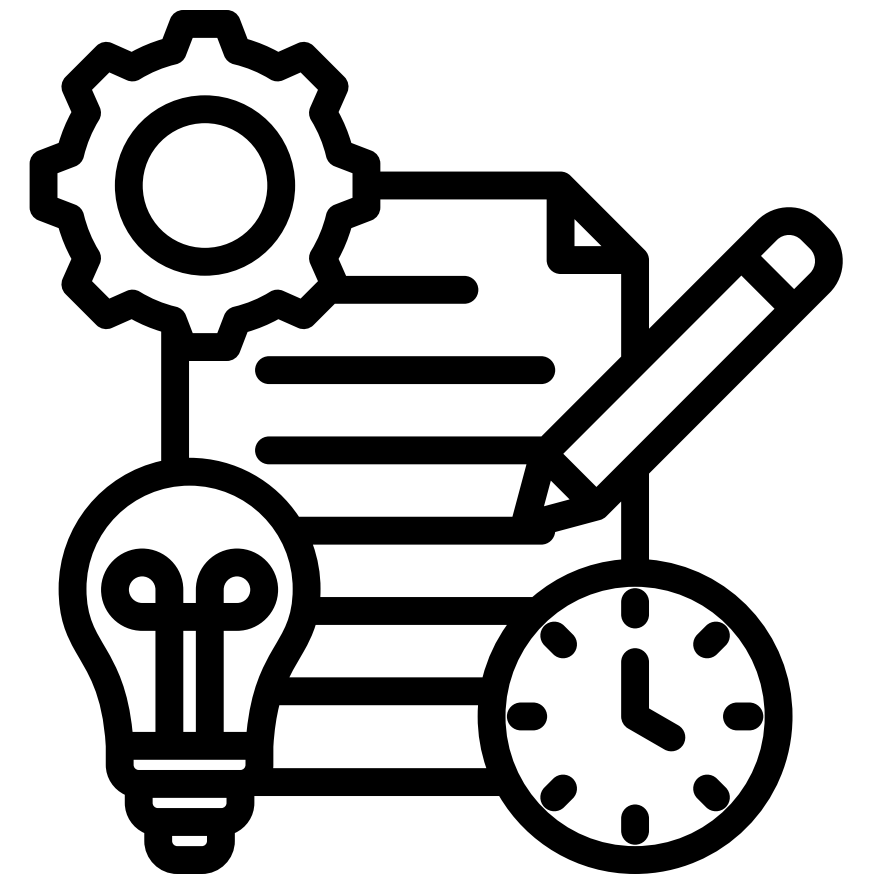
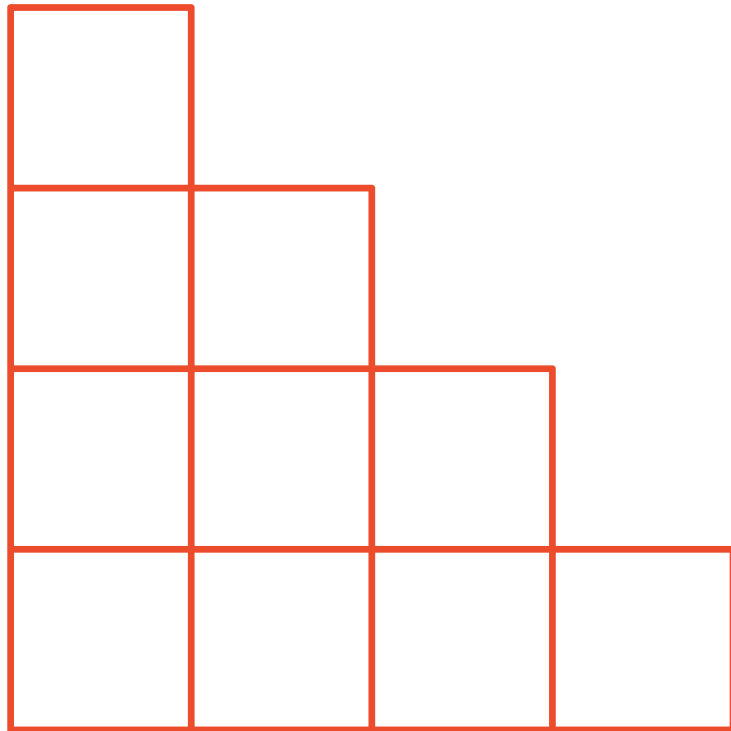
- Desarrollar una herramienta para el análisis de operaciones con tensores en proyectos de deep learning con **PyTorch**.
- Evaluar herramientas y literatura actual del problema.
- Formalizar las restricciones de dimensionalidad en las operaciones más importantes de tensores



Metodologia

08/36

- 1.Recolección de operaciones
- 2.Revisar literatura y herramientas
- 3.Diseñar una gramática formal
- 4.Evaluar la gramática con un ejemplo concreto



Recolección y Repositorios

09/36

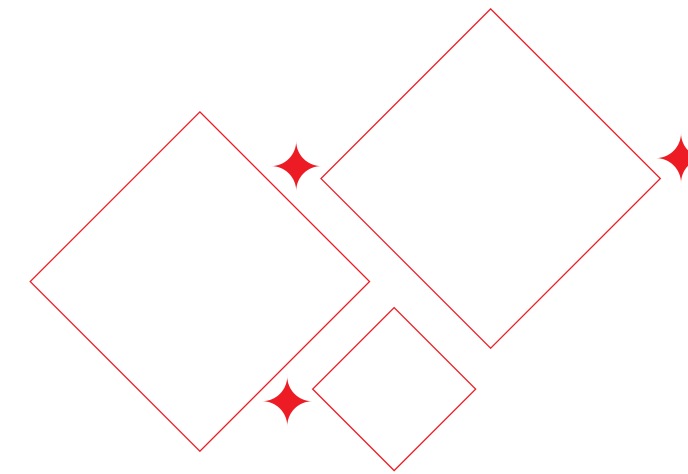
1. BERT
2. Whisper
3. LLama
4. Transformers
5. CC 6205
6. Detectron2
7. YoloV5
8. Segment anything
9. Clip
10. Blip
11. Stable diffusion



Analizar código fuente

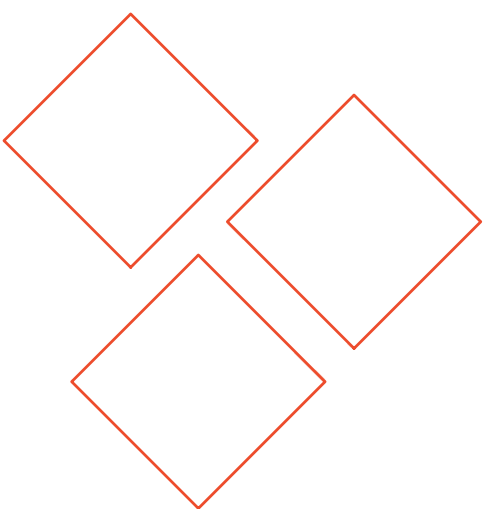
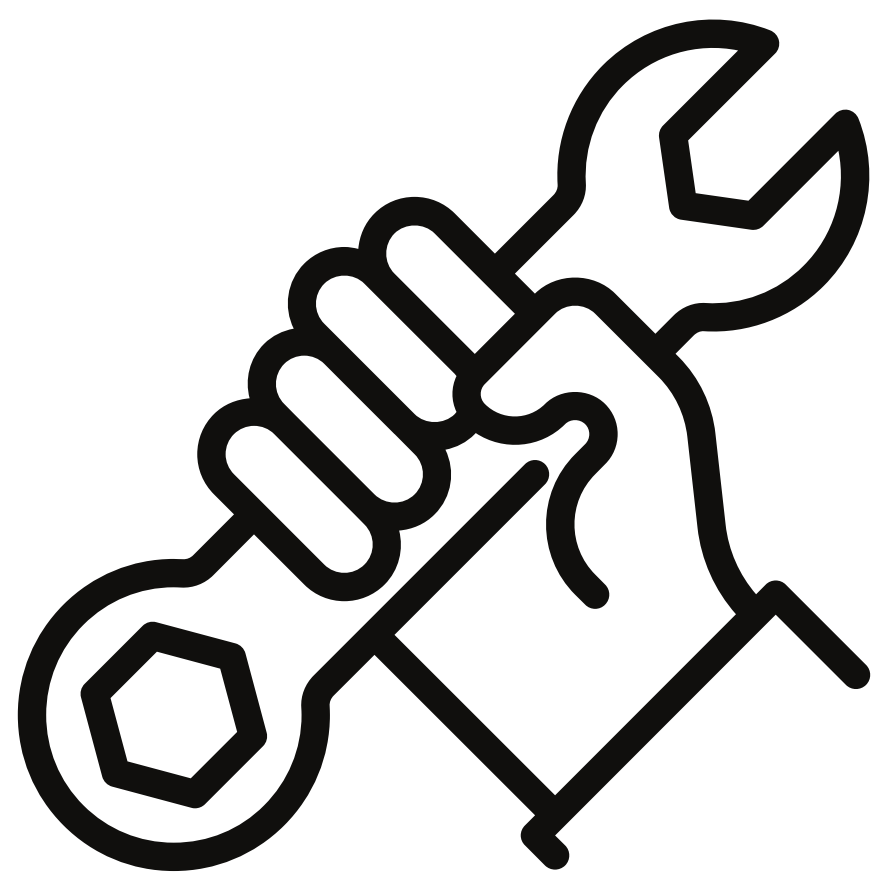


Destilar una lista de operaciones más comunes

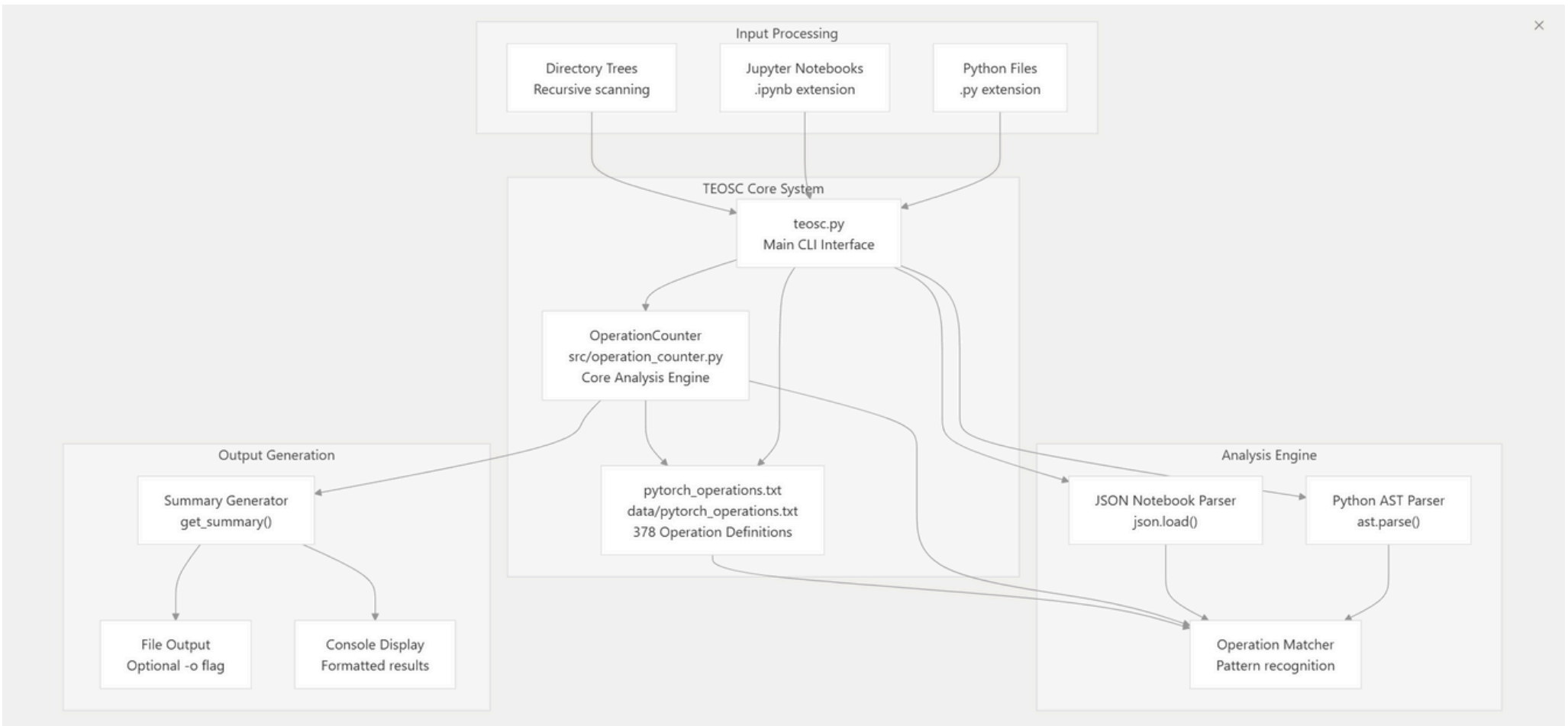


TEOSC

TEOSC, por su nombre en inglés, **Tensor Operations Static Counter**, es una herramienta que permite contar operaciones de tensores de **PyTorch** dentro de archivos.



Arquitectura



Ejecución

```
# Analyze a single Python file
python teosc.py path/to/file.py

# Analyze a Jupyter notebook
python teosc.py path/to/notebook.ipynb

# Analyze an entire directory recursively
python teosc.py path/to/directory

# Save results to a file
python teosc.py path/to/file.py -o results.txt
```

Formato de salida

```
[RESULTS] Total Operation Summary:
-----
Files scanned (3):
- /path/to/file1.py
- /path/to/file2.py
- /path/to/notebook.ipynb
-----
Operation Count Summary:
tensor: 5
add: 3
matmul: 2
...
-----
```



TEOSC

- 11 Repositorios
- 4427 Archivos .py y .ipynb
- 185 Operaciones de PyTorch

```
[RESULTS] Total Operation Summary:
-----
Files scanned (4427):
- ..\ptoc_repos\BERT\bind_pyt.py
- ..\ptoc_repos\BERT\create_pretraining_data.py
- ..\ptoc_repos\BERT\extract_features.py
- ..\ptoc_repos\BERT\file_utils.py
- ..\ptoc_repos\BERT\inference.py
- ..\ptoc_repos\BERT\modeling.py
- ..\ptoc_repos\BERT\optimization.py
- ..\ptoc_repos\BERT\run_glue.py
- ..\ptoc_repos\BERT\run_pretraining.py
- ..\ptoc_repos\BERT\run_squad.py
- ..\ptoc_repos\BERT\run_swag.py
- ..\ptoc_repos\BERT\schedulers.py
```



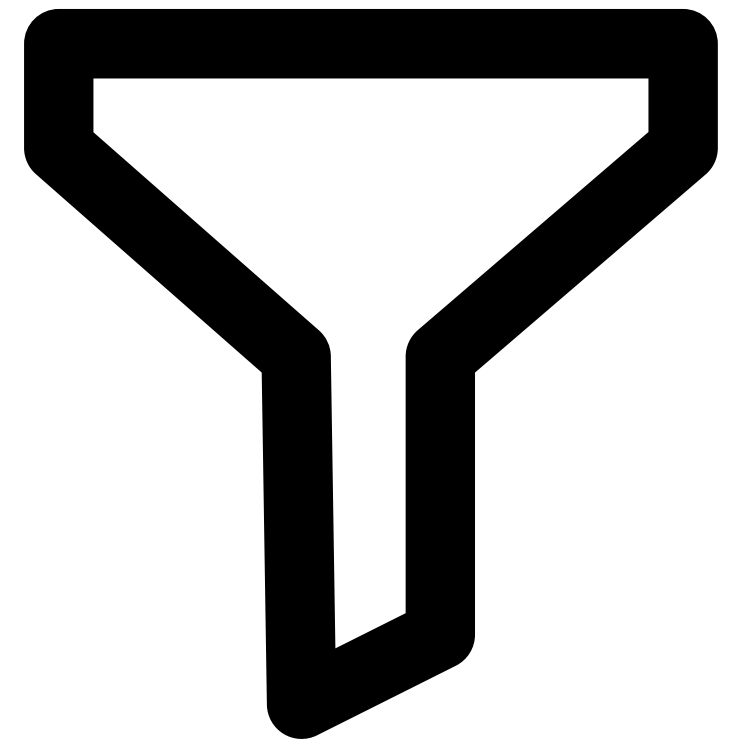
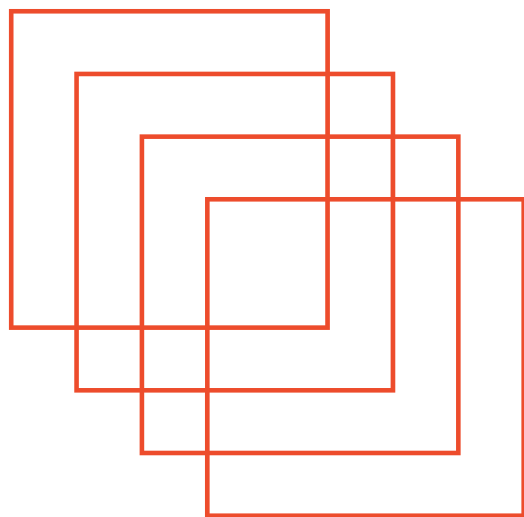
#	Operación	Frecuencia
1	range	5038
2	view	4375
3	size	4140
4	nn.Linear	3862
5	tensor	3300
6	reshape	3277
7	transpose	3101
8	cat	2357
9	sum	2146
10	split	2059
11	unsqueeze	2012
12	zeros	1907
13	arange	1807
14	ones	1643
15	max	1570
16	nn.LayerNorm	1366
17	squeeze	1306
18	expand	1288
19	any	1284
20	allclose	1224
21	nn.Dropout	1195
22	permute	1152
23	min	951
24	mean	939
25	matmul	852
26	stack	786
27	all	716
28	sqrt	682
29	flatten	661
30	nn.Embedding	660
31	where	595
32	abs	553
33	repeat	551
34	nn.Conv2d	530
35	clamp	484
36	rand	478
37	log	468
38	einsum	447
39	concat	435
40	asarray	422

Limpieza

13/36

Operaciones Descartadas:

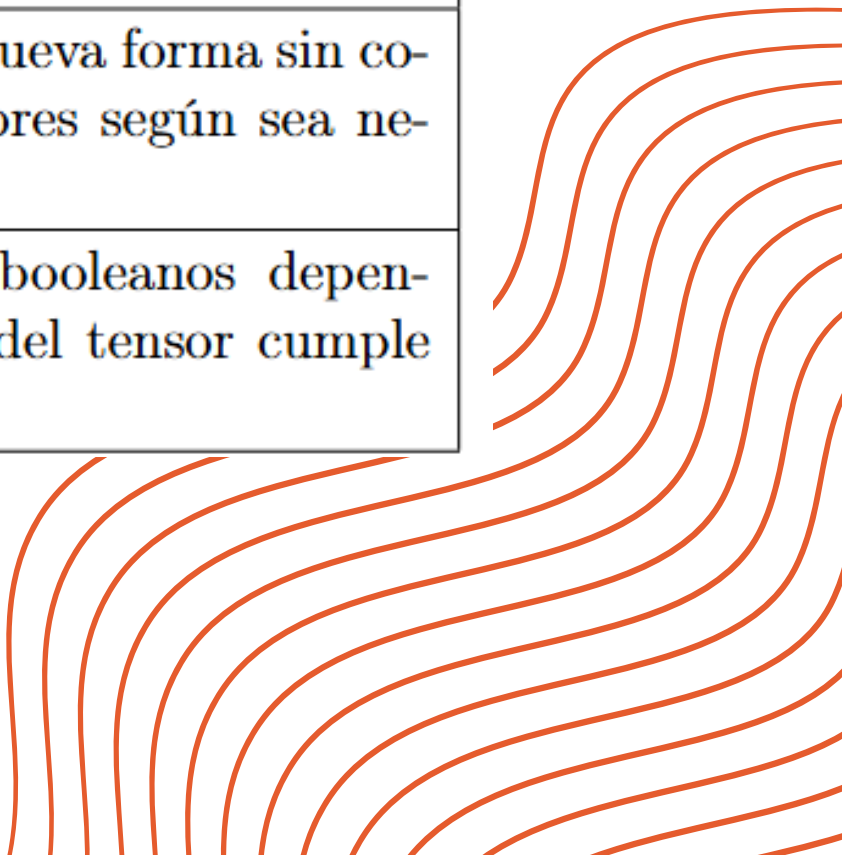
- size
- allclose
- Se eliminaron las operaciones size, allclose y todas las pertenecientes al módulo nn (asociadas a capas de redes neuronales).



Lista final de Operaciones

#	Operación	Breve descripción
1	range	Crea un tensor con una secuencia de enteros en un rango especificado.
2	view	Reinterpreta la memoria del tensor con una nueva forma, sin copiar datos. Requiere que la nueva forma sea compatible con la disposición contigua de los datos en memoria.
3	reshape	Cambia la forma de un tensor sin alterar sus datos.
4	transpose	Intercambia dos dimensiones de un tensor.
5	cat	Concatena una lista de tensores a lo largo de una dimensión específica.
6	sum	Calcula la suma de los elementos de un tensor a lo largo de dimensiones especificadas.
7	split	Divide un tensor en sub-tensores según tamaños o número de secciones.
8	unsqueeze	Añade una dimensión de tamaño uno en la posición especificada.

9	zeros	Crea un tensor lleno de ceros con la forma dada.
10	arange	Similar a range , genera un tensor con valores igualmente espaciados en un intervalo.
11	ones	Crea un tensor lleno de unos con la forma dada.
12	max	Devuelve el valor máximo y, opcionalmente, el índice a lo largo de una dimensión.
13	squeeze	Elimina dimensiones de tamaño uno de un tensor.
14	expand	Expande un tensor a una nueva forma sin copiar datos, replicando valores según sea necesario.
15	any	Devuelve un tensor con booleanos dependiendo si algún elemento del tensor cumple una condición.

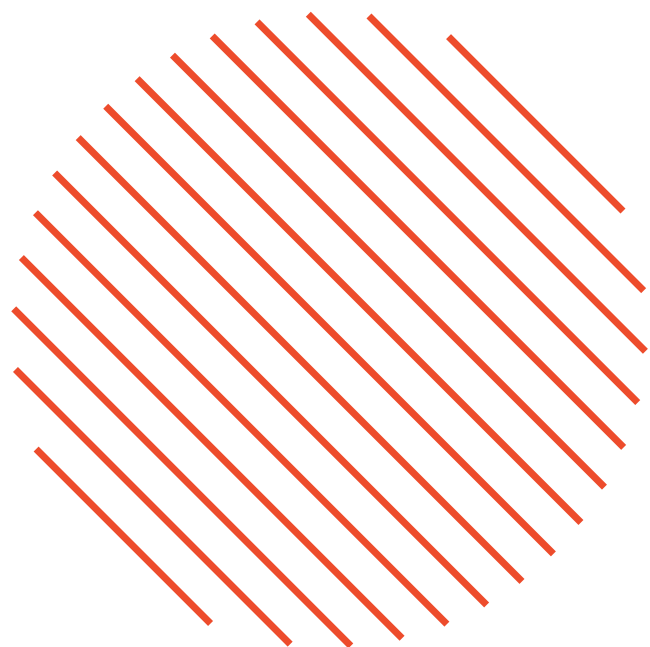


Literatura y herramientas

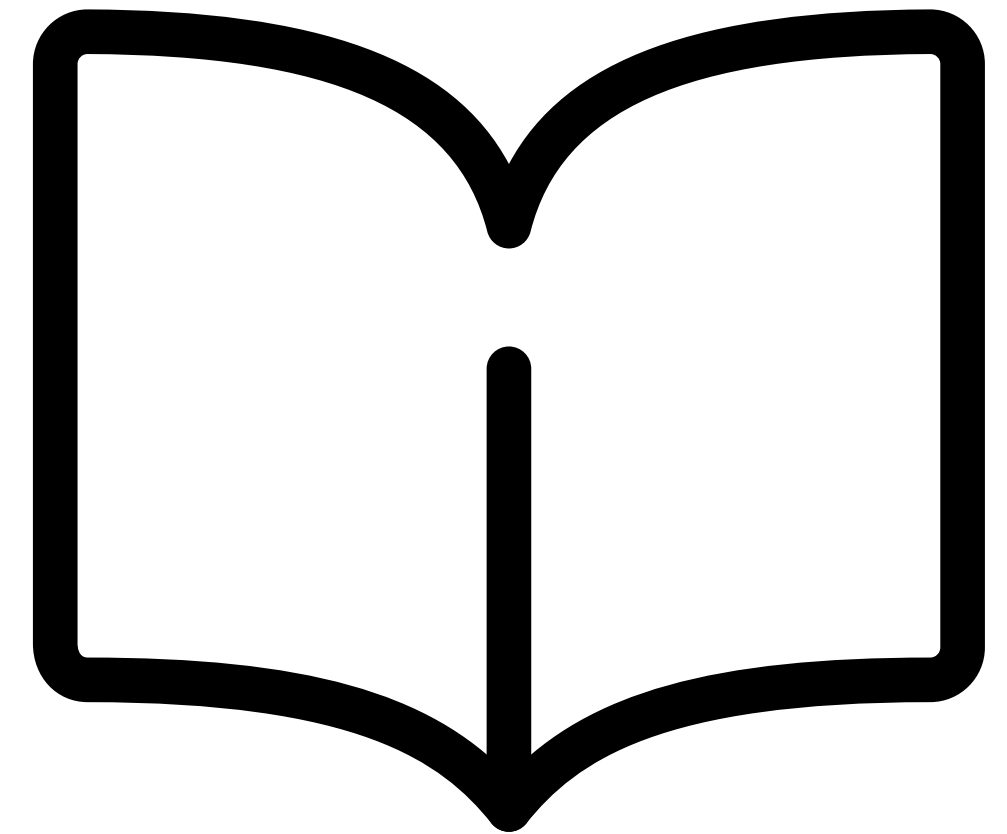
El objetivo es extraer lo más importante de estos trabajos para construir una base sólida que permita diseñar una **solución más robusta y completa**.

Cada herramienta sera evaluada en terminos de

- Capacidad de detectar errores
- Integración con ecosistémias existentes
- Uso práctico en aplicaciones



Literatura y herramientas existentes





Trabajos más relevantes

1.GraTen

Gramática expresiva, tipado gradual y chequeo híbrido

2.Pythia

Análisis basado en operaciones matemáticas , enfoque práctico y para Tensorflow

3.Ariadne

Enfoque en operaciones de batch y para Tensorflow



4.PyTea

Cobertura PyTorch, gramática expresiva, **integración con Z3**

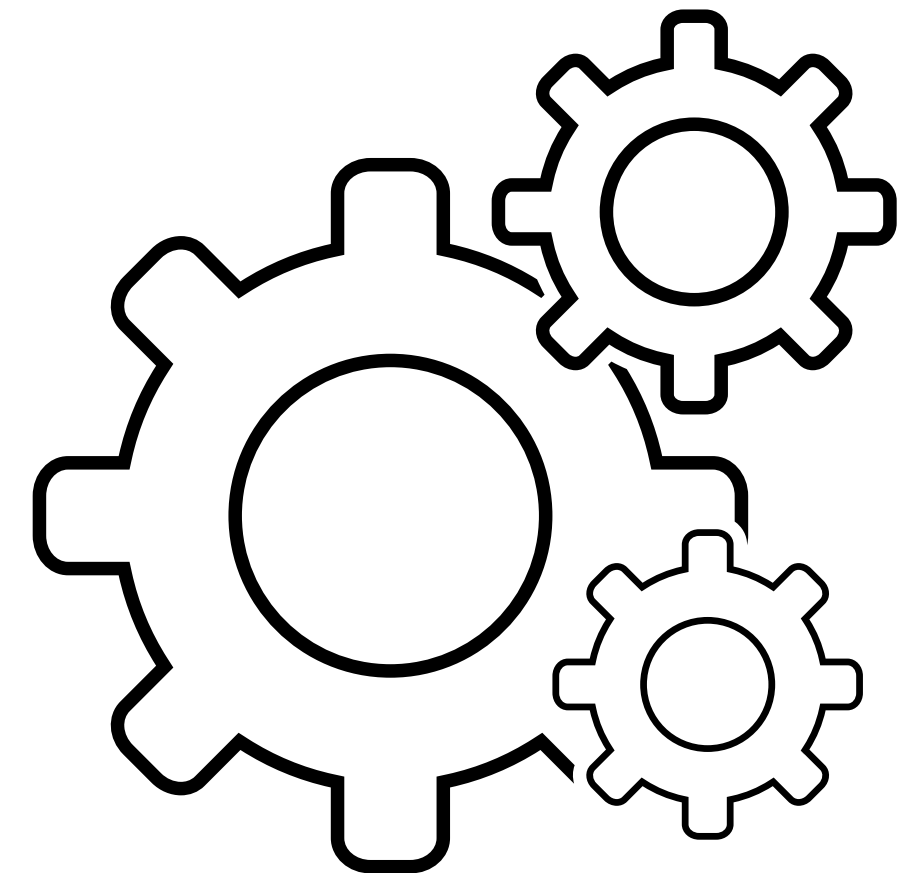
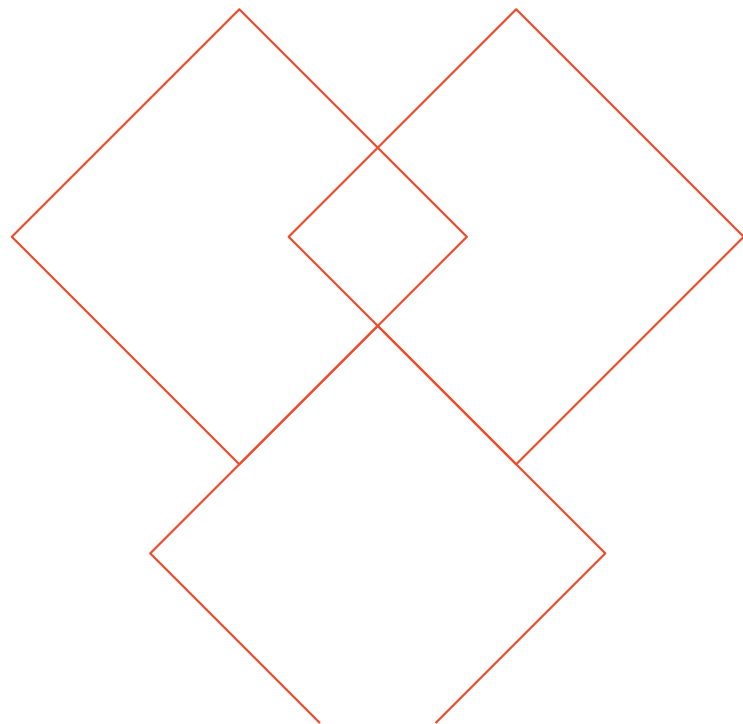
Generalizing Shape Analysis with Gradual Types

- Enfocado en **PyTorch**
- Gramática expresiva
- Desarrollado durante una pasantía en **Meta** por la autora **Zeina Migeed**
- Desarrollo industrial
- Uso de solver SMT Z3 de **Microsoft**
- Referencia a trabajos previos



Aprendizajes de la literatura y herramientas

- Necesidad de una **gramática expresiva**
- Balance entre formalización y practicidad
- Ventaja de uso de solvers SMT
- Relevancia en el contexto de **PyTorch**



Gramática: Primer acercamiento

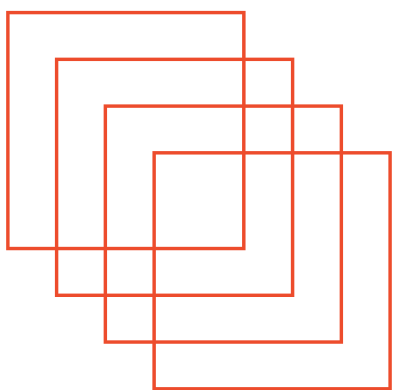
19/36

Problema:

Documentación no trae lista de errores o restricciones sino solo su uso

¿Cómo conseguir las restricciones?

1. Obtención de errores para cada operación
2. Deducir restricciones a través de errores



torch.reshape

`torch.reshape(input, shape) → Tensor`

Returns a tensor with the same data and number of elements as `input`, but with the specified shape. When possible, the returned tensor will be a view of `input`. Otherwise, it will be a copy. Contiguous inputs and inputs with compatible strides can be reshaped without copying, but you should not depend on the copying vs. viewing behavior.

See `torch.Tensor.view()` on when it is possible to return a view.

A single dimension may be -1, in which case it's inferred from the remaining dimensions and the number of elements in `input`.

Parameters

- **input** (*Tensor*) – the tensor to be reshaped
- **shape** (*tuple of int*) – the new shape

Example:

```
>>> a = torch.arange(4.)
>>> torch.reshape(a, (2, 2))
tensor([[ 0.,  1.],
        [ 2.,  3.]])
>>> b = torch.tensor([[0, 1], [2, 3]])
>>> torch.reshape(b, (-1,))
tensor([ 0,  1,  2,  3])
```

Errores

- Usar más de una dimensión con el valor -1, cuando solo una puede ser inferida automáticamente por PyTorch

```
t = torch.zeros(4)
torch.reshape(t, (-1,-1))
>>> RuntimeError: only one dimension can be inferred
```

- Especificar una dimensión con un valor negativo distinto de -1.

```
t = torch.zeros(4)
torch.reshape(t, (-2,2))
>>> RuntimeError: invalid shape dimension -2
```

=

3. Indicar una forma cuyo número total de elementos no coincide con el del tensor original.

```
t = torch.zeros(4, 2)
torch.reshape(t, (4, 3))
>>> RuntimeError: shape '[4, 3]' is invalid for input of size 8
```

Restricciones

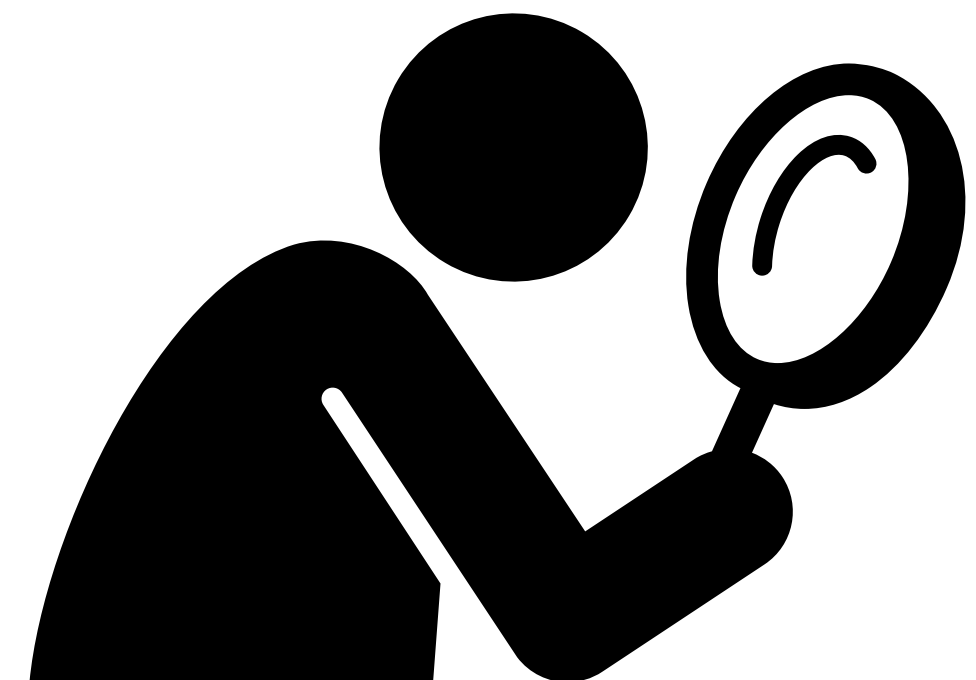
20/36

$\text{reshape}([x_1, x_2, \dots, x_n], (y_1, y_2, \dots, y_m)) = [y_1, y_2, \dots, y_m]$

$$\forall i \in [1..m] : y_i > 0 \vee y_i = -1$$

y

$$\prod_{i=1}^n x_i = \prod_{i=1}^m y_i$$



Sintaxis

22/36

$n, m \in \mathbb{Z}$

(Declaration) $decl ::= x : T$

(Type) $T ::= \mathbb{B} \mid \mathbb{Z} \mid TT([d_0, \dots, d_n]) \mid [T]$

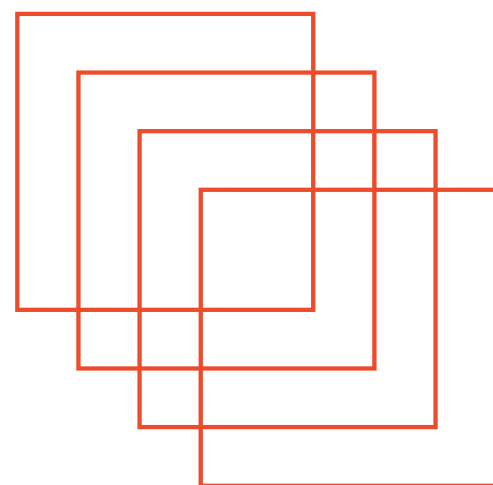
(Dimension) $d ::= n \mid x$

(Constraint) $C ::= e \text{ op}_c e \mid C \wedge C \mid C \vee C \mid \top$

(Operation Constraint) $op_c ::= = \mid \neq \mid < \mid > \mid \leq \mid \geq$

(Operation Expressions) $op_e ::= + \mid - \mid * \mid /$

(Operation Unary) $op_u ::= \mid \cdot \mid \mid \lfloor \cdot \rfloor \mid \lceil \cdot \rceil$



(Expression) $e ::= x$

$\mid n$

$\mid [e_1, \dots, e_n]$

$\mid e \text{ op}_e e$

$\mid op_u e$

$\mid \text{range}(e_1, e_2)$

$\mid \text{range}(e_1, e_2, e_3)$

$\mid \text{view}(e_1, e_2)$

$\mid \text{reshape}(e_1, e_2)$

$\mid \text{transpose}(e, m_1, m_2)$

$\mid \text{cat}(e_1, m)$

$\mid \text{sum}(e_1, m)$

$\mid \text{split}(e, m)$

$\mid \text{unsqueeze}(e_1, m)$

$\mid \text{zeros}(e)$

$\mid \text{arange}(e_1, e_2)$

$\mid \text{arange}(e_1, e_2, e_3)$

$\mid \text{ones}(e)$

$\mid \text{max}(e_1, m)$

$\mid \text{squeeze}(e_1, e_2)$

$\mid \text{expand}(e_1, e_2)$

$\mid \text{any}(e_1, m)$

(Environment) $\Gamma ::= \emptyset \mid \Gamma, x : TT([d_0, \dots, d_n]) \mid \Gamma, x : \mathbb{Z} = d$

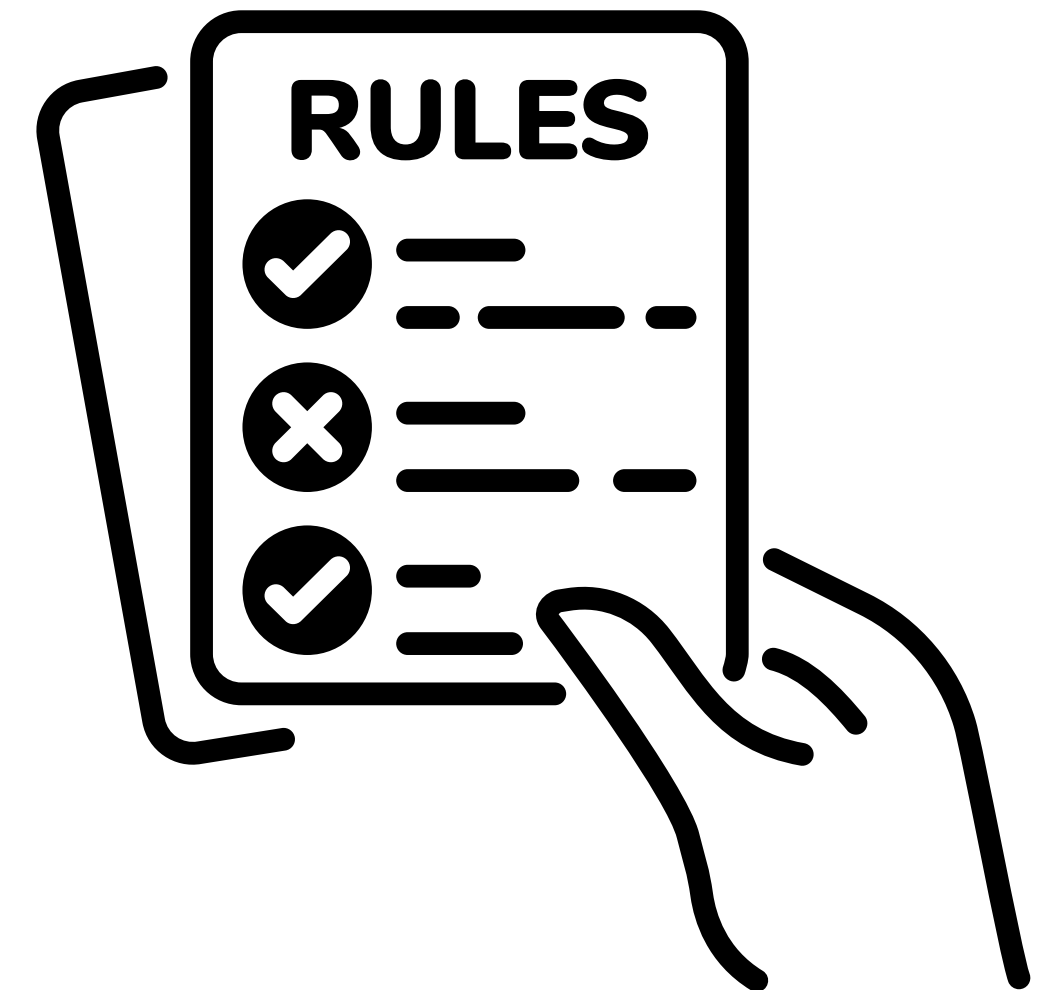
Reglas de tipo

- Garantizar coherencia estructural y semántica.
- Reglas formales para verificar **tensores**, **expresiones** y **restricciones**.
- Operaciones válidas solo si cumplen condiciones de tipado.
- Estructura: reglas básicas + reglas específicas para **PyTorch**.

`reshape([4, 2, 3], [4, 6])`



`reshape([4, 2, 3], [4, 7])`



Operaciones PyTorch: range, view y reshape

24/36

$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : \mathbb{Z}; C_1 \quad \Gamma \vdash_1 e_2 : \mathbb{Z}; C_2 \quad \Gamma \vdash_1 e_3 : \mathbb{Z}; C_3 \\
 x \text{ free} \\
 C_4 = e_3 \neq 0 \\
 C_5 = |e_2 - e_1| * e_3 > 0 \\
 C_6 = x = \lfloor \frac{e_2 - e_1}{e_3} \rfloor + 1 \\
 C = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 \\
 \text{(OP-RANGE-3)} \frac{}{\Gamma \vdash_3 \text{range}(e_1, e_2, e_3) : TT([x]); C}
 \end{array}$$

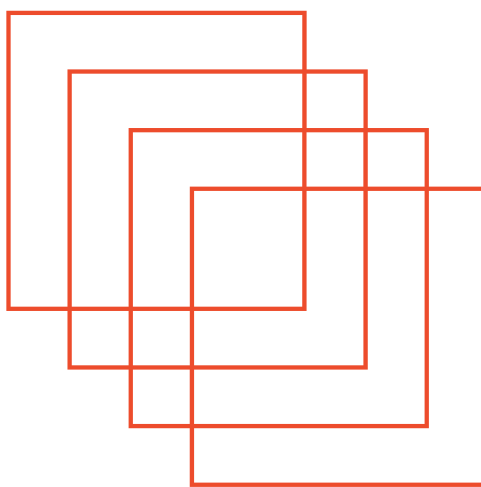
$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \\
 \Gamma \vdash_1 e_2 : TT([d'_0, \dots, d'_m]); C_2 \\
 C_3 = \prod_{i=1}^n d_i = \prod_{i=1}^m d'_i \\
 C_4 = \bigwedge_{i=1}^m (d'_i > 0) \\
 C_5 = \bigwedge_{i=1}^n (d_i > 0) \vee (d_i = -1) \\
 C = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 \\
 \text{(OP-VIEW)} \frac{}{\Gamma \vdash_3 \text{view}(e_1, e_2) : TT([d'_0, \dots, d'_n]); C}
 \end{array}$$

$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \quad \Gamma \vdash_1 e_2 : TT([d'_0, \dots, d'_n]); C_2 \\
 C_3 = \prod_{i=1}^n d_i = \prod_{i=1}^m d'_i \\
 C_4 = \bigwedge_{i=1}^m (d'_i > 0) \\
 C_5 = \bigwedge_{i=1}^n (d_i > 0) \vee (d_i = -1) \\
 C = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \\
 \text{(OP-RESHAPE)} \frac{}{\Gamma \vdash_3 \text{reshape}(e_1, e_2) : TT([d'_0, \dots, d'_n]); C}
 \end{array}$$

Operaciones PyTorch: transpose, cat y sum

25/36

$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \\
 \Gamma \vdash_1 m_1 : \mathbb{Z}; \top \quad \Gamma \vdash_1 m_2 : \mathbb{Z}; \top \\
 x \text{ free} \\
 x : TT([d'_0, \dots, d'_n]) \\
 C_2 = -n \leq m_1 \wedge m_1 < n - 1 \\
 C_3 = -n \leq m_2 \wedge m_2 < n - 1 \\
 C_4 = (d'_{m_1} = d_{m_2}) \wedge (d'_{m_2} = d_{m_1}) \\
 C = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \\
 \text{(OP-TRANSPOSE)} \frac{}{\Gamma \vdash_3 \text{tranpose}(e_1, m_1, m_2) : TT([d'_0, \dots, d'_n]); C}
 \end{array}$$



$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : [TT(d_{0,1}, \dots, d_{n,1}), \dots, TT(d_{0,l}, \dots, d_{n,l})]; C_1 \\
 \Gamma \vdash_1 m : \mathbb{Z}; \top \\
 x \text{ free} \\
 C_2 = \bigwedge_{j=1}^l \bigwedge_{\substack{i=0 \\ i \neq m}}^n d_{i,1} = d_{i,j} \\
 C_3 = x = \sum_{j=1}^l d_{m,j} \\
 C = C_1 \wedge C_2 \wedge C_3 \\
 \text{(OP-CAT)} \frac{}{\Gamma \vdash_3 \text{cat}(e_1, m) : TT([d_0, \dots, d_{m-1}, x, d_{m+1}, \dots, d_n]); C}
 \end{array}$$

$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \\
 \Gamma \vdash_1 m : \mathbb{Z}; \top \\
 C_2 = -n \leq m \wedge m < n - 1 \\
 C = C_1 \wedge C_2 \\
 \text{(OP-SUM)} \frac{}{\Gamma \vdash_3 \text{sum}(e_1, m) : TT([d_0, \dots, d_{m-1}, d_{m+1}, \dots, d_n]); C}
 \end{array}$$

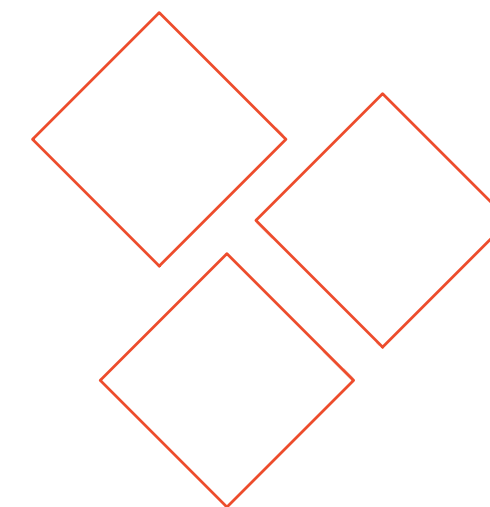
Operaciones PyTorch: split, unsqueeze y zeros

26/36

$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \\
 \Gamma \vdash_1 m : \mathbb{Z}; \top \\
 C_2 = m > 0 \\
 C = C_1 \wedge C_2 \\
 \hline
 \text{(OP-SPLIT)} \quad \Gamma \vdash_3 \text{split}(e_1, m) : [TT(d_{0,1}, \dots, d_{n,1}), \dots, TT(d_{0,m}, \dots, d_{n,m})]; C
 \end{array}$$

$$\begin{array}{c}
 \Gamma \vdash_1 e : \mathbb{Z}; C_1 \\
 x \text{ free} \\
 C_2 = 0 < e \\
 C_3 = x = e \\
 C = C_1 \wedge C_2 \wedge C_3 \\
 \hline
 \text{(OP-ZEROS-INT)} \quad \Gamma \vdash_3 \text{zeros}(e) : TT([x]); C
 \end{array}$$

$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \\
 \Gamma \vdash_1 m : \mathbb{Z}; \top \\
 C_2 = -n - 1 \leq m \wedge m < n + 1 \\
 C = C_1 \wedge C_2 \\
 \hline
 \text{(OP-UNSQUEEZE)} \quad \Gamma \vdash_3 \text{unsqueeze}(e_1, m) : TT([d_0, \dots, d_{m-1}, 1, d_{m+1}, \dots, d_n]); C
 \end{array}$$

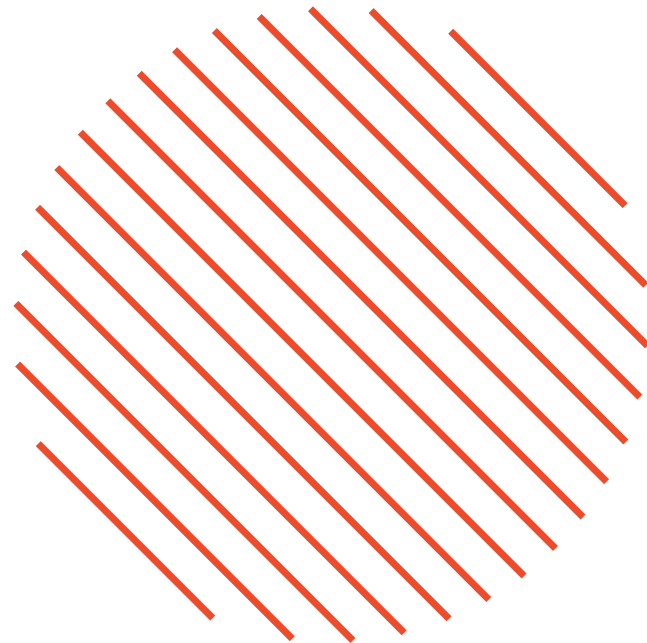


Operaciones PyTorch: arange, ones y max

27/36

$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : \mathbb{Z}; C_1 \quad \Gamma \vdash_1 e_2 : \mathbb{Z}; C_2 \quad \Gamma \vdash_1 e_3 : \mathbb{Z}; C_3 \\
 x \text{ free} \\
 C_4 = e_3 \neq 0 \\
 C_5 = |e_2 - e_1| * e_3 > 0 \\
 C_6 = x = \lceil \frac{e_2 - e_1}{e_3} \rceil \\
 C = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 \\
 \text{(OP-ARANGE-3)} \frac{}{\Gamma \vdash_3 \text{arange}(e_1, e_2, e_3) : TT([x]); C}
 \end{array}$$

$$\begin{array}{c}
 \Gamma \vdash_1 e : \mathbb{Z}; C_1 \\
 x \text{ free} \\
 C_2 = 0 < e \\
 C_3 = x = e \\
 C = C_1 \wedge C_2 \wedge C_3 \\
 \text{(OP-ONES-INT)} \frac{}{\Gamma \vdash_3 \text{ones}(e) : TT([x]); C}
 \end{array}$$



$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \\
 \Gamma \vdash_1 m : \mathbb{Z}; \top \\
 C_3 = -n \leq m \wedge m < n - 1 \\
 C = C_1 \wedge C_2 \wedge C_3 \\
 \text{(OP-MAX)} \frac{}{\Gamma \vdash_3 \text{max}(e_1, e_2) : TT([d_0, \dots, d_{m-1}, d_{m+1}, \dots, d_n]); C}
 \end{array}$$

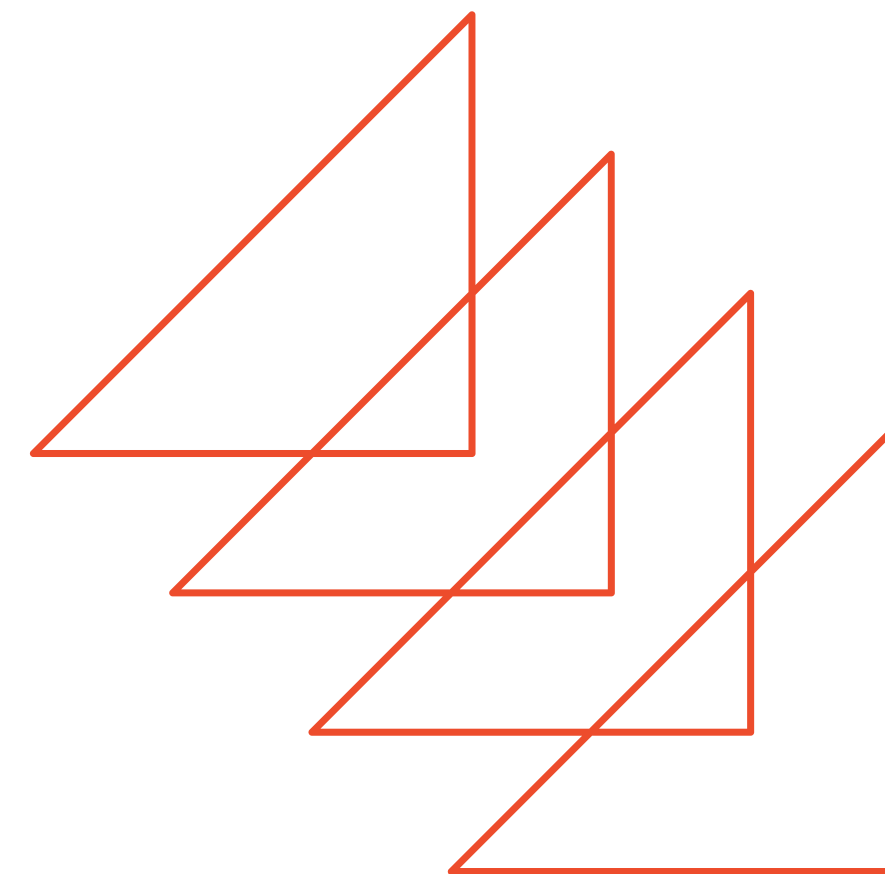
Operaciones PyTorch: arange, ones y max

28/36

$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \\
 \Gamma \vdash_1 e_2 : \mathbb{Z}; C_2 \\
 x \text{ free} \\
 x : TT([d'_0, \dots, d'_n]) \\
 C_2 = -n \leq e_2 \wedge e_2 < n - 1 \\
 C_3 = \bigwedge_{i=0}^{n'} (d'_i \neq 1) \\
 C_4 = n' \leq n \\
 C = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \\
 \text{(OP-SQUEEZE-INT)} \frac{}{\Gamma \vdash_3 \text{squeeze}(e_1, e_2) : TT([d'_0, \dots, d'_n]); C}
 \end{array}$$

$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \quad \Gamma \vdash_1 e_2 : TT([d'_0, \dots, d'_n]); C_2 \\
 C_3 = n \leq n' \\
 C = C_1 \wedge C_2 \wedge C_3 \\
 \text{(OP-EXPAND)} \frac{}{\Gamma \vdash_3 \text{expand}(e_1, e_2) : TT([d'_0, \dots, d'_n]); C}
 \end{array}$$

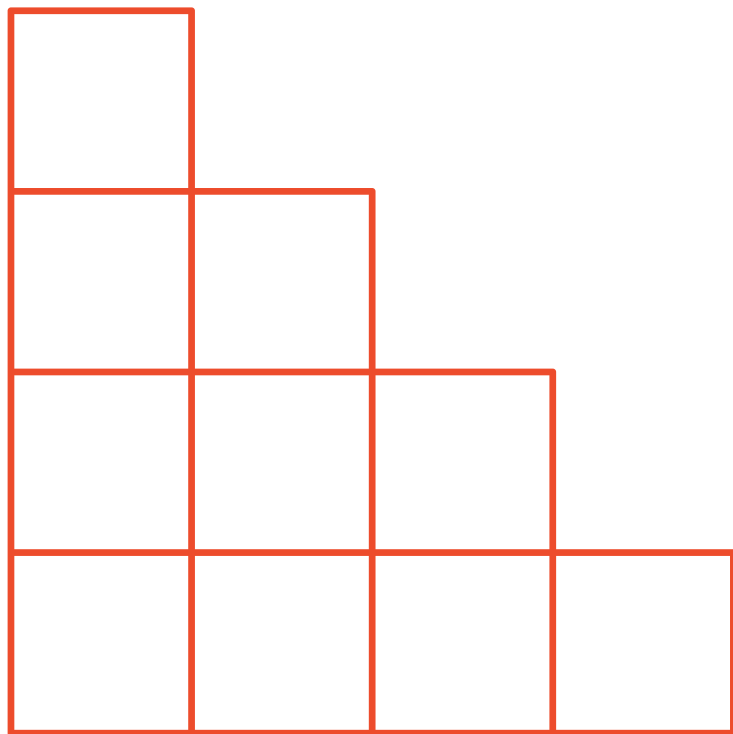
$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \\
 \Gamma \vdash_1 m : \mathbb{Z}; \top \\
 C_3 = -n \leq m \wedge m < n - 1 \\
 C = C_1 \wedge C_2 \wedge C_3 \\
 \text{(OP-ANY)} \frac{}{\Gamma \vdash_3 \text{any}(e_1, e_2) : TT([d_0, \dots, d_{m-1}, d_{m+1}, \dots, d_n]); C}
 \end{array}$$



Limitaciones de la Gramática

29/36

- Dependencia de valores numéricos concretos.
- Restricciones en la representación de firmas complejas.
- Empleo de variables libres como recurso auxiliar.



Evaluación

Restricciones y estructuras definidas en la gramática pueden trasladarse de manera directa a una

herramienta práctica

- Python
- SMT Solver Z3
- <https://z3prover.github.io>

30/36



1er Ejemplo Unsqueeze

$$\begin{array}{c} \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \\ \Gamma \vdash_1 m : \mathbb{Z}; \top \\ C_2 = -n - 1 \leq m \wedge m < n + 1 \\ C = C_1 \wedge C_2 \\ \text{(OP-UNSQUEEZE)} \frac{}{\Gamma \vdash_3 \text{unsqueeze}(e_1, m) : TT([d_0, \dots, d_{m-1}, 1, d_{m+1}, \dots, d_n]); C} \end{array}$$

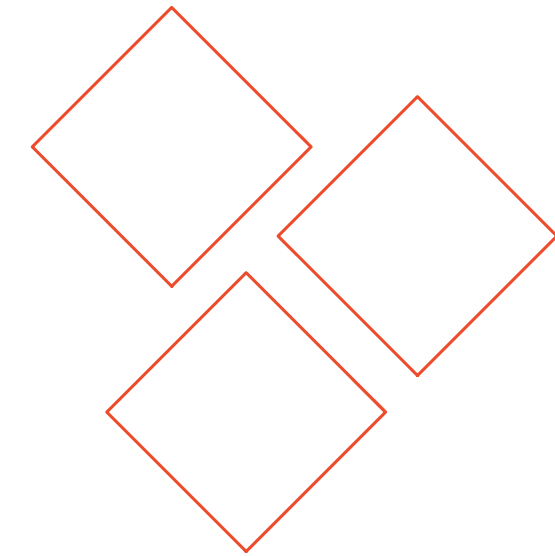
2do Ejemplo Reshape

$$\begin{array}{c} \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \quad \Gamma \vdash_1 e_2 : TT([d'_0, \dots, d'_n]); C_2 \\ C_3 = \prod_{i=1}^n d_i = \prod_{i=1}^m d'_i \\ C_4 = \bigwedge_{i=1}^m (d'_i > 0) \\ C_5 = \bigwedge_{i=1}^n (d_i > 0) \vee (d_i = -1) \\ C = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \\ \text{(OP-RESHAPE)} \frac{}{\Gamma \vdash_3 \text{reshape}(e_1, e_2) : TT([d'_0, \dots, d'_n]); C} \end{array}$$

Primer Ejemplo Unsqueeze: código

Para verificar la satisfacibilidad de estas operaciones se implementó una solución utilizando un solver y variables simbólicas.

$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \\
 \Gamma \vdash_1 m : \mathbb{Z}; \top \\
 C_2 = -n - 1 \leq m \wedge m < n + 1 \\
 C = C_1 \wedge C_2 \\
 \hline
 \text{(OP-UNSQUEEZE)} \quad \Gamma \vdash_3 \text{unsqueeze}(e_1, m) : TT([d_0, \dots, d_{m-1}, 1, d_{m+1}, \dots, d_n]); C
 \end{array}$$



1. Crear solver

```
def is_unsqueeze_valid(e1, m: int) -> bool:
    # Crear solver Z3
    s = Solver()

    # Variables simbólicas para las dimensiones
    n = len(e1)
    e1_dims = [Int(f"d_{i}") for i in range(n)]
```

2. Agregar restricciones

```
# Restriccion C2
constraint_c2_leq = -n <= m
constraint_c2_gt: bool = m < (n + 1)
constraint_c2 = And(constraint_c2_leq, constraint_c2_gt)
s.add(constraint_c2)
```

3. Ver si es satisfacible

```
return s.check() == sat
```

Primer Ejemplo Unsqueeze: resultados

1. Caso Valido

```
Ejemplo 1: unsqueeze([4, 2, 3], 3)
C1: True
C2: -3 <= 2: True
C2: 2 < 4: True
VALID operation.
```

2. Operacion invalida: C2 no se cumple

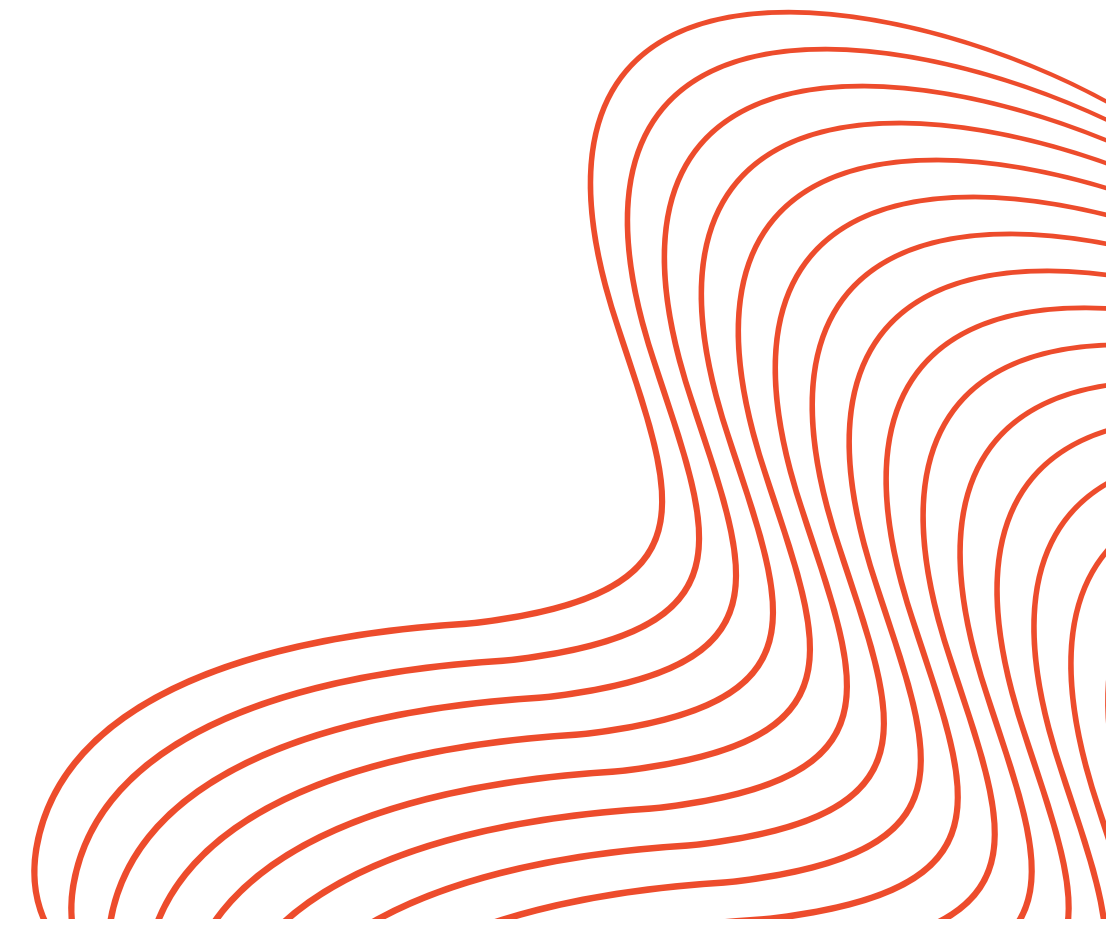
```
Ejemplo 2: unsqueeze([1, 2, 3], -200)
C1: True
C2: -3 <= -200: False
C2: -200 < 4: True
INVALID operation.
```

3. Operacion invalida: C3 no se cumple

```
Ejemplo 3: unsqueeze([1, 2, 3], 10)
C1: True
C2: -3 <= 10: True
C2: 10 < 4: False
INVALID operation.
```

$$\begin{array}{c} \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \\ \Gamma \vdash_1 m : \mathbb{Z}; \top \\ C_2 = -n - 1 \leq m \wedge m < n + 1 \\ C = C_1 \wedge C_2 \end{array}$$

$$\Gamma \vdash_3 \text{unsqueeze}(e_1, m) : TT([d_0, \dots, d_{m-1}, 1, d_{m+1}, \dots, d_n]); C$$



Segundo ejemplo reshape: código

33/36

$$\begin{array}{c} \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \quad \Gamma \vdash_1 e_2 : TT([d'_0, \dots, d'_n]); C_2 \\ C_3 = \prod_{i=1}^n d_i = \prod_{i=1}^m d'_i \\ C_4 = \bigwedge_{i=1}^m (d'_i > 0) \\ C_5 = \bigwedge_{i=1}^n (d_i > 0) \vee (d_i = -1) \\ C = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \\ \hline \text{(OP-RESHAPE)} \quad \Gamma \vdash_3 \text{reshape}(e_1, e_2) : TT([d'_0, \dots, d'_n]); C \end{array}$$

1. Crear solver

```
def is_reshape_valid(e1, e2) -> bool:
    # Crear solver Z3
    s = Solver()

    # Variables simbolicas para las dimensiones
    e1_dims = [Int(f"d_{i}") for i in range(len(e1))]
    e2_dims = [Int(f"d'_{i}") for i in range(len(e2))]
```

2. Agregar restricciones

```
# Restricciones C3: Producto de dimensiones debe ser igual
product_e1 = 1
for i, dim in enumerate(e1):
    product_e1 *= dim

product_e2 = 1
for i, dim in enumerate(e2):
    product_e2 *= dim
# Verifica que el producto de las dimensiones del tensor de entrada sea igual al
# producto de las dimensiones objetivo
C_3 = product_e1 == product_e2
s.add(C3)
```

```
# Restricciones C4: Las dimensiones de salida tienen que ser todas positivas
constraints_c4 = []
for i, dim in enumerate(e2):
    constraint = e2_dims[i] > 0
    constraints_c4.append(constraint)
s.add(constraint)
```

```
# Restriccion C5: Las dimensiones de entradas pueden ser mayor a 0 o -1
constraints_c5 = []
for i, dim in enumerate(e1):
    # Usar Or de Z3: e1_dims[i] > 0 OR e1_dims[i] == -1
    constraint = Or(e1_dims[i] > 0, e1_dims[i] == -1)
    constraints_c5.append(constraint)
s.add(constraint)
```

3. Ver si es satisfacible

```
return s.check() == sat
```


Segundo ejemplo reshape: resultados

1. Caso Valido

```
Ejemplo 1: reshape([4, 2, 3], [4, 6])
C1: True
C2: True
C3: 24 == 24
C4: True ^ True
C5: Or(True, False) ^ Or(True, False) ^ Or(True, False)
VALID operation.
```

$$\begin{array}{c}
 \Gamma \vdash_1 e_1 : TT([d_0, \dots, d_n]); C_1 \quad \Gamma \vdash_1 e_2 : TT([d'_0, \dots, d'_n]); C_2 \\
 C_3 = \prod_{i=1}^n d_i = \prod_{i=1}^m d'_i \\
 C_4 = \bigwedge_{i=1}^m (d'_i > 0) \\
 C_5 = \bigwedge_{i=1}^n (d_i > 0) \vee (d_i = -1) \\
 C = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \\
 \hline
 \Gamma \vdash_3 \text{reshape}(e_1, e_2) : TT([d'_0, \dots, d'_n]); C
 \end{array}$$

2. Operación invalida: C3 no se cumple

```
Ejemplo 2: reshape([4, 2, 3], [4, 7])
C1: True
C2: True
C3: 24 == 28
C4: True ^ True
C5: Or(True, False) ^ Or(True, False) ^ Or(True, False)
INVALID operation.
```

3. Operacion invalida: C4 no se cumple

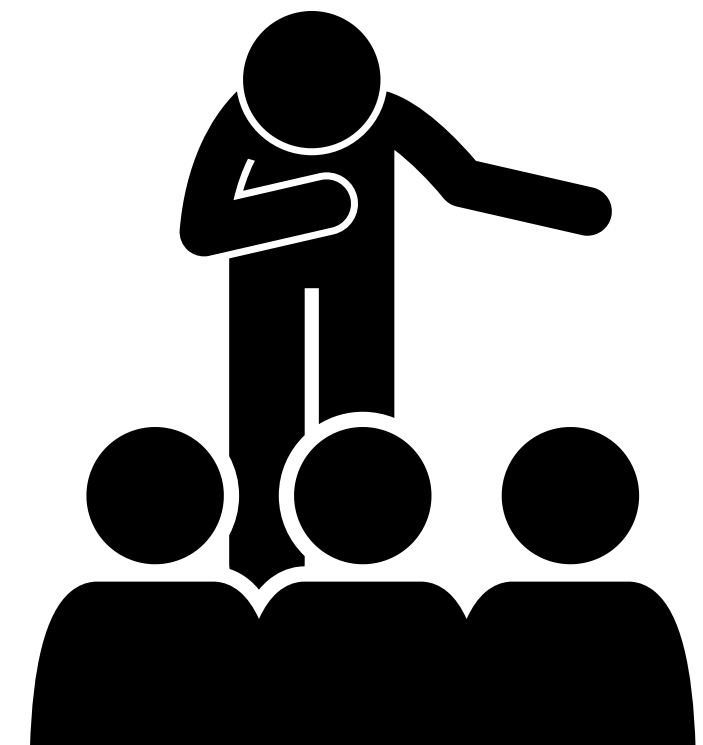
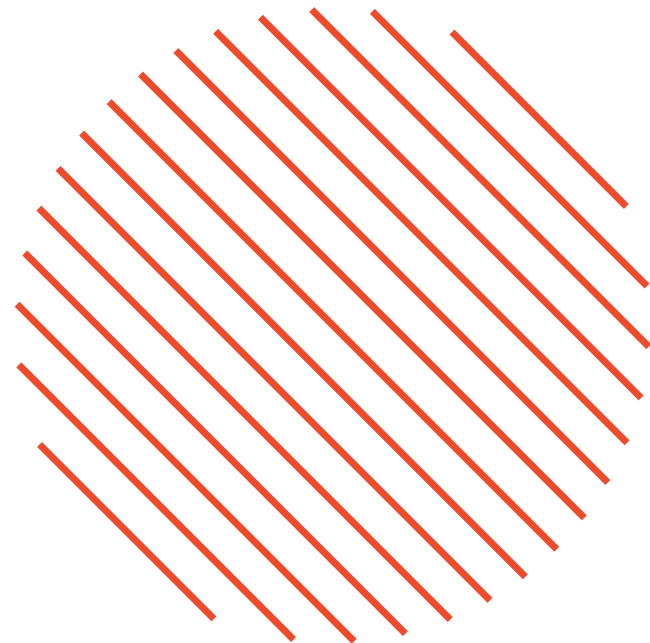
```
Ejemplo 3: reshape([6, 4], [-3, -8])
C1: True
C2: True
C3: 24 == 24
C4: False ^ False
C5: Or(True, False) ^ Or(True, False)
INVALID operation.
```



Conclusión

Este trabajo constituye un **primer avance hacia la detección temprana de errores relacionados con la compatibilidad de dimensiones en operaciones tensoriales** dentro de programas basados en **PyTorch**

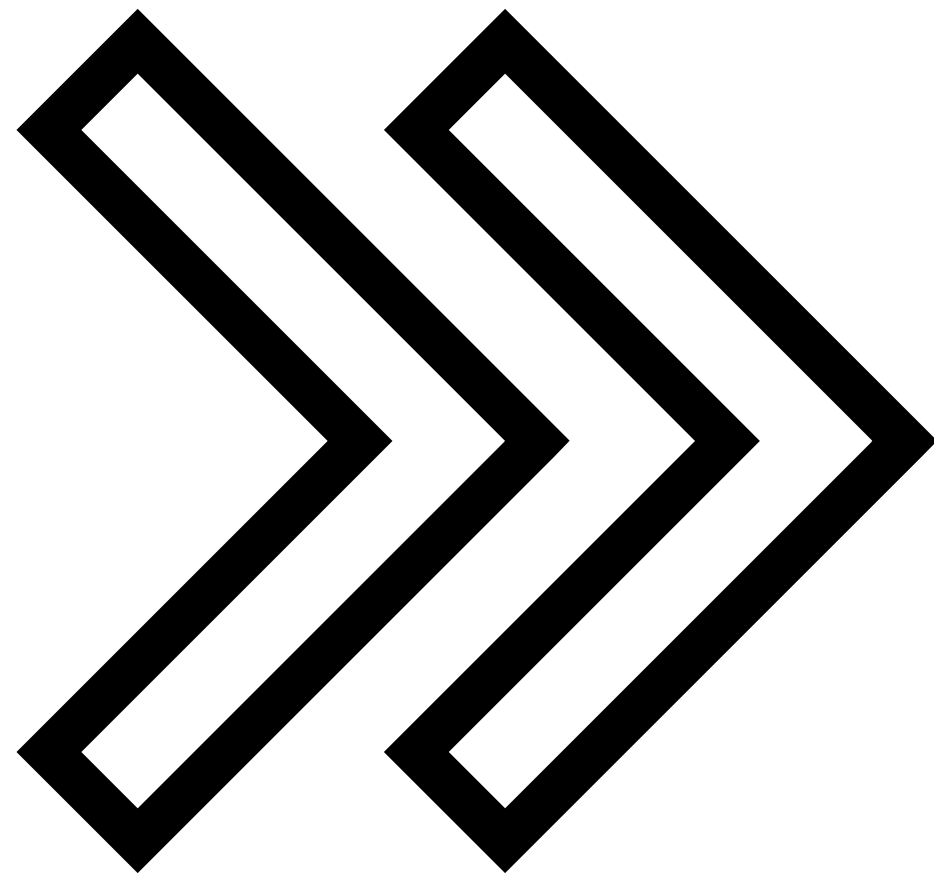
Esta gramática permite formalizar y estructurar el trabajo realizado, facilitando su lectura y posterior implementación.

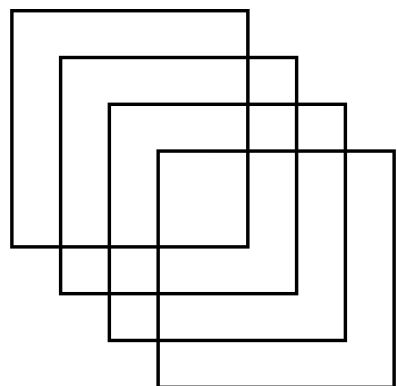


Trabajo a futuro

36/36

- Gradual Typing
- Ampliar cobertura de operaciones
- Agregar soporte a operaciones firmas más complejas
- Implementar en un ambiente práctico





Formalización enfocada a operaciones de Tensores en PyTorch

FECHA

2025-10-07

ESTUDIANTE

Jorge Cruces

PROFESORES GUÍAS

Matías Toro
Éric Tanter

COMISIÓN

Luis Mateu
Valentin Barriere

