

# Code Transformation for Distributed Python ML Code

## 1 Python Abstract Syntax

<i>module</i>	::=	<i>stmt</i> * <i>type_ignore</i>	(MODULEDEF)
<i>stmt</i>	::=	(@ <i>expr</i> )* <b>def</b> <i>id</i> ( <i>args</i> ) (-> <i>expr</i> )? : (# <b>type:s</b> )? <i>stmt</i> *	(FUNDEF)
		(@ <i>expr</i> )* <b>async def</b> <i>id</i> ( <i>args</i> ) (-> <i>expr</i> )? : (# <b>type:s</b> )? <i>stmt</i> *	(ASYNCFUNDEF)
		(@ <i>expr</i> )* <b>class</b> <i>id</i> ( <i>expr</i> * <i>keyword</i> *) : <i>stmt</i> *	(CLASSDEF)
		<b>return</b> <i>expr</i> ?	(RETURN)
		<b>delete</b> <i>expr</i> *	(DELETE)
		<i>expr</i> * = <i>expr</i> (# <b>type:s</b> )?	(ASSIGN)
		<i>expr</i> <i>binop</i> = <i>expr</i>	(AUGASSIGN)
		<i>expr</i> : <i>expr</i> (= <i>expr</i> )?	(ANNASSIGN)
		(# <b>type:s</b> )? <b>for</b> <i>expr</i> <b>in</b> <i>expr</i> : <i>stmt</i> * ( <b>else</b> : <i>stmt</i> )?	(FORLOOP)
		(# <b>type:s</b> )? <b>async for</b> <i>expr</i> <b>in</b> <i>expr</i> : <i>stmt</i> * ( <b>else</b> : <i>stmt</i> )?	(ASYNCFORLOOP)
		<b>while</b> ( <i>expr</i> ) : <i>stmt</i> * ( <b>else</b> : <i>stmt</i> )?	(WHILELOOP)
		<b>if</b> ( <i>expr</i> ) : <i>stmt</i> * ( <b>else</b> : <i>stmt</i> )?	(IF)
		(# <b>type:s</b> )? <b>with</b> <i>with_item</i> * : <i>stmt</i> *	(WITH)
		(# <b>type:s</b> )? <b>async with</b> <i>with_item</i> * : <i>stmt</i> *	(ASYNCWITH)
		<b>match</b> <i>expr</i> : <i>match_case</i> *	(MATCH)
		<b>raise</b> <i>expr</i> ? ( <b>from</b> <i>expr</i> )?	(RAISE)
		<b>try</b> : <i>stmt</i> * <i>exc_handler</i> * ( <b>else</b> : <i>stmt</i> )? ( <b>finally</b> : <i>stmt</i> )?	(TRY)
		<b>assert</b> <i>expr</i> <i>expr</i> ?	(ASSERT)
		<b>import</b> <i>alias</i> *	(IMPORT)
		<b>from</b> <i>i id</i> ? <b>import</b> <i>alias</i> *	(IMPORTFROM)
		<b>global</b> <i>id</i> *	(GLOBAL)
		<b>nonlocal</b> <i>id</i> *	(NONLOCAL)
		<i>expr</i>	(EXPRSTMT)
		<b>pass</b>	(PASS)
		<b>break</b>	(BREAK)
		<b>continue</b>	(CONTINUE)

<i>expr</i>	<code>::=</code>	<code>expr boolop expr</code>	(BOOLOP)
		<code>expr := expr</code>	(NAMEDEXPR)
		<code>expr binop expr</code>	(BINARYOP)
		<code>unop expr</code>	(UNARYOP)
		<code>lambda args : expr</code>	(LAMBDA)
		<code>expr if expr else expr</code>	(IFEXPR)
		<code>{(expr : expr)* (**expr)*}</code>	(DICTIONARY)
		<code>{expr*}</code>	(SET)
		<code>[expr*]</code>	(LIST)
		<code>(expr*)</code>	(TUPLE)
		<code>{expr:expr comprehension*}</code>	(DICTIONARYCOMP)
		<code>{expr comprehension*}</code>	(SETCOMP)
		<code>[expr comprehension*]</code>	(LISTCOMP)
		<code>(expr comprehension*)</code>	(GENERATORCOMP)
		<code>await expr</code>	(AWAIT)
		<code>yield expr?</code>	(YIELD)
		<code>yield from expr</code>	(YIELDFROM)
		<code>expr (compop expr)*</code>	(COMPOP)
		<code>expr (expr* keyword*)</code>	(CALL)
		<code>{expr (!i)? (:expr)?}</code>	(FORMATTEDVALUE)
		<code>expr*</code>	(JOINEDSTR)
		<code>constant</code>	(CONSTANT)
		<code>expr.id</code>	(ATTRIBUTE)
		<code>expr[expr]</code>	(SUBSCRIPT)
		<code>*expr</code>	(STARRED)
		<code>**expr</code>	(DOUBLESTARRED)
		<code>id</code>	(NAME)
		<code>expr? (:expr)? (:expr)?</code>	(SLICE)
<i>boolop</i>	<code>::=</code>	<code>and   or</code>	(BOOLOPERATOR)
<i>binop</i>	<code>::=</code>	<code>+   -   *   @   /   **   &lt;&lt;   &gt;&gt;       ^   &amp;   //   %</code>	(BINOPERATOR)
<i>unop</i>	<code>::=</code>	<code>~   not   +   -</code>	(UNOPERATOR)
<i>compop</i>	<code>::=</code>	<code>==   !=   &lt;   &lt;=   &gt;   &gt;=   is   is not   in   not in</code>	(COMPOP)
<i>comprehension</i>	<code>::=</code>	<code>for expr in expr (if expr)*</code>	(COMPREHENSION)
		<code>async for expr in expr (if expr)*</code>	(ASYNCCOMPREHEN)
<i>exc.handler</i>	<code>::=</code>	<code>except expr? (as id)? : stmt*</code>	(EXCHANDLER)
<i>args</i>	<code>::=</code>	<code>(arg (= expr)?)*, (arg (= expr)?)*, arg?, (arg (= expr)?)*, arg?</code>	(ARGUMENTS)
<i>arg</i>	<code>::=</code>	<code>id expr? s?</code>	(ARGUMENT)
<i>keyword</i>	<code>::=</code>	<code>id? = expr</code>	(KEYWORD)
<i>alias</i>	<code>::=</code>	<code>id (.id)* (as id)?</code>	(ALIAS)
<i>with.item</i>	<code>::=</code>	<code>expr (as expr)?</code>	(WITHITEM)

<i>match_case</i>	<code>::= case pattern (if expr)? : stmt*</code>	(MATCHCASE)
<i>pattern</i>	<code>::= expr</code>	(MATCHVALUE)
	<i>constant</i>	(MATCHSINGLETON)
	[ <i>pattern</i> *]	(MATCHSEQUENCE)
	<i>*(id)?</i>	(MATCHSTAR)
	{( <i>expr</i> : <i>pattern</i> )* <i>id</i> ?}	(MATCHMAPPING)
	<i>expr</i> ( <i>pattern</i> * ( <i>id</i> = <i>pattern</i> )*)	(MATCHCLASS)
	( <i>pattern as</i> )? <i>id</i>	(MATCHAS)
	<i>pattern</i>   <i>pattern</i>	(MATCHOR)
	-	(MATCHWILDCARD)
<i>constant</i>	<code>::= None</code>	(NONELITERAL)
	<i>i</i>	(INTLITERAL)
	<i>f</i>	(FLOATLITERAL)
	<i>c</i>	(COMPLEXLITERAL)
	<i>s</i>	(STRINGLITERAL)
	<i>b</i>	(BOOLEANLITERAL)
	( <i>constant</i> *)	(TUPLELITERAL)
	...	(ELLIPSIS)
<i>type_ignore</i>	<code>::= i*</code>	(TYPEIGNORE)
<i>id</i>	$\in Id$	
<i>s</i>	$\in Str$	
<i>b</i>	$\in \{\text{True}, \text{False}\}$	
<i>i</i>	$\in \mathbb{Z}$	
<i>f</i>	$\in \mathbb{R}$	
<i>c</i>	$\in \mathbb{C}$	

Note: there may be more constant terms such as an immutable container containing only constant elements. Please update the *constant* rule if you know such terms.

## 2 Transformation Rule for TensorFlow ML Training Code

### 2.1 Restrictions

1. All import statements must be placed at the top of a module.
2. The `tensorflow` module must be assigned to a variable only using an import statement.
3. Members of the `tensorflow` module must not be aliased.
4. `print` function call expressions must not introduce side-effects.
5. A dataset and an optimizer object must only be created once via an assignment statement with a function call expression and must not be aliased.
6. Variables storing a dataset or a optimizer object must not be reassigned to store another value that is not a dataset or an optimizer object.
7. A dataset and an optimizer object must not be created conditionally.
8. `optimizer.apply_gradients` function call expressions must be expression statements or a direct RHS of an assignment statement.
9. Global variables storing an `optimizer` object and referred in functions must be defined before the functions' definitions and must not be changed after their initializations.
10. A checkpoint object must only be created once via an assignment statement with a function call expression and must not be aliased.

### 2.2 Rules

#### 2.2.1 Types and Auxiliary Functions

$\tau ::=$	<i>Module</i>	(MODULES)
	<i>Stmt</i>	(STATEMENTS)
	<i>Expr</i>	(EXPRESSIONS)
	<i>Comprehension</i>	(COMPREHENSIONS)
	<i>ExcHandler</i>	(EXCEPTION HANDLERS)
	<i>Alias</i>	(ALIASES)
	<i>WithItem</i>	(WITHITEMS)
	<i>Pattern</i>	(PATTERNS)
	<i>Id</i>	(IDENTIFIERS)
	<i>Str</i>	(STRINGS)
	$\tau$ list	(LIST OF ELEMENTS TYPED $\tau$ )

$\sigma \in \Sigma = Str \xrightarrow{\text{fin}} Id$       ENVIRONMENT STORING MAPPINGS FROM STRINGS TO IDENTIFIERS

<b>.1</b>	:	$\tau \times \tau \rightarrow \tau$	GET THE FIRST ELEMENT OF THE GIVEN PAIR
<b>@</b>	:	$\tau \text{ list} \rightarrow \tau \text{ list} \rightarrow \tau \text{ list}$	CONCATENATE TWO LISTS
<b>::</b>	:	$\tau \rightarrow \tau \text{ list} \rightarrow \tau \text{ list}$	APPEND AN ELEMENT TO A LIST (RIGHT-ASSOCIATIVE)
<b>&lt;:</b>	:	$Expr \times Expr \rightarrow \mathbb{B}$	WHETHER CLASSES SPECIFIED BY THE EXPRESSIONS ARE SUBCLASSES

## 2.3 Rule: TensorFlow 2.x Keras API Pattern

### 2.3.1 Transformation Rules

$trans_M : Module \rightarrow Module$
---------------------------------------

$$trans_M \llbracket stmt^* \text{ type\_ignore} \rrbracket = trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1 \text{ type\_ignore}$$

$trans_{\bar{S}} : Stmt \text{ list} \rightarrow \Sigma \rightarrow (Stmt \text{ list} \times \Sigma)$
--

$$\begin{aligned} trans_{\bar{S}} \llbracket stmt_1 \text{ } stmt_2 \dots stmt_n \rrbracket(\sigma) = & \text{LET } stmt_1', \sigma_1 = trans_S \llbracket stmt_1 \rrbracket(\sigma) \text{ IN} \\ & \text{LET } stmt_2', \sigma_2 = trans_S \llbracket stmt_2 \rrbracket(\sigma_1) \text{ IN} \\ & \dots \\ & \text{LET } stmt_n', \sigma_n = trans_S \llbracket stmt_n \rrbracket(\sigma_{n-1}) \text{ IN} \\ & (stmt_1' @ stmt_2' @ \dots @ stmt_n', \sigma_n) \end{aligned}$$

$trans_S : Stmt \rightarrow \Sigma \rightarrow (Stmt \text{ list} \times \Sigma)$
---

$$\begin{aligned} trans_S \llbracket (@expr_1)^* \text{ def } id (args) (-> expr_2)? : (\#type:s)? stmt^* \rrbracket(\sigma) = \\ ([(@expr_1)^* \text{ def } id (args) (-> expr_2)? : (\#type:s)? trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1], \sigma) \\ trans_S \llbracket (@expr_1)^* \text{ async def } id (args) (-> expr_2)? : (\#type:s)? stmt^* \rrbracket(\sigma) = \\ ([(@expr_1)^* \text{ async def } id (args) (-> expr_2)? : (\#type:s)? trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1], \sigma) \\ trans_S \llbracket (@expr_1)^* \text{ class } id (expr_2^* keyword^*) : stmt^* \rrbracket(\sigma) = \\ ([(@expr_1)^* \text{ class } id (expr_2^* keyword^*) : trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1], \sigma) \\ trans_S \llbracket \text{return } expr? \rrbracket(\sigma) = ([\text{return } (trans_E \llbracket expr \rrbracket(\sigma))], \sigma) \\ trans_S \llbracket \text{delete } expr^* \rrbracket(\sigma) = ([\text{delete } expr^*], \sigma) \end{aligned}$$

#### A strict form of assignment statements

$$\begin{aligned} trans_S \llbracket id_r = expr_1 (expr_{11} \dots expr_{1n} (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) (\#type:s)? \rrbracket(\sigma) = \\ \text{\textcolor{blue}{\# Learning rate scheduler scaling}} \\ \text{IF } expr_1 <: \text{tensorflow.keras.optimizers.schedules.PiecewiseConstantDecay THEN} \\ ([id_r = expr_1 (expr_{11} \dots expr_{1n} (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) (\#type:s)?], \\ \sigma["lr\_scheduler" \mapsto id_r]) \\ \text{ELIF } expr_1 <: \text{tensorflow.keras.experimental.PiecewiseConstantDecay THEN} \\ ([id_r = expr_1 (expr_{11} \dots expr_{1n} (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) (\#type:s)?], \\ \sigma["lr\_scheduler" \mapsto id_r]) \\ \text{ELIF } expr_1 <: \text{tensorflow.keras.optimizers.schedules.LearningRateSchedule THEN} \\ \text{IF } id_i = \text{initial\_learning\_rate WHEN } 1 \leq i \leq k \text{ THEN} \\ ([id_r = expr_1 (expr_{11} \dots expr_{1n} (id_1 = )? expr_{21} \dots id_i = expr_{2i} * \text{hvd.size}() \\ \dots (id_k = )? expr_{2k}) (\#type:s)?], \sigma["lr\_scheduler" \mapsto id_r]) \\ \text{ELSE} \\ ([id_r = expr_1 (expr_{11} * \text{hvd.size}() \dots expr_{1n} (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) (\#type:s)?], \\ \sigma["lr\_scheduler" \mapsto id_r]) \\ \text{ELIF } expr_1 <: \text{tensorflow.compat.v1.train.exponential.decay THEN} \end{aligned}$$

```

IF  $id_i = \text{learning\_rate}$  WHEN  $1 \leq i \leq k$  THEN
  ( $[id_r = \text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots id_i = \text{expr}_{2i} * \text{hvd.size}()$ 
     $\dots (id_k = )? \text{expr}_{2k}) \ (\#type:s)?], \sigma["lr\_scheduler" \mapsto id_r]$ )
ELSE
  ( $[id_r = \text{expr}_1 (\text{expr}_{11} * \text{hvd.size}() \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots (id_k = )? \text{expr}_{2k}) \ (\#type:s)?],$ 
     $\sigma["lr\_scheduler" \mapsto id_r]$ )
# Optimizer learning rate scaling and wrapping
ELIF  $\text{expr}_1 <: \text{tensorflow.keras.optimizers.Optimizer}$  THEN
  IF  $id_i = \text{learning\_rate}$  WHEN  $1 \leq i \leq k$  THEN
    ( $[id_r = \text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots id_i = \text{expr}_{2i} * \text{hvd.size}()$ 
       $\dots (id_k = )? \text{expr}_{2k}) \ (\#type:s)?,$ 
       $id_r = \text{hvd.DistributedOptimizer}(id_r)] \ \sigma["optimizer" \mapsto id_r]$ )
  ELSE
    ( $[id_r = \text{expr}_1 (\text{expr}_{11} * \text{hvd.size}() \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots (id_k = )? \text{expr}_{2k}) \ (\#type:s)?,$ 
       $id_r = \text{hvd.DistributedOptimizer}(id_r)] \ \sigma["optimizer" \mapsto id_r]$ )
# Model related
ELIF  $\text{expr}_1 <: \text{tensorflow.keras.models.Model}$  THEN
  ( $[id_r = \text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots (id_k = )? \text{expr}_{2k}) \ (\#type:s)?],$ 
     $\sigma["model" \mapsto id_r]$ )
ELIF  $\sigma["model"] = id_t$  AND  $\text{expr}_1 = id_t.\text{evaluate}$  THEN
  IF  $id_i = \text{verbose}$  WHEN  $1 \leq i \leq k$  THEN
    ( $[id_r = \text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots id_i = 1 \text{ if } \text{hvd.rank}() == 0 \text{ else } 0$ 
       $\dots (id_k = )? \text{expr}_{2k}) \ (\#type:s)?], \sigma$ )
  ELSE
    ( $[id_r = \text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots (id_k = )? \text{expr}_{2k}$ 
       $\text{verbose} = 1 \text{ if } \text{hvd.rank}() == 0 \text{ else } 0) \ (\#type:s)?], \sigma$ )
# Checkpoint
ELIF  $\sigma["\text{tensor\_flow}"] = id_t$  AND  $\text{expr}_1 = id_t.\text{train.Checkpoint}$  THEN
  ( $[id_r = \text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots (id_k = )? \text{expr}_{2k}) \ (\#type:s)?],$ 
     $\sigma["checkpoint" \mapsto id_r]$ )
# Default
ELSE ( $[id_r = \text{trans}_E \llbracket \text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_2 = )? \text{expr}_{21} \dots (id_k = )? \text{expr}_{2k}) \rrbracket (\sigma) \ (\#type:s)?],$ 
   $\sigma$ )

```

```

 $\text{trans}_S \llbracket \text{expr}_0 = \text{expr}_1 \ (\#type:s)? \rrbracket (\sigma) =$ 
  IF  $\sigma["os"] = id_{os}$  AND  $\text{expr}_0 = id_{os}.\text{environ} ['CUDA\_VISIBLE\_DEVICES']$ 
  THEN ( $\llbracket \cdot \rrbracket, \sigma$ )

```

**A general form of assignment statements except for the strict form**

```

 $\text{trans}_S \llbracket \text{expr}_1^* = \text{expr}_2 \ (\#type:s)? \rrbracket (\sigma) = ([\text{expr}_1^* = \text{trans}_E \llbracket \text{expr}_2 \rrbracket (\sigma) \ (\#type:s)?], \sigma)$ 

```

```

 $\text{trans}_S \llbracket \text{expr}_1 \text{ binop} = \text{expr}_2 \rrbracket (\sigma) = ([\text{expr}_1 \text{ binop} = \text{trans}_E \llbracket \text{expr}_2 \rrbracket (\sigma)], \sigma)$ 

```

$trans_S \llbracket expr_1 : expr_2 (= expr_3)? \rrbracket(\sigma) = ([expr_1 : expr_2 (= trans_E \llbracket expr_3 \rrbracket(\sigma))?), \sigma)$   
*# Dataset in Assign*  
**IF**  $expr_1 = id_1$  **AND**  $\sigma(\text{"tensor\_flow"}) = id_2$   
**AND**  $expr_3 = id_2.data.Dataset.expr_4 (expr_5^* keyword^*)$  **THEN**  
 $([expr_1 : expr_2 = expr_3 (\#type:s)?], \sigma[\text{"dataset"} \mapsto id_1])$   
**ELSE**  $([expr_1 : expr_2 (= trans_E \llbracket expr_3 \rrbracket(\sigma))?), \sigma)$

$trans_S \llbracket (\#type:s)? \text{ for } expr_1 \text{ in } expr_2 : stmt_1^* (\text{else} : stmt_2^*)? \rrbracket(\sigma) =$   
 $([(\#type:s)? \text{ for } expr_1 \text{ in } trans_E \llbracket expr_2 \rrbracket(\sigma) :$   
 $trans_{\overline{S}} \llbracket stmt_1^* \rrbracket(\sigma).1 (\text{else} : trans_{\overline{S}} \llbracket stmt_2^* \rrbracket(\sigma).1)?], \sigma)$

$trans_S \llbracket (\#type:s)? \text{ async for } expr_1 \text{ in } expr_2 : stmt_1^* (\text{else} : stmt_2^*)? \rrbracket(\sigma) =$   
 $([(\#type:s)? \text{ async for } expr_1 \text{ in } trans_E \llbracket expr_2 \rrbracket(\sigma) :$   
 $trans_{\overline{S}} \llbracket stmt_1^* \rrbracket(\sigma).1 (\text{else} : trans_{\overline{S}} \llbracket stmt_2^* \rrbracket(\sigma).1)?], \sigma)$

$trans_S \llbracket \text{while } (expr) : stmt_1^* (\text{else} : stmt_2^*)? \rrbracket(\sigma) =$   
 $([\text{while } (trans_E \llbracket expr \rrbracket(\sigma)) : trans_{\overline{S}} \llbracket stmt_1^* \rrbracket(\sigma).1 (\text{else} : trans_{\overline{S}} \llbracket stmt_2^* \rrbracket(\sigma).1)?], \sigma)$

$trans_S \llbracket \text{if } (expr) : stmt^* (\text{else} : stmt^*)? \rrbracket(\sigma) =$   
 $([\text{if } (trans_E \llbracket expr \rrbracket(\sigma)) : trans_{\overline{S}} \llbracket stmt^* \rrbracket(\sigma).1 (\text{else} : trans_{\overline{S}} \llbracket stmt^* \rrbracket(\sigma).1)?], \sigma)$

$trans_S \llbracket (\#type:s)? \text{ with } with\_item^* : stmt^* \rrbracket(\sigma) =$   
**LET**  $with\_item^{*'}, \sigma_1 = trans_{\overline{W}} \llbracket with\_item^* \rrbracket(\sigma)$  **IN**  
**LET**  $stmt^{*'}, \sigma_2 = trans_{\overline{S}} \llbracket stmt^* \rrbracket(\sigma_1)$  **IN**  
**IF**  $\sigma_1 \setminus \sigma = [\text{"gradient\_tape"} \mapsto id]$  **THEN**  
 $([(\#type:s)? \text{ with } with\_item^{*'} : stmt^{*'},$   
 $id = hvd.DistributedGradientTape(id)], \sigma_2)$   
**ELSE**  $([(\#type:s)? \text{ with } with\_item^{*'} : stmt^{*'}], \sigma_2)$

$trans_S \llbracket (\#type:s)? \text{ async with } with\_item^* : stmt^* \rrbracket(\sigma) =$   
**LET**  $with\_item^{*'}, \sigma_1 = trans_{\overline{W}} \llbracket with\_item^* \rrbracket(\sigma)$  **IN**  
**LET**  $stmt^{*'}, \sigma_2 = trans_{\overline{S}} \llbracket stmt^* \rrbracket(\sigma_1)$  **IN**  
**IF**  $\sigma_1 \setminus \sigma = [\text{"gradient\_tape"} \mapsto id]$  **THEN**  
 $([(\#type:s)? \text{ async with } with\_item^{*'} : stmt^{*'},$   
 $id = hvd.DistributedGradientTape(id)], \sigma_2)$   
**ELSE**  $([(\#type:s)? \text{ async with } with\_item^{*'} : stmt^{*'}], \sigma_2)$

$trans_S \llbracket \text{match } expr : match\_case^* \rrbracket(\sigma) =$   
 $([\text{match } trans_E \llbracket expr \rrbracket(\sigma) : (trans_C \llbracket match\_case \rrbracket(\sigma))^*], \sigma)$

$trans_S \llbracket \text{raise } expr_1? (\text{from } expr_2)? \rrbracket(\sigma) = ([\text{raise } expr_1? (\text{from } expr_2)?], \sigma)$

$trans_S \llbracket \text{try} : stmt_1^* exc\_handler^* (\text{else} : stmt_2^*)? (\text{finally} : stmt_3^*)? \rrbracket(\sigma) =$   
 $([\text{try} : trans_{\overline{S}} \llbracket stmt_1^* \rrbracket(\sigma).1 (trans_H \llbracket exc\_handler \rrbracket(\sigma))^*$   
 $(\text{else} : trans_{\overline{S}} \llbracket stmt_2^* \rrbracket(\sigma).1)? (\text{finally} : trans_{\overline{S}} \llbracket stmt_3^* \rrbracket(\sigma).1)?], \sigma)$

$trans_S \llbracket \text{assert } expr_1 \text{ } expr_2? \rrbracket(\sigma) = ([\text{assert } trans_E \llbracket expr_1 \rrbracket(\sigma) \text{ } expr_2?, \sigma)$

$trans_S \llbracket \text{import } alias^* \rrbracket(\sigma) =$   
**LET**  $\sigma_1 = trans_{\bar{A}} \llbracket alias^* \rrbracket(\sigma)$  **IN**  
**IF**  $\sigma_1 \setminus \sigma = [\text{“tensor\_flow”} \mapsto id]$  **THEN**  
 ([import  $alias^*$ ,  
 import horovod.tensorflow as hvd,  
 hvd.broadcast\_done = False,  
 hvd.init(),  
 gpus =  $id.config.experimental.list\_physical\_devices('GPU')$ ,  
 for gpu in gpus:  $id.config.experimental.set\_memory\_growth(gpu, True)$ ,  
 if gpus:  $id.config.experimental.set\_visible\_devices(gpus[hvd.local\_rank()], 'GPU')$ ],  $\sigma_1$ )  
**ELSE** ([import  $alias^*$ ],  $\sigma_1$ )

$trans_S \llbracket \text{import } alias^* \rrbracket(\sigma) =$   
**LET**  $\sigma_1 = trans_{\bar{A}} \llbracket alias^* \rrbracket(\sigma)$  **IN**  
**IF**  $\sigma_1 \setminus \sigma = [\text{“tensor\_flow”} \mapsto id]$  **THEN**  
 ([import  $alias^*$ ,  
 import horovod.tensorflow as hvd,  
 hvd.init(),  
 gpus =  $id.config.experimental.list\_physical\_devices('GPU')$ ,  
 for gpu in gpus:  $id.config.experimental.set\_memory\_growth(gpu, True)$ ,  
 if gpus:  $id.config.experimental.set\_visible\_devices(gpus[hvd.local\_rank()], 'GPU')$ ],  $\sigma_1$ )  
**ELSE** ([import  $alias^*$ ],  $\sigma_1$ )

$trans_S \llbracket \text{from } i \text{ } id? \text{ import } alias^* \rrbracket(\sigma) =$   
**LET**  $\sigma_1 = trans_{\bar{A}} \llbracket alias^* \rrbracket(\sigma)$  **IN**  
**IF**  $id = \text{tensorflow}$  **AND**  $\sigma_1 \setminus \sigma = [\text{“keras”} \mapsto id]$  **THEN**  
 ([from  $i \text{ } id?$  import  $alias^*$ ,  
 import horovod.tensorflow.keras as hvd,  
 hvd.init(),  
 gpus =  $id.config.experimental.list\_physical\_devices('GPU')$ ,  
 for gpu in gpus:  $id.config.experimental.set\_memory\_growth(gpu, True)$ ,  
 if gpus:  $id.config.experimental.set\_visible\_devices(gpus[hvd.local\_rank()], 'GPU')$ ],  $\sigma_1$ )  
**ELSE** ([from  $i \text{ } id?$  import  $alias^*$ ],  $\sigma_1$ )

$trans_S \llbracket \text{global } id^* \rrbracket(\sigma) = ([\text{global } id^*], \sigma)$

$trans_S \llbracket \text{nonlocal } id^* \rrbracket(\sigma) = ([\text{nonlocal } id^*], \sigma)$

#### A strict form of expr statements

$trans_S \llbracket expr_1 (expr_{11} \dots expr_{1n} \text{ } (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) \rrbracket(\sigma) =$   
**IF**  $\sigma(\text{“optimizer”}) = id_t$  **AND**  $expr_1 = id_t.apply\_gradients$  **THEN**



```

IF  $id_i = \text{grads\_and\_vars}$  WHEN  $1 \leq i \leq k$  THEN
  LET  $id_z = \text{NewID}()$  IN
    ( $id_z = \text{expr}_{2i}$ ,
     $\text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots id_i = id_z \dots (id_k = )? \text{expr}_{2k})$ ,
    global hvd_broadcast_done,
    if not hvd_broadcast_done: [ hvd.broadcast_variables([x[1] for x in  $id_z$ ], root_rank=0),
      hvd.broadcast_variables( $id_t$ .variables(), root_rank=0),
      hvd_broadcast_done = True ],  $\sigma$ )
ELSE
  LET  $id_z = \text{NewID}()$  IN
    ( $id_z = \text{expr}_{11}$ ,
     $\text{expr}_1 (id_z \text{ expr}_{12} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots (id_k = )? \text{expr}_{2k})$ ,
    global hvd_broadcast_done,
    if not hvd_broadcast_done: [ hvd.broadcast_variables([x[1] for x in  $id_z$ ], root_rank=0),
      hvd.broadcast_variables( $id_t$ .variables(), root_rank=0),
      hvd_broadcast_done = True ],  $\sigma$ )
# MainScript rule
ELIF  $\text{expr}_1 = \text{print}$  OR  $\text{expr}_1 = \text{expr.write}$  OR  $\text{expr}_1 = \text{expr.summary}$  OR
   $\text{expr}_1 = \text{expr.save\_weights}$  OR  $\text{expr}_1 = \text{expr.load\_weights}$  THEN
  ( $[\text{if hvd.rank()} == 0: \text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots (id_k = )? \text{expr}_{2k})]$ ,  $\sigma$ )
ELIF  $\sigma(\text{"checkpoint"}) = id_t$  AND  $\text{expr}_1 = id_t.\text{save}$  THEN
  ( $[\text{if hvd.rank()} == 0: [\text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots (id_k = )? \text{expr}_{2k})]]$ ,  $\sigma$ )
ELIF  $\sigma(\text{"model"}) = id_t$  AND  $\text{expr}_1 = id_t.\text{fit}$  THEN
  IF  $id_i = \text{verbose}$  WHEN  $1 \leq i \leq k$  THEN
    IF  $id_j = \text{callbacks}$  WHEN  $1 \leq j \leq k$  THEN
      ( $[\text{callback} = [\text{hvd.callbacks.BroadcastGlobalVariablesCallback}(\text{root\_rank}=0)]$ 
      if hvd.rank() == 0:  $\text{callbacks.append}(\text{expr}_{2j})$ 
       $\text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots id_i = 1$  if hvd.rank() == 0 else 0
       $\dots id_j = \text{callbacks} \dots (id_k = )? \text{expr}_{2k})]$ ,  $\sigma$ )
    ELSE
      ( $[\text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots$ 
       $id_i = 1$  if hvd.rank() == 0 else 0  $\dots (id_k = )? \text{expr}_{2k}$ 
       $\text{callbacks} = [\text{hvd.callbacks.BroadcastGlobalVariablesCallback}(\text{root\_rank}=0)]]$ ,  $\sigma$ )
    ELSE
      IF  $id_j = \text{callbacks}$  WHEN  $1 \leq j \leq k$  THEN
        ( $[\text{callback} = [\text{hvd.callbacks.BroadcastGlobalVariablesCallback}(\text{root\_rank}=0)]$ 
        if hvd.rank() == 0:  $\text{callbacks.append}(\text{expr}_{2j})$ 
         $\text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots id_j = \text{callbacks} \dots (id_k = )? \text{expr}_{2k}$ 
         $\text{verbose} = 1$  if hvd.rank() == 0 else 0  $)]$ ,  $\sigma$ )
      ELSE
        ( $[\text{expr}_1 (\text{expr}_{11} \dots \text{expr}_{1n} \ (id_1 = )? \text{expr}_{21} \dots (id_k = )? \text{expr}_{2k}$ 
         $\text{verbose} = 1$  if hvd.rank() == 0 else 0
         $\text{callbacks} = [\text{hvd.callbacks.BroadcastGlobalVariablesCallback}(0)]]$ ,  $\sigma$ )
      ELIF  $\sigma(\text{"model"}) = id_t$  AND  $\text{expr}_1 = id_t.\text{compile}$  THEN

```

```

IF  $id_i = \text{optimizer}$  AND  $expr_{2i} = \text{"adam"}$  WHEN  $1 \leq i \leq k$  THEN
  ( $id_z = \text{tf.optimizers.Adam(learning\_rate=0.001 * hvd.size())}$ 
    $id_z = \text{hvd.DistributedOptimizer}(id_z)$ 
    $expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots id_i = id_z \dots (id_k = )? expr_{2k})], \sigma)$ 
ELIF  $expr_{11} = \text{"adam"}$  THEN
  ( $id_z = \text{tf.optimizers.Adam(learning\_rate=0.001 * hvd.size())}$ 
    $id_z = \text{hvd.DistributedOptimizer}(id_z)$ 
    $expr_1 (id_z \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k})], \sigma)$ 
ELSE
  ( $trans_E \llbracket expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) \rrbracket(\sigma), \sigma)$ 
ELIF  $expr_1 = \text{model.summary}$  THEN
  ( $[\text{if hvd.rank()} == 0: expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k})], \sigma)$ 
ELSE
  ( $trans_E \llbracket expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) \rrbracket(\sigma), \sigma)$ 

```

**A general form of expr statements except for the strict form**

$trans_S \llbracket expr \rrbracket(\sigma) = ([trans_E \llbracket expr \rrbracket(\sigma)], \sigma)$

$trans_S \llbracket \text{pass} \rrbracket(\sigma) = ([\text{pass}], \sigma)$

$trans_S \llbracket \text{break} \rrbracket(\sigma) = ([\text{break}], \sigma)$

$trans_S \llbracket \text{continue} \rrbracket(\sigma) = ([\text{continue}], \sigma)$

$trans_E : Expr \rightarrow \Sigma \rightarrow Expr$
--

$trans_E \llbracket expr_1 \ \text{boolop} \ expr_2 \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma) \ \text{boolop} \ trans_E \llbracket expr_2 \rrbracket(\sigma)$

$trans_E \llbracket expr_1 \ := \ expr_2 \rrbracket(\sigma) = expr_1 \ := \ trans_E \llbracket expr_2 \rrbracket(\sigma)$

$trans_E \llbracket expr_1 \ \text{binop} \ expr_2 \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma) \ \text{binop} \ trans_E \llbracket expr_2 \rrbracket(\sigma)$

$trans_E \llbracket \text{unop} \ expr \rrbracket(\sigma) = \text{unop} \ trans_E \llbracket expr \rrbracket(\sigma)$

$trans_E \llbracket \text{lambda} \ args : expr \rrbracket(\sigma) = \text{lambda} \ args : trans_E \llbracket expr \rrbracket(\sigma)$

$trans_E \llbracket expr_1 \ \text{if} \ expr_2 \ \text{else} \ expr_3 \rrbracket(\sigma) =$   
 $trans_E \llbracket expr_1 \rrbracket(\sigma) \ \text{if} \ trans_E \llbracket expr_2 \rrbracket(\sigma) \ \text{else} \ trans_E \llbracket expr_3 \rrbracket(\sigma)$

$trans_E \llbracket expr_1 \ \text{if} \ expr_2 \ \text{else} \ expr_3 \rrbracket(\sigma) =$   
 $trans_E \llbracket expr_1 \rrbracket(\sigma) \ \text{if} \ trans_E \llbracket expr_2 \rrbracket(\sigma) \ \text{else} \ trans_E \llbracket expr_3 \rrbracket(\sigma)$

$trans_E \llbracket \{(expr_1 : expr_2)^* \ (**expr_3)^*\} \rrbracket(\sigma) = \{(expr_1 : trans_E \llbracket expr_2 \rrbracket(\sigma))^* \ (**expr_3)^*\}$

$$trans_E \llbracket \{expr^*\} \rrbracket(\sigma) = \{(trans_E \llbracket expr \rrbracket(\sigma))^*\}$$

$$trans_E \llbracket [expr^*] \rrbracket(\sigma) = [(trans_E \llbracket expr \rrbracket(\sigma))^*]$$

$$trans_E \llbracket (expr^*) \rrbracket(\sigma) = ((trans_E \llbracket expr \rrbracket(\sigma))^*)$$

$$trans_E \llbracket \{expr_1 : expr_2 \text{ comprehension}^*\} \rrbracket(\sigma) = \{expr_1 : trans_E \llbracket expr_2 \rrbracket(\sigma) \ (trans_O \llbracket comprehension \rrbracket(\sigma))^*\}$$

$$trans_E \llbracket \{expr \text{ comprehension}^*\} \rrbracket(\sigma) = \{trans_E \llbracket expr \rrbracket(\sigma) \ (trans_O \llbracket comprehension \rrbracket(\sigma))^*\}$$

$$trans_E \llbracket [expr \text{ comprehension}^*] \rrbracket(\sigma) = [(trans_E \llbracket expr \rrbracket(\sigma) \ (trans_O \llbracket comprehension \rrbracket(\sigma))^*)]$$

$$trans_E \llbracket (expr \text{ comprehension}^*) \rrbracket(\sigma) = (trans_E \llbracket expr \rrbracket(\sigma) \ (trans_O \llbracket comprehension \rrbracket(\sigma))^*)$$

$$trans_E \llbracket \text{await } expr \rrbracket(\sigma) = \text{await } trans_E \llbracket expr \rrbracket(\sigma)$$

$$trans_E \llbracket \text{yield } expr? \rrbracket(\sigma) = \text{yield } (trans_E \llbracket expr \rrbracket(\sigma))?$$

$$trans_E \llbracket \text{yield from } expr \rrbracket(\sigma) = \text{yield from } trans_E \llbracket expr \rrbracket(\sigma)$$

$$trans_E \llbracket expr_1 \text{ (compop } expr_2)^* \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma) \text{ (compop } trans_E \llbracket expr_2 \rrbracket(\sigma))^*$$

$$trans_E \llbracket expr_1 \text{ (} expr_{11} \dots expr_{1n} \text{ (} id_1 = \text{ )? } expr_{21} \dots (id_k = \text{ )? } expr_{2k} \text{ )} \rrbracket(\sigma) =$$

**IF**  $\sigma(\text{"dataset"}) = id_t$  **AND**  $expr_1 = id_t.\text{take}$  **THEN**

**IF**  $id_i = \text{count}$  **WHEN**  $1 \leq i \leq k$  **THEN**

$expr_1 \text{ (} expr_{11} \dots expr_{1n} \text{ (} id_1 = \text{ )? } expr_{21} \dots id_i = expr_{2i} // \text{ hvd.size() } \dots (id_k = \text{ )? } expr_{2k} \text{ )}$

**ELSE**

$expr_1 \text{ (} expr_{11} // \text{ hvd.size() } \dots expr_{1n} \text{ (} id_1 = \text{ )? } expr_{21} \dots (id_k = \text{ )? } expr_{2k} \text{ )}$

**ELSE**

$trans_E \llbracket expr_1 \rrbracket(\sigma) \ (trans_E \llbracket expr_{11} \rrbracket(\sigma) \dots trans_E \llbracket expr_{1n} \rrbracket(\sigma)$

$(id_1 = \text{ )? } trans_E \llbracket expr_{21} \rrbracket(\sigma) \dots (id_k = \text{ )? } trans_E \llbracket expr_{2k} \rrbracket(\sigma))$

$$trans_E \llbracket \{expr_1 \text{ (!} i \text{ )? } (:expr_2)?\} \rrbracket(\sigma) = \{expr_1 \text{ (!} i \text{ )? } (:expr_2)?\}$$

$$trans_E \llbracket expr^* \rrbracket(\sigma) = expr^*$$

$$trans_E \llbracket \text{constant} \rrbracket(\sigma) = \text{constant}$$

$$trans_E \llbracket expr.id \rrbracket(\sigma) = (trans_E \llbracket expr \rrbracket(\sigma)).id$$

$$trans_E \llbracket expr_1[expr_2] \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma)[trans_E \llbracket expr_2 \rrbracket(\sigma)]$$

$$trans_E \llbracket *expr \rrbracket(\sigma) = *expr$$

$$trans_E \llbracket **expr \rrbracket(\sigma) = **expr$$

$$trans_E \llbracket id \rrbracket(\sigma) = id$$

$$trans_E \llbracket expr_1? (:expr_2)? (:expr_3)? \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma)? (:trans_E \llbracket expr_2 \rrbracket(\sigma))? (:trans_E \llbracket expr_3 \rrbracket(\sigma))?$$

$$\boxed{trans_O : Comprehension \rightarrow \Sigma \rightarrow Comprehension}$$

$$trans_O \llbracket \text{for } expr_1 \text{ in } expr_2 \text{ (if } expr_3)^* \rrbracket(\sigma) = \text{for } expr_1 \text{ in } trans_E \llbracket expr_2 \rrbracket(\sigma) \text{ (if } trans_E \llbracket expr_3 \rrbracket(\sigma))^*$$

$$\boxed{trans_H : ExcHandler \rightarrow \Sigma \rightarrow ExcHandler}$$

$$trans_H \llbracket \text{except } expr? \text{ (as } id)? : stmt^* \rrbracket(\sigma) = \text{except } expr? \text{ (as } id)? : trans_{\overline{S}} \llbracket stmt^* \rrbracket(\sigma).1$$

$$\boxed{trans_{\overline{A}} : Alias \text{ list} \rightarrow \Sigma \rightarrow \Sigma}$$

$$trans_{\overline{A}} \llbracket alias_1 \dots alias_{n-1} alias_n \rrbracket(\sigma) = trans_A \llbracket alias_n \rrbracket(trans_A \llbracket alias_{n-1} \rrbracket(\dots trans_A \llbracket alias_1 \rrbracket(\sigma)))$$

$$\boxed{trans_A : Alias \rightarrow \Sigma \rightarrow \Sigma}$$

$$trans_A \llbracket id \rrbracket(\sigma) =$$

**LET**  $id = \text{tensorflow}$  **THEN**  $\sigma["\text{tensorflow}"] \mapsto id$   
**LET**  $id = \text{keras}$  **THEN**  $\sigma["\text{keras}"] \mapsto id$   
**LET**  $id = \text{os}$  **THEN**  $\sigma["\text{os}"] \mapsto id$   
**LET**  $id = \text{optimizers}$  **THEN**  $\sigma["\text{optimizers}"] \mapsto id$   
**ELSE**  $\sigma$

$$trans_A \llbracket id_1 \text{ as } id_2 \rrbracket(\sigma) =$$

**LET**  $id_1 = \text{tensorflow}$  **THEN**  $\sigma["\text{tensorflow}"] \mapsto id_2$   
**LET**  $id_1 = \text{tensorflow.keras}$  **THEN**  $\sigma["\text{keras}"] \mapsto id_2$   
**LET**  $id_1 = \text{os}$  **THEN**  $\sigma["\text{os}"] \mapsto id_2$   
**LET**  $id = \text{optimizers}$  **THEN**  $\sigma["\text{optimizers}"] \mapsto id$   
**ELSE**  $\sigma$

$$trans_A \llbracket id_1 . id_2 (.id_3)^* \text{ (as } id_2)? \rrbracket(\sigma) = \sigma$$

$$\boxed{trans_{\overline{W}} : WithItem \text{ list} \rightarrow \Sigma \rightarrow (WithItem \text{ list} \times \Sigma)}$$

$trans_{\overline{W}} \llbracket with\_item_1 with\_item_2 \dots with\_item_n \rrbracket(\sigma) =$   
**LET**  $with\_item_1', \sigma_1 = trans_W \llbracket with\_item_1 \rrbracket(\sigma)$  **IN**  
**LET**  $with\_item_2', \sigma_2 = trans_W \llbracket with\_item_2 \rrbracket(\sigma_1)$  **IN**  
 $\dots$   
**LET**  $with\_item_n', \sigma_n = trans_W \llbracket with\_item_n \rrbracket(\sigma_{n-1})$  **IN**  
 $(with\_item_1' :: with\_item_2' :: \dots :: [with\_item_n'], \sigma_n)$

$trans_W : WithItem \rightarrow \Sigma \rightarrow (WithItem \times \Sigma)$
--

$trans_W \llbracket expr \rrbracket(\sigma) = (trans_E \llbracket expr \rrbracket(\sigma), \sigma)$

$trans_W \llbracket expr_1 \text{ as } expr_2 \rrbracket(smodenv) =$   
**IF**  $\sigma(\text{"tensor\_flow"}) = id_1$  **AND**  $expr_1 = id_1.\text{GradientTape}()$  **AND**  $expr_2 = id_2$  **THEN**  
 $(expr_1 \text{ as } expr_2, \sigma[\text{"gradient\_tape"} \mapsto id_2])$   
**ELSE**  $(trans_E \llbracket expr_1 \rrbracket(\sigma) \text{ as } expr_2, \sigma)$

$trans_C : MatchCase \rightarrow \Sigma \rightarrow MatchCase$
--

$trans_C \llbracket \text{case pattern (if expr)? : stmt*} \rrbracket(\sigma) =$   
 $\text{case } trans_P \llbracket pattern \rrbracket(\sigma) \text{ (if } trans_E \llbracket expr \rrbracket(\sigma))? : trans_{\overline{S}} \llbracket stmt* \rrbracket(\sigma).1$

$trans_P : Pattern \rightarrow \Sigma \rightarrow Pattern$
--

$trans_P \llbracket expr \rrbracket(\sigma) = trans_E \llbracket expr \rrbracket(\sigma)$

$trans_P \llbracket constant \rrbracket(\sigma) = constant$

$trans_P \llbracket [pattern*] \rrbracket(\sigma) = [trans_P \llbracket pattern \rrbracket(\sigma)*]$

$trans_P \llbracket *(id)? \rrbracket(\sigma) = *(id)?$

$trans_P \llbracket \{(expr : pattern)* id?\} \rrbracket(\sigma) = \{(expr : trans_P \llbracket pattern \rrbracket(\sigma))* id?\}$

$trans_P \llbracket expr (pattern_1^* (id = pattern_2^*)) \rrbracket(\sigma) = expr (trans_P \llbracket pattern_1 \rrbracket(\sigma)^* (id = trans_P \llbracket pattern_2 \rrbracket(\sigma))^*)$

$trans_P \llbracket (pattern \text{ as})? id \rrbracket(\sigma) = (trans_P \llbracket pattern \rrbracket(\sigma) \text{ as})? id$

$trans_P \llbracket pattern_1 \mid pattern_2 \rrbracket(\sigma) = trans_P \llbracket pattern_1 \rrbracket(\sigma) \mid trans_P \llbracket pattern_2 \rrbracket(\sigma)$

$trans_P \llbracket - \rrbracket(\sigma) = -$

## 2.4 Rule: TensorFlow 2.x GradientTape API Pattern

### 2.4.1 Transformation Rules

$trans_M : Module \rightarrow Module$
---------------------------------------

$$trans_M \llbracket stmt^* \text{ type\_ignore} \rrbracket = trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1 \text{ type\_ignore}$$

$trans_{\bar{S}} : Stmt \text{ list} \rightarrow \Sigma \rightarrow (Stmt \text{ list} \times \Sigma)$
--

$$\begin{aligned} trans_{\bar{S}} \llbracket stmt_1 \text{ } stmt_2 \dots stmt_n \rrbracket(\sigma) = & \text{LET } stmt_1', \sigma_1 = trans_S \llbracket stmt_1 \rrbracket(\sigma) \text{ IN} \\ & \text{LET } stmt_2', \sigma_2 = trans_S \llbracket stmt_2 \rrbracket(\sigma_1) \text{ IN} \\ & \dots \\ & \text{LET } stmt_n', \sigma_n = trans_S \llbracket stmt_n \rrbracket(\sigma_{n-1}) \text{ IN} \\ & (stmt_1' @ stmt_2' @ \dots @ stmt_n', \sigma_n) \end{aligned}$$

$trans_S : Stmt \rightarrow \Sigma \rightarrow (Stmt \text{ list} \times \Sigma)$
---

$$\begin{aligned} trans_S \llbracket (@expr_1)^* \text{ def } id (args) (-> expr_2)? : (\#type:s)? stmt^* \rrbracket(\sigma) = \\ ([(@expr_1)^* \text{ def } id (args) (-> expr_2)? : (\#type:s)? trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1], \sigma) \\ trans_S \llbracket (@expr_1)^* \text{ async def } id (args) (-> expr_2)? : (\#type:s)? stmt^* \rrbracket(\sigma) = \\ ([(@expr_1)^* \text{ async def } id (args) (-> expr_2)? : (\#type:s)? trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1], \sigma) \\ trans_S \llbracket (@expr_1)^* \text{ class } id (expr_2^* keyword^*) : stmt^* \rrbracket(\sigma) = \\ ([(@expr_1)^* \text{ class } id (expr_2^* keyword^*) : trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1], \sigma) \\ trans_S \llbracket \text{return } expr? \rrbracket(\sigma) = ([\text{return } (trans_E \llbracket expr \rrbracket(\sigma))?), \sigma) \\ trans_S \llbracket \text{delete } expr^* \rrbracket(\sigma) = ([\text{delete } expr^*], \sigma) \end{aligned}$$

#### A strict form of assignment statements

$$\begin{aligned} trans_S \llbracket id_r = expr_1 (expr_{11} \dots expr_{1n} (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) (\#type:s)? \rrbracket(\sigma) = \\ \text{\textcolor{blue}{\# Dataset}} \\ \text{IF } \sigma(\text{"tensor\_flow"}) = id_t \text{ AND } expr_1 = id_t.data.Dataset.expr_3 \text{ THEN} \\ ([id_r = expr_1 (expr_{11} \dots expr_{1n} (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) (\#type:s)?], \\ \sigma[\text{"dataset"} \mapsto id_r]) \\ \text{\textcolor{blue}{\# Variable Broadcasting}} \\ \text{ELIF } \sigma(\text{"optimizer"}) = id_t \text{ AND } expr_1 = id_t.apply\_gradients \text{ THEN} \\ \text{IF } id_i = grads\_and\_vars \text{ WHEN } 1 \leq i \leq k \text{ THEN} \\ \text{LET } id_z = \text{NewID}() \text{ IN} \\ ([id_z = expr_{2i}, \\ id_r = expr_1 (expr_{11} \dots expr_{1n} (id_1 = )? expr_{21} \dots id_i = id_z \dots (id_k = )? expr_{2k}) (\#type:s)?, \\ \text{global hvd\_broadcast\_done,} \\ \text{if not hvd\_broadcast\_done: [ hvd.broadcast\_variables([x[1] for x in id_z], root\_rank=0),} \\ \text{hvd.broadcast\_variables(id\_t.variables(), root\_rank=0),} \\ \text{hvd\_broadcast\_done = True }], \sigma) \\ \text{ELSE} \end{aligned}$$

```

LET  $id_z$  = NewID() IN
  ( $id_z$  =  $expr_{11}$ ,
   $id_r$  =  $expr_1$  ( $id_z$   $expr_{12}$  ...  $expr_{1n}$  ( $id_1$  = )?  $expr_{21}$  ... ( $id_k$  = )?  $expr_{2k}$ ) (#type:s)?,
  global hvd_broadcast_done,
  if not hvd_broadcast_done: [ hvd.broadcast_variables([x[1] for x in  $id_z$ ], root_rank=0),
    hvd.broadcast_variables( $id_t$ .variables(), root_rank=0),
    hvd_broadcast_done = True ],  $\sigma$ )
# Learning rate scheduler scaling
IF  $expr_1$  <: "tensorflow.keras.optimizers.schedules.PiecewiseConstantDecay" THEN
  ( $id_r$  =  $expr_1$  ( $expr_{11}$  ...  $expr_{1n}$  ( $id_1$  = )?  $expr_{21}$  ... ( $id_k$  = )?  $expr_{2k}$ ) (#type:s)?],
   $\sigma$ ["lr_scheduler"  $\mapsto id_r$ ])
ELIF  $expr_1$  <: tensorflow.keras.optimizers.schedules.LearningRateSchedule THEN
  IF  $id_i$  = initial_learning_rate WHEN  $1 \leq i \leq k$  THEN
    ( $id_r$  =  $expr_1$  ( $expr_{11}$  ...  $expr_{1n}$  ( $id_1$  = )?  $expr_{21}$  ...  $id_i$  =  $expr_{2i}$  * hvd.size()
    ... ( $id_k$  = )?  $expr_{2k}$ ) (#type:s)?],  $\sigma$ ["lr_scheduler"  $\mapsto id_r$ ])
  ELSE
    ( $id_r$  =  $expr_1$  ( $expr_{11}$  * hvd.size() ...  $expr_{1n}$  ( $id_1$  = )?  $expr_{21}$  ... ( $id_k$  = )?  $expr_{2k}$ ) (#type:s)?],
     $\sigma$ ["lr_scheduler"  $\mapsto id_r$ ])
ELIF  $expr_1$  <: tensorflow.compat.v1.train.exponential_decay THEN
  IF  $id_i$  = learning_rate WHEN  $1 \leq i \leq k$  THEN
    ( $id_r$  =  $expr_1$  ( $expr_{11}$  ...  $expr_{1n}$  ( $id_1$  = )?  $expr_{21}$  ...  $id_i$  =  $expr_{2i}$  * hvd.size()
    ... ( $id_k$  = )?  $expr_{2k}$ ) (#type:s)?],  $\sigma$ ["lr_scheduler"  $\mapsto id_r$ ])
  ELSE
    ( $id_r$  =  $expr_1$  ( $expr_{11}$  * hvd.size() ...  $expr_{1n}$  ( $id_1$  = )?  $expr_{21}$  ... ( $id_k$  = )?  $expr_{2k}$ ) (#type:s)?],
     $\sigma$ ["lr_scheduler"  $\mapsto id_r$ ])
# Optimizer learning rate scaling and wrapping
IF  $expr_1$  <: tensorflow.keras.optimizers.Optimizer THEN
  ( $id_r$  =  $expr_1$  ( $expr_{11}$  ...  $expr_{1n}$  ( $id_1$  = )?  $expr_{21}$  ... ( $id_k$  = )?  $expr_{2k}$ ) (#type:s)?],
   $\sigma$ ["optimizer"  $\mapsto id_r$ ])
# Model related
ELIF  $expr_1$  <: tensorflow.keras.Model THEN
  ( $id_r$  =  $expr_1$  ( $expr_{11}$  ...  $expr_{1n}$  ( $id_2$  = )?  $expr_{21}$  ... ( $id_k$  = )?  $expr_{2k}$ )],  $\sigma$ ["model"  $\mapsto id_r$ ])
ELIF  $id_m$  = "model" AND  $expr_1$  =  $id_t$ .evaluate THEN
  IF  $id_i$  = verbose WHEN  $1 \leq i \leq k$  THEN
    ( $id_r$  =  $expr_1$  ( $expr_{11}$  ...  $expr_{1n}$  ( $id_1$  = )?  $expr_{21}$  ...  $id_i$  = 1 if hvd.rank() == 0 else 0
    ... ( $id_k$  = )?  $expr_{2k}$ ) (#type:s)?],  $\sigma$ )
  ELSE
    ( $id_r$  =  $expr_1$  ( $expr_{11}$   $expr_{12}$  1 if hvd.rank() == 0 else 0 ...  $expr_{1n}$ 
    ( $id_1$  = )?  $expr_{21}$  ... ( $id_k$  = )?  $expr_{2k}$ ) (#type:s)?],  $\sigma$ )
# Checkpoint
ELIF  $\sigma$ ("tensor_flow") =  $id_t$  AND  $expr_1$  =  $id_t$ .train.Checkpoint THEN
  ( $id_r$  =  $expr_1$  ( $expr_{11}$  ...  $expr_{1n}$  ( $id_1$  = )?  $expr_{21}$  ... ( $id_k$  = )?  $expr_{2k}$ ) (#type:s)?],
   $\sigma$ ["checkpoint"  $\mapsto id_r$ ])
# Default

```

**ELSE**  $([id_r = trans_E \llbracket expr_1 (expr_{11} \dots expr_{1n} \ (id_2 = )? expr_{21} \dots (id_k = )? expr_{2k} \rrbracket (\sigma) \ (\#type:s)?], \sigma)$

**A general form of assignment statements except for the strict form**

$trans_S \llbracket expr_1^* = expr_2 \ (\#type:s)? \rrbracket (\sigma) =$

*# CUDA Visible Devices*

**IF**  $\sigma(\text{"os"}) = id_{os}$  **AND**  $expr_0 = id_{os}.environ[\text{"CUDA_VISIBLE_DEVICES"}]$  **THEN**  
 $([], \sigma)$

**ELSE**  $([expr_1^* = trans_E \llbracket expr_2 \rrbracket (\sigma) \ (\#type:s)?], \sigma)$

$trans_S \llbracket expr_1 \ binop = expr_2 \rrbracket (\sigma) = ([expr_1 \ binop = trans_E \llbracket expr_2 \rrbracket (\sigma)], \sigma)$

$trans_S \llbracket expr_1 : expr_2 (= expr_3)? \rrbracket (\sigma) =$

*# Dataset*

**IF**  $expr_1 = id_1$  **AND**  $\sigma(\text{"tensor\_flow"}) = id_2$   
**AND**  $expr_3 = id_2.data.Dataset.expr_4 (expr_5^* \ keyword^*)$  **THEN**  
 $([expr_1 : expr_2 = expr_3 \ (\#type:s)?], \sigma[\text{"dataset"} \mapsto id_1])$

**ELSE**  $([expr_1 : expr_2 (= trans_E \llbracket expr_3 \rrbracket (\sigma))?), \sigma)$

$trans_S \llbracket (\#type:s)? \ for \ expr_1 \ in \ expr_2 : stmt_1^* \ (else : stmt_2^*)? \rrbracket (\sigma) =$

*# Adjust Steps*

**IF**  $\sigma(\text{"config"}) = id_c$  **AND**  $(expr_2 = range(id_c.iterations\_per\_epoch) \ OR$   
 $expr_2 = tqdm(range(id_c.iterations\_per\_epoch)))$  **THEN**  
 $([(\#type:s)? \ for \ expr_1 \ in \ range(id_c.iterations\_per\_epoch \ // \ hvd.size()) :$   
 $trans_{\overline{S}} \llbracket stmt_1^* \rrbracket (\sigma).1 \ (else : trans_{\overline{S}} \llbracket stmt_2^* \rrbracket (\sigma).1)?], \sigma)$

**ELSE**  $([(\#type:s)? \ for \ expr_1 \ in \ trans_E \llbracket expr_2 \rrbracket (\sigma) :$   
 $trans_{\overline{S}} \llbracket stmt_1^* \rrbracket (\sigma).1 \ (else : trans_{\overline{S}} \llbracket stmt_2^* \rrbracket (\sigma).1)?], \sigma)$

$trans_S \llbracket (\#type:s)? \ async \ for \ expr_1 \ in \ expr_2 : stmt_1^* \ (else : stmt_2^*)? \rrbracket (\sigma) =$

*# Adjust Steps*

**IF**  $\sigma(\text{"config"}) = id_c$  **AND**  $(expr_2 = range(id_c.iterations\_per\_epoch) \ OR$   
 $expr_2 = tqdm(range(id_c.iterations\_per\_epoch)))$  **THEN**  
 $([(\#type:s)? \ async \ for \ expr_1 \ in \ range(id_c.iterations\_per\_epoch \ // \ hvd.size()) :$   
 $trans_{\overline{S}} \llbracket stmt_1^* \rrbracket (\sigma).1 \ (else : trans_{\overline{S}} \llbracket stmt_2^* \rrbracket (\sigma).1)?], \sigma)$

**ELSE**  $([(\#type:s)? \ async \ for \ expr_1 \ in \ trans_E \llbracket expr_2 \rrbracket (\sigma) :$   
 $trans_{\overline{S}} \llbracket stmt_1^* \rrbracket (\sigma).1 \ (else : trans_{\overline{S}} \llbracket stmt_2^* \rrbracket (\sigma).1)?], \sigma)$

$trans_S \llbracket while \ (expr) : stmt_1^* \ (else : stmt_2^*)? \rrbracket (\sigma) =$

$([while \ (trans_E \llbracket expr \rrbracket (\sigma)) : trans_{\overline{S}} \llbracket stmt_1^* \rrbracket (\sigma).1 \ (else : trans_{\overline{S}} \llbracket stmt_2^* \rrbracket (\sigma).1)?], \sigma)$

$trans_S \llbracket if \ (expr) : stmt^* \ (else : stmt^*)? \rrbracket (\sigma) =$

$([if \ (trans_E \llbracket expr \rrbracket (\sigma)) : trans_{\overline{S}} \llbracket stmt^* \rrbracket (\sigma).1 \ (else : trans_{\overline{S}} \llbracket stmt^* \rrbracket (\sigma).1)?], \sigma)$



```

transS[(#type:s)? with with_item* : stmt*](σ) =
  LET with_item*', σ1 = transW[(with_item*)](σ) IN
  LET stmt*', σ2 = transS[(stmt*)](σ1) IN
  # Wrapping Gradient Tape
  IF σ1 \ σ = ["gradient_tape" ↦ id] THEN
    (([#type:s)? with with_item*' : stmt*',
      id = hvd.DistributedGradientTape(id)], σ2)
  ELSE (([#type:s)? with with_item*' : stmt*'], σ2)

transS[(#type:s)? async with with_item* : stmt*](σ) =
  LET with_item*', σ1 = transW[(with_item*)](σ) IN
  LET stmt*', σ2 = transS[(stmt*)](σ1) IN
  # Wrapping Gradient Tape
  IF σ1 \ σ = ["gradient_tape" ↦ id] THEN
    (([#type:s)? async with with_item*' : stmt*',
      id = hvd.DistributedGradientTape(id)], σ2)
  ELSE (([#type:s)? async with with_item*' : stmt*'], σ2)

transS[(match expr : match_case*)](σ) =
  ([match transE[(expr)]](σ) : (transC[(match_case)](σ))*], σ)

transS[(raise expr1? (from expr2)?)](σ) = ([raise expr1? (from expr2)?], σ)

transS[(try : stmt1* exc_handler* (else : stmt2*)? (finally : stmt3*)?)](σ) =
  ([try : transS[(stmt1*)](σ).1 (transH[(exc_handler)](σ))*
    (else : transS[(stmt2*)](σ).1)? (finally : transS[(stmt3*)](σ).1)?], σ)

transS[(assert expr1 expr2?)](σ) = ([assert transE[(expr1)](σ) expr2?], σ)

transS[(import alias*)](σ) =
  LET σ1 = transA[(alias*)](σ) IN
  # import
  IF σ1 \ σ = ["tensor_flow" ↦ id] THEN
    ([import alias*,
      import horovod.tensorflow as hvd,
      hvd.broadcast_done = False,
      hvd.init(),
      gpus = id.config.experimental.list_physical_devices('GPU'),
      for gpu in gpus: id.config.experimental.set_memory_growth(gpu, True),
      if gpus: id.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')], σ1)
  ELSE ([import alias*], σ1)

transS[(global id*)](σ) = ([global id*], σ)

```

$trans_S \llbracket \text{nonlocal } id^* \rrbracket(\sigma) = ([\text{nonlocal } id^*], \sigma)$

#### A strict form of expr statements

$trans_S \llbracket expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) \rrbracket(\sigma) =$   
*# Variable Broadcasting*  
**IF**  $\sigma(\text{"optimizer"}) = id_t$  **AND**  $expr_1 = id_t.apply\_gradients$  **THEN**  
     **IF**  $id_i = grads\_and\_vars$  **WHEN**  $1 \leq i \leq k$  **THEN**  
         **LET**  $id_z = \text{NewID}()$  **IN**  
              $([id_z = expr_{2i},$   
              $expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots id_i = id_z \dots (id_k = )? expr_{2k}),$   
              $global \ hvd\_broadcast\_done,$   
             **if not**  $hvd\_broadcast\_done: [hvd.broadcast\_variables([x[1] \text{ for } x \text{ in } id_z], \text{root\_rank}=0),$   
              $hvd.broadcast\_variables(id_t.variables(), \text{root\_rank}=0),$   
              $hvd\_broadcast\_done = \text{True}] , \sigma)$   
     **ELSE**  
         **LET**  $id_z = \text{NewID}()$  **IN**  
              $([id_z = expr_{11},$   
              $expr_1 (id_z \ expr_{12} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}),$   
              $global \ hvd\_broadcast\_done,$   
             **if not**  $hvd\_broadcast\_done: [hvd.broadcast\_variables([x[1] \text{ for } x \text{ in } id_z], \text{root\_rank}=0),$   
              $hvd.broadcast\_variables(id_t.variables(), \text{root\_rank}=0),$   
              $hvd\_broadcast\_done = \text{True}] , \sigma)$   
     *# Config*  
     **IF**  $id_t = \sigma(\text{"tensor\_flow"})$  **AND**  
          $expr_1 = id_t.config.experimental.set\_visible\_devices$  **THEN**  $([], \sigma)$   
     *# Root Rank Blocking*  
     **ELIF**  $id_m = \sigma(\text{"model"})$  **AND**  $(expr_1 = id_m.write$  **OR**  
          $expr_1 = id_m.summary$  **OR**  $expr_1 = id_m.save\_weights$  **OR**  
          $expr_1 = expr.load\_weights$  **OR**  $expr_1 = id_m.save)$  **OR**  
          $id_c = \sigma(\text{"checkpoint"})$  **AND**  $expr_1 = id_c.save$  **OR**  
          $id_t = \sigma(\text{"tensor\_flow"})$  **AND**  $expr_1 = id_t.print$  **OR**  
          $expr_1 = \text{print}$  **THEN**  
              $([\text{if } hvd.rank() == 0: expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k})], \sigma)$   
     **ELSE**  
          $[trans_E \llbracket expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) \rrbracket(\sigma), \sigma)$

#### A general form of expr statements except for the strict form

$trans_S \llbracket expr \rrbracket(\sigma) = ([trans_E \llbracket expr \rrbracket(\sigma)], \sigma)$

$trans_S \llbracket \text{pass} \rrbracket(\sigma) = ([\text{pass}], \sigma)$

$trans_S \llbracket \text{break} \rrbracket(\sigma) = ([\text{break}], \sigma)$

$$trans_S \llbracket \text{continue} \rrbracket(\sigma) = ([\text{continue}], \sigma)$$

$$\boxed{trans_E : Expr \rightarrow \Sigma \rightarrow Expr}$$

$$trans_E \llbracket expr_1 \text{ boolop } expr_2 \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma) \text{ boolop } trans_E \llbracket expr_2 \rrbracket(\sigma)$$

$$trans_E \llbracket expr_1 := expr_2 \rrbracket(\sigma) = expr_1 := trans_E \llbracket expr_2 \rrbracket(\sigma)$$

$$trans_E \llbracket expr_1 \text{ binop } expr_2 \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma) \text{ binop } trans_E \llbracket expr_2 \rrbracket(\sigma)$$

$$trans_E \llbracket unop \text{ expr} \rrbracket(\sigma) = unop \ trans_E \llbracket expr \rrbracket(\sigma)$$

$$trans_E \llbracket \text{lambda } args : expr \rrbracket(\sigma) = \text{lambda } args : trans_E \llbracket expr \rrbracket(\sigma)$$

$$trans_E \llbracket expr_1 \text{ if } expr_2 \text{ else } expr_3 \rrbracket(\sigma) = \\ trans_E \llbracket expr_1 \rrbracket(\sigma) \text{ if } trans_E \llbracket expr_2 \rrbracket(\sigma) \text{ else } trans_E \llbracket expr_3 \rrbracket(\sigma)$$

$$trans_E \llbracket expr_1 \text{ if } expr_2 \text{ else } expr_3 \rrbracket(\sigma) = \\ trans_E \llbracket expr_1 \rrbracket(\sigma) \text{ if } trans_E \llbracket expr_2 \rrbracket(\sigma) \text{ else } trans_E \llbracket expr_3 \rrbracket(\sigma)$$

$$trans_E \llbracket (expr_1 : expr_2)^* (**expr_3)^* \rrbracket(\sigma) = \{(expr_1 : trans_E \llbracket expr_2 \rrbracket(\sigma))^* (**expr_3)^*\}$$

$$trans_E \llbracket \{expr^*\} \rrbracket(\sigma) = \{(trans_E \llbracket expr \rrbracket(\sigma))^*\}$$

$$trans_E \llbracket [expr^*] \rrbracket(\sigma) = [(trans_E \llbracket expr \rrbracket(\sigma))^*]$$

$$trans_E \llbracket (expr^*) \rrbracket(\sigma) = ((trans_E \llbracket expr \rrbracket(\sigma))^*)$$

$$trans_E \llbracket \{expr_1 : expr_2 \text{ comprehension}^*\} \rrbracket(\sigma) = \{expr_1 : trans_E \llbracket expr_2 \rrbracket(\sigma) \ (trans_O \llbracket comprehension \rrbracket(\sigma))^*\}$$

$$trans_E \llbracket \{expr \text{ comprehension}^*\} \rrbracket(\sigma) = \{trans_E \llbracket expr \rrbracket(\sigma) \ (trans_O \llbracket comprehension \rrbracket(\sigma))^*\}$$

$$trans_E \llbracket [expr \text{ comprehension}^*] \rrbracket(\sigma) = [(trans_E \llbracket expr \rrbracket(\sigma) \ (trans_O \llbracket comprehension \rrbracket(\sigma))^*)]$$

$$trans_E \llbracket (expr \text{ comprehension}^*) \rrbracket(\sigma) = (trans_E \llbracket expr \rrbracket(\sigma) \ (trans_O \llbracket comprehension \rrbracket(\sigma))^*)$$

$$trans_E \llbracket \text{await } expr \rrbracket(\sigma) = \text{await } trans_E \llbracket expr \rrbracket(\sigma)$$

$$trans_E \llbracket \text{yield } expr? \rrbracket(\sigma) = \text{yield } (trans_E \llbracket expr \rrbracket(\sigma))?$$

$$trans_E \llbracket \text{yield from } expr \rrbracket(\sigma) = \text{yield from } trans_E \llbracket expr \rrbracket(\sigma)$$

$$trans_E \llbracket expr_1 (compop expr_2)^* \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma) (compop trans_E \llbracket expr_2 \rrbracket(\sigma))^*$$

$$trans_E \llbracket expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) \rrbracket(\sigma) =$$

*# Adjust Steps*

**IF**  $\sigma(\text{"dataset"}) = id_t$  **AND**  $expr_1 = id_t.\text{take}$  **THEN**

**IF**  $id_i = \text{count}$  **WHEN**  $1 \leq i \leq k$  **THEN**

$expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots id_i = expr_{2i} // \text{hvd.size}() \dots (id_k = )? expr_{2k})$

**ELSE**

$expr_1 (expr_{11} // \text{hvd.size}() \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k})$

**ELSE**

$trans_E \llbracket expr_1 \rrbracket(\sigma) (trans_E \llbracket expr_{11} \rrbracket(\sigma) \dots trans_E \llbracket expr_{1n} \rrbracket(\sigma)$

$(id_1 = )? trans_E \llbracket expr_{21} \rrbracket(\sigma) \dots (id_k = )? trans_E \llbracket expr_{2k} \rrbracket(\sigma))$

$$trans_E \llbracket \{expr_1 \ (!i)? \ (:expr_2)?\} \rrbracket(\sigma) = \{expr_1 \ (!i)? \ (:expr_2)?\}$$

$$trans_E \llbracket expr^* \rrbracket(\sigma) = expr^*$$

$$trans_E \llbracket \text{constant} \rrbracket(\sigma) = \text{constant}$$

$$trans_E \llbracket expr.id \rrbracket(\sigma) = (trans_E \llbracket expr \rrbracket(\sigma)).id$$

$$trans_E \llbracket expr_1[expr_2] \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma)[trans_E \llbracket expr_2 \rrbracket(\sigma)]$$

$$trans_E \llbracket *expr \rrbracket(\sigma) = *expr$$

$$trans_E \llbracket **expr \rrbracket(\sigma) = **expr$$

$$trans_E \llbracket id \rrbracket(\sigma) = id$$

$$trans_E \llbracket expr_1? (:expr_2)? (:expr_3)? \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma)? (:trans_E \llbracket expr_2 \rrbracket(\sigma))? (:trans_E \llbracket expr_3 \rrbracket(\sigma))?$$

$$\boxed{trans_O : \text{Comprehension} \rightarrow \Sigma \rightarrow \text{Comprehension}}$$

$$trans_O \llbracket \text{for } expr_1 \text{ in } expr_2 \text{ (if } expr_3)^* \rrbracket(\sigma) = \text{for } expr_1 \text{ in } trans_E \llbracket expr_2 \rrbracket(\sigma) \text{ (if } trans_E \llbracket expr_3 \rrbracket(\sigma))^*$$

$$\boxed{trans_H : \text{ExcHandler} \rightarrow \Sigma \rightarrow \text{ExcHandler}}$$

$$trans_H \llbracket \text{except } expr? \text{ (as } id)? : stmt^* \rrbracket(\sigma) = \text{except } expr? \text{ (as } id)? : trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).\_1$$

$$\boxed{trans_{\bar{A}} : \text{Alias list} \rightarrow \Sigma \rightarrow \Sigma}$$

$$trans_{\overline{A}}[alias_1 \dots alias_{n-1} alias_n](\sigma) = trans_A[alias_n](trans_A[alias_{n-1}](\dots trans_A[alias_1](\sigma)))$$

$$trans_A : Alias \rightarrow \Sigma \rightarrow \Sigma$$

$$trans_A[id](\sigma) =$$

```

LET  $id = \text{tensorflow}$  THEN  $\sigma["\text{tensorflow}"] \mapsto id$ 
LET  $id = \text{os}$  THEN  $\sigma["\text{os}"] \mapsto id$ 
LET  $id = \text{config}$  THEN  $\sigma["\text{config}"] \mapsto id$ 
ELSE  $\sigma$ 

```

$$trans_A[id_1 \text{ as } id_2](\sigma) =$$

```

LET  $id_1 = \text{tensorflow}$  THEN  $\sigma["\text{tensorflow}"] \mapsto id_2$ 
LET  $id_1 = \text{os}$  THEN  $\sigma["\text{os}"] \mapsto id_2$ 
LET  $id = \text{config}$  THEN  $\sigma["\text{config}"] \mapsto id_2$ 
ELSE  $\sigma$ 

```

$$trans_A[id_1 . id_2 (.id_3)^* (\text{as } id_2)?](\sigma) = \sigma$$

$$trans_{\overline{W}} : WithItem \text{ list} \rightarrow \Sigma \rightarrow (WithItem \text{ list} \times \Sigma)$$

$$trans_{\overline{W}}[with\_item_1 with\_item_2 \dots with\_item_n](\sigma) =$$

```

LET  $with\_item_1', \sigma_1 = trans_W[with\_item_1](\sigma)$  IN
LET  $with\_item_2', \sigma_2 = trans_W[with\_item_2](\sigma_1)$  IN
...
LET  $with\_item_n', \sigma_n = trans_W[with\_item_n](\sigma_{n-1})$  IN
 $(with\_item_1' :: with\_item_2' :: \dots :: [with\_item_n'], \sigma_n)$ 

```

$$trans_W : WithItem \rightarrow \Sigma \rightarrow (WithItem \times \Sigma)$$

$$trans_W[expr](\sigma) = (trans_E[expr](\sigma), \sigma)$$

$$trans_W[expr_1 \text{ as } expr_2](\text{smodenv}) =$$

```

IF  $\sigma["\text{tensorflow}"] = id_1$  AND  $expr_1 = id_1.\text{GradientTape}()$  AND  $expr_2 = id_2$  THEN
   $(expr_1 \text{ as } expr_2, \sigma["\text{gradient\_tape"}] \mapsto id_2)$ 
ELSE  $(trans_E[expr_1](\sigma) \text{ as } expr_2, \sigma)$ 

```

$$trans_C : MatchCase \rightarrow \Sigma \rightarrow MatchCase$$

$$trans_C[\text{case pattern (if expr)? : stmt*}](\sigma) =$$

```

  case  $trans_P[pattern](\sigma)$  (if  $trans_E[expr](\sigma)$ )? :  $trans_{\overline{S}}[stmt*](\sigma).1$ 

```

$$trans_P : Pattern \rightarrow \Sigma \rightarrow Pattern$$

$$trans_P[expr](\sigma) = trans_E[expr](\sigma)$$

$$trans_P[constant](\sigma) = constant$$

$$trans_P[[pattern^*]](\sigma) = [trans_P[pattern](\sigma)^*]$$

$$trans_P \llbracket \star(id)? \rrbracket(\sigma) = \star(id)?$$

$$trans_P \llbracket \{(expr : pattern)^* id?\} \rrbracket(\sigma) = \{(expr : trans_P \llbracket pattern \rrbracket(\sigma))^* id?\}$$

$$trans_P \llbracket expr \ (pattern_1^* (id = pattern_2)^*) \rrbracket(\sigma) = expr \ (trans_P \llbracket pattern_1 \rrbracket(\sigma)^* (id = trans_P \llbracket pattern_2 \rrbracket(\sigma))^*)$$

$$trans_P \llbracket (pattern \ \mathbf{as})? id \rrbracket(\sigma) = (trans_P \llbracket pattern \rrbracket(\sigma) \ \mathbf{as})? id$$

$$trans_P \llbracket pattern_1 \mid pattern_2 \rrbracket(\sigma) = trans_P \llbracket pattern_1 \rrbracket(\sigma) \mid trans_P \llbracket pattern_2 \rrbracket(\sigma)$$

$$trans_P \llbracket - \rrbracket(\sigma) = -$$

## 2.5 Rule: TensorFlow 1.x Manuscript API Pattern

### 2.5.1 Transformation Rules

$trans_M : Module \rightarrow Module$
---------------------------------------

$$trans_M \llbracket stmt^* \ type\_ignore \rrbracket = trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1 \ type\_ignore$$

$trans_{\bar{S}} : Stmt \ list \rightarrow \Sigma \rightarrow (Stmt \ list \times \Sigma)$
--

$$trans_{\bar{S}} \llbracket stmt_1 \ stmt_2 \ \dots \ stmt_n \rrbracket(\sigma) = \begin{array}{l} \mathbf{LET} \ stmt_1^{*'}, \sigma_1 = trans_S \llbracket stmt_1 \rrbracket(\sigma) \ \mathbf{IN} \\ \mathbf{LET} \ stmt_2^{*'}, \sigma_2 = trans_S \llbracket stmt_2 \rrbracket(\sigma_1) \ \mathbf{IN} \\ \dots \\ \mathbf{LET} \ stmt_n^{*'}, \sigma_n = trans_S \llbracket stmt_n \rrbracket(\sigma_{n-1}) \ \mathbf{IN} \\ (stmt_1^{*'} @ stmt_2^{*'} @ \dots @ stmt_n^{*'}, \sigma_n) \end{array}$$

$trans_S : Stmt \rightarrow \Sigma \rightarrow (Stmt \ list \times \Sigma)$
---

$$trans_S \llbracket (@expr_1)^* \ \mathbf{def} \ id \ (args) \ (-> \ expr_2)? : (\#type:s)? \ stmt^* \rrbracket(\sigma) =$$

$$((@expr_1)^* \ \mathbf{def} \ id \ (args) \ (-> \ expr_2)? : (\#type:s)? \ trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1, \sigma)$$

$$trans_S \llbracket (@expr_1)^* \ \mathbf{async} \ \mathbf{def} \ id \ (args) \ (-> \ expr_2)? : (\#type:s)? \ stmt^* \rrbracket(\sigma) =$$

$$((@expr_1)^* \ \mathbf{async} \ \mathbf{def} \ id \ (args) \ (-> \ expr_2)? : (\#type:s)? \ trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1, \sigma)$$

$$trans_S \llbracket (@expr_1)^* \ \mathbf{class} \ id \ (expr_2^* \ keyword^*) : stmt^* \rrbracket(\sigma) =$$

$$((@expr_1)^* \ \mathbf{class} \ id \ (expr_2^* \ keyword^*) : trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1, \sigma) =$$

$$trans_S \llbracket \mathbf{return} \ expr? \rrbracket(\sigma) = ([\mathbf{return} \ (trans_E \llbracket expr \rrbracket(\sigma))?], \sigma)$$

$$trans_S \llbracket \mathbf{delete} \ expr^* \rrbracket(\sigma) = ([\mathbf{delete} \ expr^*], \sigma)$$

## A strict form of assignment statements

```

transS[[ idr = expr1 (expr11 ... expr1n (id1 = )? expr21 ... (idk = )? expr2k) (#type:s)? ]] (σ) =
  # Config Proto
  IF idt = "tensor_flow" AND expr1 = idt.ConfigProto THEN
    (idr = expr1 (expr11 ... expr1n (id1 = )? expr21 ... (idk = )? expr2k) (#type:s)?,
      σ[idr ↦ "config_proto"])
  # Minimize
  # Learning rate scheduler scaling
  IF expr1 <: tensorflow.keras.optimizers.schedules.PiecewiseConstantDecay THEN
    ([idr = expr1 (expr11 ... expr1n (id1 = )? expr21 ... (idk = )? expr2k) (#type:s)?],
      σ["lr_scheduler" ↦ idr])
  ELIF expr1 <: tensorflow.keras.optimizers.schedules.LearningRateSchedule THEN
    IF idi = initial_learning_rate WHEN 1 ≤ i ≤ k THEN
      ([idr = expr1 (expr11 ... expr1n (id1 = )? expr21 ... idi = expr2i * hvd.size()
        ... (idk = )? expr2k) (#type:s)?], σ["lr_scheduler" ↦ idr])
    ELSE
      ([idr = expr1 (expr11 * hvd.size() ... expr1n (id1 = )? expr21 ... (idk = )? expr2k) (#type:s)?],
        σ["lr_scheduler" ↦ idr])
  ELIF expr1 <: tensorflow.compat.v1.train.exponential_decay THEN
    IF idi = learning_rate WHEN 1 ≤ i ≤ k THEN
      ([idr = expr1 (expr11 ... expr1n (id1 = )? expr21 ... idi = expr2i * hvd.size()
        ... (idk = )? expr2k) (#type:s)?], σ["lr_scheduler" ↦ idr])
    ELSE
      ([idr = expr1 (expr11 * hvd.size() ... expr1n (id1 = )? expr21 ... (idk = )? expr2k) (#type:s)?],
        σ["lr_scheduler" ↦ idr])
  # Optimizer learning rate scaling and wrapping
  ELIF expr1 <: tensorflow.keras.optimizers.Optimizer THEN
    IF idi = learning_rate WHEN 1 ≤ i ≤ k THEN
      ([idr = expr1 (expr11 ... expr1n (id1 = )? expr21 ... idi = expr2i * hvd.size()
        ... (idk = )? expr2k) (#type:s)?],
        idr = hvd.DistributedOptimizer(idr), σ["optimizer" ↦ idr])
    ELSE
      ([idr = expr1 (expr11 * hvd.size() ... expr1n (id1 = )? expr21 ... (idk = )? expr2k) (#type:s)?],
        idr = hvd.DistributedOptimizer(idr), σ["optimizer" ↦ idr])
  # Model related
  ELIF expr1 <: tensorflow.keras.Model THEN
    ([idr = expr1 (expr11 ... expr1n (id2 = )? expr21 ... (idk = )? expr2k)], σ["model" ↦ idr])
  ELIF idm = "model" AND expr1 = idt.evaluate THEN
    IF idi = verbose WHEN 1 ≤ i ≤ k THEN
      ([idr = expr1 (expr11 ... expr1n (id1 = )? expr21 ... idi = 1 if hvd.rank() == 0 else 0
        ... (idk = )? expr2k) (#type:s)?], σ)
    ELSE
      ([idr = expr1 (expr11 expr12 1 if hvd.rank() == 0 else 0 ... expr1n
        (id1 = )? expr21 ... (idk = )? expr2k) (#type:s)?], σ)

```

*# Checkpoint*

**ELIF**  $\sigma(\text{"tensor\_flow"}) = id_t$  **AND**  $expr_1 = id_t.train.Checkpoint$  **THEN**  
 $([id_r = expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) \ (\#type:s)?],$   
 $\sigma[\text{"checkpoint"} \mapsto id_r])$

*# Default*

**ELSE**  $([id_r = trans_E \llbracket expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) \rrbracket(\sigma) \ (\#type:s)?],$   
 $\sigma)$

$trans_S \llbracket expr_0 = expr_1 \ (\#type:s)? \rrbracket(\sigma) =$

**IF**  $\sigma(\text{"os"}) = id_{os}$  **AND**  $expr_0 = id_{os}.environ \ [\text{'CUDA\_VISIBLE\_DEVICES'}]$   
**THEN**  $([], \sigma)$

**A general form of assignment statements except for the strict form**

$trans_S \llbracket expr_1^* = expr_2 \ (\#type:s)? \rrbracket(\sigma) = ([expr_1^* = trans_E \llbracket expr_2 \rrbracket(\sigma) \ (\#type:s)?], \sigma)$

$trans_S \llbracket expr_1 \ binop = expr_2 \rrbracket(\sigma) = ([expr_1 \ binop = trans_E \llbracket expr_2 \rrbracket(\sigma)], \sigma)$

$trans_S \llbracket expr_1 : expr_2 \ (= \ expr_3)? \rrbracket(\sigma) = ([expr_1 : expr_2 \ (= \ trans_E \llbracket expr_3 \rrbracket(\sigma))], \sigma)$

$trans_S \llbracket (\#type:s)? \ \text{for } expr_1 \ \text{in } expr_2 : stmt_1^* \ (\text{else} : stmt_2^*)? \rrbracket(\sigma) =$   
 $([(\#type:s)? \ \text{for } expr_1 \ \text{in } trans_E \llbracket expr_2 \rrbracket(\sigma) :$   
 $trans_{\bar{S}} \llbracket stmt_1^* \rrbracket(\sigma).1 \ (\text{else} : trans_{\bar{S}} \llbracket stmt_2^* \rrbracket(\sigma).1)?], \sigma)$

$trans_S \llbracket (\#type:s)? \ \text{async for } expr_1 \ \text{in } expr_2 : stmt_1^* \ (\text{else} : stmt_2^*)? \rrbracket(\sigma) =$   
 $([(\#type:s)? \ \text{async for } expr_1 \ \text{in } trans_E \llbracket expr_2 \rrbracket(\sigma) :$   
 $trans_{\bar{S}} \llbracket stmt_1^* \rrbracket(\sigma).1 \ (\text{else} : trans_{\bar{S}} \llbracket stmt_2^* \rrbracket(\sigma).1)?], \sigma)$

$trans_S \llbracket \text{while } (expr) : stmt_1^* \ (\text{else} : stmt_2^*)? \rrbracket(\sigma) =$   
 $([\text{while } (trans_E \llbracket expr \rrbracket(\sigma)) : trans_{\bar{S}} \llbracket stmt_1^* \rrbracket(\sigma).1 \ (\text{else} : trans_{\bar{S}} \llbracket stmt_2^* \rrbracket(\sigma).1)?], \sigma)$

$trans_S \llbracket \text{if } (expr) : stmt^* \ (\text{else} : stmt^*)? \rrbracket(\sigma) =$   
 $([\text{if } (trans_E \llbracket expr \rrbracket(\sigma)) : trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1 \ (\text{else} : trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1)?], \sigma)$

$trans_S \llbracket (\#type:s)? \ \text{with } with\_item^* : stmt^* \rrbracket(\sigma) = ([(\#type:s)? \ \text{with } with\_item^{*'} : stmt^{*'}], \sigma_2)$

$trans_S \llbracket (\#type:s)? \ \text{async with } with\_item^* : stmt^* \rrbracket(\sigma) =$   
 $([(\#type:s)? \ \text{async with } with\_item^{*'} : stmt^{*'}], \sigma_2)$

$trans_S \llbracket \text{match } expr : match\_case^* \rrbracket(\sigma) =$   
 $([\text{match } trans_E \llbracket expr \rrbracket(\sigma) : (trans_C \llbracket match\_case \rrbracket(\sigma))^*], \sigma)$

$trans_S \llbracket \text{raise } expr_1? \ (\text{from } expr_2)? \rrbracket(\sigma) = ([\text{raise } expr_1? \ (\text{from } expr_2)?], \sigma)$



$$\begin{aligned} trans_S \llbracket \text{try} : stmt_1^* \text{ exc\_handler}^* (\text{else} : stmt_2^*)? (\text{finally} : stmt_3^*)? \rrbracket(\sigma) = \\ ([\text{try} : trans_{\bar{S}} \llbracket stmt_1^* \rrbracket(\sigma).1 \ (trans_H \llbracket \text{exc\_handler} \rrbracket(\sigma))^* \\ (\text{else} : trans_{\bar{S}} \llbracket stmt_2^* \rrbracket(\sigma).1)? \ (\text{finally} : trans_{\bar{S}} \llbracket stmt_3^* \rrbracket(\sigma).1)?], \sigma) \end{aligned}$$

$$trans_S \llbracket \text{assert } expr_1 \ expr_2? \rrbracket(\sigma) = ([\text{assert } trans_E \llbracket expr_1 \rrbracket(\sigma) \ expr_2?], \sigma)$$

$$\begin{aligned} trans_S \llbracket \text{import } alias^* \rrbracket(\sigma) = \\ ([\text{import } alias^*], \sigma_1) \end{aligned}$$

$$trans_S \llbracket \text{from } i \ id? \text{ import } alias^* \rrbracket(\sigma) = ([\text{from } i \ id? \text{ import } alias^*], \sigma_1)$$

$$trans_S \llbracket \text{global } id^* \rrbracket(\sigma) = ([\text{global } id^*], \sigma)$$

$$trans_S \llbracket \text{nonlocal } id^* \rrbracket(\sigma) = ([\text{nonlocal } id^*], \sigma)$$

#### A strict form of expr statements

$$\begin{aligned} trans_S \llbracket expr_1 \ (expr_{11} \dots expr_{1n} \ (id_1 = )? \ expr_{21} \dots (id_k = )? \ expr_{2k}) \rrbracket(\sigma) = \\ \text{\textcolor{blue}{\# Config}} \\ \text{IF } id_t = \sigma(\text{“tensor\_flow”}) \ \text{AND} \\ \quad expr_1 = id_t.config.experimental.set\_visible\_devices \ \text{THEN } ([], \sigma) \\ \text{\textcolor{blue}{\# Root Rank Blocking}} \\ \text{ELIF } id_m = \sigma(\text{“model”}) \ \text{AND } (expr_1 = id_m.write \ \text{OR} \\ \quad expr_1 = id_m.summary \ \text{OR } expr_1 = id_m.save\_weights \ \text{OR} \\ \quad expr_1 = expr.load\_weights \ \text{OR } expr_1 = id_m.save) \ \text{OR} \\ \quad id_c = \sigma(\text{“checkpoint”}) \ \text{AND } expr_1 = id_c.save \ \text{OR} \\ \quad id_t = \sigma(\text{“tensor\_flow”}) \ \text{AND } expr_1 = id_t.print \ \text{OR} \\ \quad expr_1 = \text{print} \ \text{THEN} \\ \quad ([\text{if hvd.rank()} == 0: \ expr_1 \ (expr_{11} \dots expr_{1n} \ (id_1 = )? \ expr_{21} \dots (id_k = )? \ expr_{2k})], \sigma) \\ \text{ELSE} \\ \quad [trans_E \llbracket expr_1 \ (expr_{11} \dots expr_{1n} \ (id_1 = )? \ expr_{21} \dots (id_k = )? \ expr_{2k}) \rrbracket(\sigma), \sigma] \end{aligned}$$

#### A general form of expr statements except for the strict form

$$trans_S \llbracket expr \rrbracket(\sigma) = ([trans_E \llbracket expr \rrbracket(\sigma)], \sigma)$$

$$trans_S \llbracket \text{pass} \rrbracket(\sigma) = ([\text{pass}], \sigma)$$

$$trans_S \llbracket \text{break} \rrbracket(\sigma) = ([\text{break}], \sigma)$$

$$trans_S \llbracket \text{continue} \rrbracket(\sigma) = ([\text{continue}], \sigma)$$

$$trans_{\bar{E}} \llbracket expr_1 \ expr_2 \dots expr_n \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma) \ trans_E \llbracket expr_2 \rrbracket(\sigma) \dots trans_E \llbracket expr_n \rrbracket(\sigma)$$

$trans_E : Expr \rightarrow \Sigma \rightarrow Expr$
--

$$\text{trans}_E \llbracket \text{expr}_1 \text{ boolop } \text{expr}_2 \rrbracket(\sigma) = \text{trans}_E \llbracket \text{expr}_1 \rrbracket(\sigma) \text{ boolop } \text{trans}_E \llbracket \text{expr}_2 \rrbracket(\sigma)$$

$$\text{trans}_E \llbracket \text{expr}_1 \text{ := } \text{expr}_2 \rrbracket(\sigma) = \text{expr}_1 \text{ := } \text{trans}_E \llbracket \text{expr}_2 \rrbracket(\sigma)$$

$$\text{trans}_E \llbracket \text{expr}_1 \text{ binop } \text{expr}_2 \rrbracket(\sigma) = \text{trans}_E \llbracket \text{expr}_1 \rrbracket(\sigma) \text{ binop } \text{trans}_E \llbracket \text{expr}_2 \rrbracket(\sigma)$$

$$\text{trans}_E \llbracket \text{unop } \text{expr} \rrbracket(\sigma) = \text{unop } \text{trans}_E \llbracket \text{expr} \rrbracket(\sigma)$$

$$\text{trans}_E \llbracket \text{lambda } \text{args} : \text{expr} \rrbracket(\sigma) = \text{lambda } \text{args} : \text{trans}_E \llbracket \text{expr} \rrbracket(\sigma)$$

$$\begin{aligned} \text{trans}_E \llbracket \text{expr}_1 \text{ if } \text{expr}_2 \text{ else } \text{expr}_3 \rrbracket(\sigma) = \\ \text{trans}_E \llbracket \text{expr}_1 \rrbracket(\sigma) \text{ if } \text{trans}_E \llbracket \text{expr}_2 \rrbracket(\sigma) \text{ else } \text{trans}_E \llbracket \text{expr}_3 \rrbracket(\sigma) \end{aligned}$$

$$\begin{aligned} \text{trans}_E \llbracket \text{expr}_1 \text{ if } \text{expr}_2 \text{ else } \text{expr}_3 \rrbracket(\sigma) = \\ \text{trans}_E \llbracket \text{expr}_1 \rrbracket(\sigma) \text{ if } \text{trans}_E \llbracket \text{expr}_2 \rrbracket(\sigma) \text{ else } \text{trans}_E \llbracket \text{expr}_3 \rrbracket(\sigma) \end{aligned}$$

$$\text{trans}_E \llbracket \{(\text{expr}_1 : \text{expr}_2)^* (**\text{expr}_3)^*\} \rrbracket(\sigma) = \{(\text{expr}_1 : \text{trans}_E \llbracket \text{expr}_2 \rrbracket(\sigma))^* (**\text{expr}_3)^*\}$$

$$\text{trans}_E \llbracket \{\text{expr}^*\} \rrbracket(\sigma) = \{(\text{trans}_E \llbracket \text{expr} \rrbracket(\sigma))^*\}$$

$$\text{trans}_E \llbracket [\text{expr}^*] \rrbracket(\sigma) = [(\text{trans}_E \llbracket \text{expr} \rrbracket(\sigma))^*]$$

$$\text{trans}_E \llbracket (\text{expr}^*) \rrbracket(\sigma) = ((\text{trans}_E \llbracket \text{expr} \rrbracket(\sigma))^*)$$

$$\text{trans}_E \llbracket \{\text{expr}_1 : \text{expr}_2 \text{ comprehension}^*\} \rrbracket(\sigma) = \{\text{expr}_1 : \text{trans}_E \llbracket \text{expr}_2 \rrbracket(\sigma) (\text{trans}_O \llbracket \text{comprehension} \rrbracket(\sigma))^*\}$$

$$\text{trans}_E \llbracket \{\text{expr} \text{ comprehension}^*\} \rrbracket(\sigma) = \{\text{trans}_E \llbracket \text{expr} \rrbracket(\sigma) (\text{trans}_O \llbracket \text{comprehension} \rrbracket(\sigma))^*\}$$

$$\text{trans}_E \llbracket [\text{expr} \text{ comprehension}^*] \rrbracket(\sigma) = [(\text{trans}_E \llbracket \text{expr} \rrbracket(\sigma) (\text{trans}_O \llbracket \text{comprehension} \rrbracket(\sigma))^*)]$$

$$\text{trans}_E \llbracket (\text{expr} \text{ comprehension}^*) \rrbracket(\sigma) = (\text{trans}_E \llbracket \text{expr} \rrbracket(\sigma) (\text{trans}_O \llbracket \text{comprehension} \rrbracket(\sigma))^*)$$

$$\text{trans}_E \llbracket \text{await } \text{expr} \rrbracket(\sigma) = \text{await } \text{trans}_E \llbracket \text{expr} \rrbracket(\sigma)$$

$$\text{trans}_E \llbracket \text{yield } \text{expr}? \rrbracket(\sigma) = \text{yield } (\text{trans}_E \llbracket \text{expr} \rrbracket(\sigma))?$$

$$\text{trans}_E \llbracket \text{yield from } \text{expr} \rrbracket(\sigma) = \text{yield from } \text{trans}_E \llbracket \text{expr} \rrbracket(\sigma)$$

$$\text{trans}_E \llbracket \text{expr}_1 (\text{compop } \text{expr}_2)^* \rrbracket(\sigma) = \text{trans}_E \llbracket \text{expr}_1 \rrbracket(\sigma) (\text{compop } \text{trans}_E \llbracket \text{expr}_2 \rrbracket(\sigma))^*$$

$$\begin{aligned} trans_E \llbracket expr_1 (expr_{11} \dots expr_{1n} \ (id_1 = )? \ expr_{21} \dots (id_k = )? \ expr_{2k}) \rrbracket(\sigma) = \\ trans_E \llbracket expr_1 \rrbracket(\sigma) \ (trans_E \llbracket expr_{11} \rrbracket(\sigma) \dots trans_E \llbracket expr_{1n} \rrbracket(\sigma) \\ (id_1 = )? \ trans_E \llbracket expr_{21} \rrbracket(\sigma) \dots (id_k = )? \ trans_E \llbracket expr_{2k} \rrbracket(\sigma)) \end{aligned}$$

$$trans_E \llbracket \{expr_1 \quad (!i)? \quad (:expr_2)?\} \rrbracket(\sigma) = \{expr_1 \quad (!i)? \quad (:expr_2)?\}$$

$$trans_E \llbracket expr^* \rrbracket(\sigma) = expr^*$$

$$trans_E \llbracket constant \rrbracket(\sigma) = constant$$

$$trans_E \llbracket expr.id \rrbracket(\sigma) = (trans_E \llbracket expr \rrbracket(\sigma)).id$$

$$trans_E \llbracket expr_1[expr_2] \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma)[trans_E \llbracket expr_2 \rrbracket(\sigma)]$$

$$trans_E \llbracket *expr \rrbracket(\sigma) = *expr$$

$$trans_E \llbracket **expr \rrbracket(\sigma) = **expr$$

$$trans_E \llbracket id \rrbracket(\sigma) = id$$

$$trans_E \llbracket expr_1? \ (:expr_2)? \ (:expr_3)? \rrbracket(\sigma) = trans_E \llbracket expr_1 \rrbracket(\sigma)? \ (:trans_E \llbracket expr_2 \rrbracket(\sigma))? \ (:trans_E \llbracket expr_3 \rrbracket(\sigma))?$$

$$\boxed{trans_O \quad : \quad Comprehension \rightarrow \Sigma \rightarrow Comprehension}$$

$$trans_O \llbracket \text{for } expr_1 \text{ in } expr_2 \text{ (if } expr_3)^* \rrbracket(\sigma) = \text{for } expr_1 \text{ in } trans_E \llbracket expr_2 \rrbracket(\sigma) \text{ (if } trans_E \llbracket expr_3 \rrbracket(\sigma))^*$$

$$\boxed{trans_H \quad : \quad ExcHandler \rightarrow \Sigma \rightarrow ExcHandler}$$

$$trans_H \llbracket \text{except } expr? \text{ (as } id)? \text{ : } stmt^* \rrbracket(\sigma) = \text{except } expr? \text{ (as } id)? \text{ : } trans_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma).1$$

$$\boxed{trans_{\bar{A}} \quad : \quad Alias \text{ list} \rightarrow \Sigma \rightarrow \Sigma}$$

$$trans_{\bar{A}} \llbracket alias_1 \dots alias_{n-1} alias_n \rrbracket(\sigma) = trans_A \llbracket alias_n \rrbracket(trans_A \llbracket alias_{n-1} \rrbracket(\dots trans_A \llbracket alias_1 \rrbracket(\sigma)))$$

$$\boxed{trans_A \quad : \quad Alias \rightarrow \Sigma \rightarrow \Sigma}$$

$$trans_A \llbracket id \rrbracket(\sigma) =$$

**LET**  $id = \text{tensorflow.compat.v1}$  **THEN**  $\sigma[\text{"tensorflow.compat"} \mapsto id]$

**LET**  $id = \text{tensorflow}$  **THEN**  $\sigma[\text{"tensorflow"} \mapsto id]$

**LET**  $id = \text{os}$  **THEN**  $\sigma[\text{"os"} \mapsto id]$

**ELSE**  $\sigma$

$trans_A \llbracket id_1 \text{ as } id_2 \rrbracket(\sigma) =$   
 $\text{LET } id_1 = \text{tensorflow.compat.v1} \text{ THEN } \sigma[\text{“tensorflow.compat”} \mapsto id_2]$   
 $\text{LET } id_1 = \text{tensorflow} \text{ THEN } \sigma[\text{“tensorflow”} \mapsto id_2]$   
 $\text{LET } id_1 = \text{os} \text{ THEN } \sigma[\text{“os”} \mapsto id_2]$   
 $\text{ELSE } \sigma$

$trans_A \llbracket id_1 . id_2 (.id_3)^* (\text{as } id_2)? \rrbracket(\sigma) = \sigma$

$trans_{\overline{W}} : WithItem \text{ list} \rightarrow \Sigma \rightarrow (WithItem \text{ list} \times \Sigma)$

$trans_{\overline{W}} \llbracket with\_item_1 with\_item_2 \dots with\_item_n \rrbracket(\sigma) =$   
 $\text{LET } with\_item_1', \sigma_1 = trans_W \llbracket with\_item_1 \rrbracket(\sigma) \text{ IN}$   
 $\text{LET } with\_item_2', \sigma_2 = trans_W \llbracket with\_item_2 \rrbracket(\sigma_1) \text{ IN}$   
 $\dots$   
 $\text{LET } with\_item_n', \sigma_n = trans_W \llbracket with\_item_n \rrbracket(\sigma_{n-1}) \text{ IN}$   
 $(with\_item_1' :: with\_item_2' :: \dots :: [with\_item_n'], \sigma_n)$

$trans_W : WithItem \rightarrow \Sigma \rightarrow (WithItem \times \Sigma)$

$trans_W \llbracket expr \rrbracket(\sigma) = (trans_E \llbracket expr \rrbracket(\sigma), \sigma)$

$trans_W \llbracket expr_1 \text{ as } expr_2 \rrbracket(\sigma) =$   
 $(trans_E \llbracket expr_1 \rrbracket(\sigma) \text{ as } expr_2, \sigma)$

$trans_C : MatchCase \rightarrow \Sigma \rightarrow MatchCase$

$trans_C \llbracket \text{case pattern (if expr)? : stmt*} \rrbracket(\sigma) =$   
 $\text{case } trans_P \llbracket pattern \rrbracket(\sigma) \text{ (if } trans_E \llbracket expr \rrbracket(\sigma))? : trans_{\overline{S}} \llbracket stmt* \rrbracket(\sigma).1$

$trans_P : Pattern \rightarrow \Sigma \rightarrow Pattern$

$trans_P \llbracket expr \rrbracket(\sigma) = trans_E \llbracket expr \rrbracket(\sigma)$

$trans_P \llbracket constant \rrbracket(\sigma) = constant$

$trans_P \llbracket [pattern^*] \rrbracket(\sigma) = [trans_P \llbracket pattern \rrbracket(\sigma)]^*$

$trans_P \llbracket *(id)? \rrbracket(\sigma) = *(id)?$

$trans_P \llbracket \{(expr : pattern)^* id?\} \rrbracket(\sigma) = \{(expr : trans_P \llbracket pattern \rrbracket(\sigma))^* id?\}$

$trans_P \llbracket expr (pattern_1^* (id = pattern_2)^*) \rrbracket(\sigma) = expr (trans_P \llbracket pattern_1 \rrbracket(\sigma)^* (id = trans_P \llbracket pattern_2 \rrbracket(\sigma))^*)$

$trans_P \llbracket (pattern \text{ as})? id \rrbracket(\sigma) = (trans_P \llbracket pattern \rrbracket(\sigma) \text{ as})? id$

$$trans_P \llbracket pattern_1 \mid pattern_2 \rrbracket(\sigma) = trans_P \llbracket pattern_1 \rrbracket(\sigma) \mid trans_P \llbracket pattern_2 \rrbracket(\sigma)$$

$$trans_P \llbracket - \rrbracket(\sigma) = -$$

### 3 Identifying trainig loop

#### 3.1 Restrictions

1. Training loop must be defined in only one file.
2. Training loop type is either `distributed gradient tape` or `distributed optimizer`.
3. Each model must have only one type of training loop.
4. Function must not be assigned to the variables or passed as an argument.
5. Training loop must not be defined conditionally.

#### 3.2 Rules

##### 3.2.1 Summary

$$\begin{aligned}
Summary & ::= \text{ModuleSummary } id \ \sigma_{\top} \ tl & (\text{MODULESUMMARY}) \\
& \quad \text{FuncSummary } tl & (\text{FUNCSUMMARY}) \\
& \quad \text{ClassSummary } arg & (\text{CLASSSUMMARY}) \\
& \quad \text{ValueSummary } s & (\text{VALUESUMMARY}) \\
arg & ::= Model \mid \perp & (\text{ARG}) \\
tl & ::= GradTape \mid Optimizer \mid \perp & (\text{TRAININGLOOP}) \\
\sigma_{\top} & ::= \sigma \cup \top
\end{aligned}$$

$$\sigma \in \Sigma = Id \xrightarrow{\text{fin}} Summary \quad \text{ENVIRONMENT STORING MAPPINGS FROM IDS TO SUMMARIES}$$

##### 3.2.2 Training Loop

$$\boxed{summary_M : Module \rightarrow \Sigma \rightarrow ModuleSummary}$$

$$\begin{aligned}
summary_M \llbracket id \ stmt^* \ type\_ignore \rrbracket = \\
\quad \mathbf{LET} \ \sigma, tl = summary_{\overline{S}} \llbracket stmt^* \rrbracket(\sigma) \ \mathbf{IN} \\
\quad \text{ModuleSummary } id \ \sigma \ tl
\end{aligned}$$

$$\boxed{summary_{\overline{S}} : Stmt \ \mathbf{list} \rightarrow \Sigma \rightarrow (\Sigma \times tl)}$$

$$\begin{aligned}
summary_{\overline{S}} \llbracket stmt_1 \ stmt_2 \dots stmt_n \rrbracket(\sigma) = \quad & \mathbf{LET} \ \sigma_1, tl_1 = summary_S \llbracket stmt_1 \rrbracket(\sigma) \ \mathbf{IN} \\
& \mathbf{LET} \ \sigma_2, tl_2 = summary_{\overline{S}} \llbracket stmt_2 \rrbracket(\sigma_1) \ \mathbf{IN} \\
& \dots \\
& \mathbf{LET} \ \sigma_n, tl_n = summary_{\overline{S}} \llbracket stmt_n \rrbracket(\sigma_{n-1}) \ \mathbf{IN} \\
& (\sigma_n, tl_1 \sqcup tl_2 \sqcup \dots tl_n)
\end{aligned}$$

$$\boxed{summary_S : Stmt \rightarrow \Sigma \rightarrow (\Sigma \times tl)}$$

$summary_S \llbracket (@expr_1)^* \text{ def } id (args) (-> expr_2)? : (\#type:s)? stmt^* \rrbracket(\sigma) =$   
 $\text{LET } \sigma', tl = summary_{\bar{S}} \llbracket stmt^* \rrbracket \text{ IN}$   
 $(\sigma[id \mapsto \text{FuncSummary } tl], \perp)$

$summary_S \llbracket (@expr_1)^* \text{ async def } id (args) (-> expr_2)? : (\#type:s)? stmt^* \rrbracket(\sigma) =$   
 $\text{LET } \sigma', tl = summary_{\bar{S}} \llbracket stmt^* \rrbracket \text{ IN}$   
 $(\sigma[id \mapsto \text{FuncSummary } tl], \perp)$

$summary_S \llbracket (@expr)^* \text{ class } id (expr_{11} \dots expr_{1n} (id_1 = )? expr_{21} \dots (id_k = )? expr_{2k}) : stmt^* \rrbracket(\sigma) =$   
 $\text{IF } expr_{1i} = id_c.keras.Model \text{ WHEN } 1 \leq i \leq n \text{ AND}$   
 $\sigma(id_c) = \text{ModuleSummary Id("tensorflow")} \sqcup \perp \text{ THEN } (\sigma[id \mapsto \text{ClassSummary } Model], \perp)$   
 $\text{ELIF } expr_{2i} = id_c.keras.Model \text{ WHEN } 1 \leq i \leq k \text{ AND}$   
 $\sigma(id_c) = \text{ModuleSummary Id("tensorflow")} \sqcup \perp \text{ THEN } (\sigma[id \mapsto \text{ClassSummary } Model], \perp)$   
 $\text{ELSE } (\sigma[id \mapsto \text{ClassSummary } \perp], \perp)$

#### A strict form of assignment statements

$summary_S \llbracket id_r = id_c () (\#type:s)? \rrbracket(\sigma) =$   
 $\text{IF } \sigma(id_c) = \text{ClassSummary } Model \text{ THEN } (\sigma[id_r \mapsto \text{ValueSummary "model.instance"}], \perp)$   
 $\text{ELSE } (\sigma, \perp)$

$summary_S \llbracket (\#type:s)? \text{ for } expr_1 \text{ in } expr_2 : stmt_1^* (\text{else} : stmt_2^*)? \rrbracket(\sigma) =$   
 $\text{LET } \_, tl = summary_{\bar{S}} \llbracket stmt_1^* \rrbracket(\sigma) \text{ IN}$   
 $(\sigma, tl)$

$summary_S \llbracket (\#type:s)? \text{ async for } expr_1 \text{ in } expr_2 : stmt_1^* (\text{else} : stmt_2^*)? \rrbracket(\sigma) =$   
 $\text{LET } \_, tl = summary_{\bar{S}} \llbracket stmt_1^* \rrbracket(\sigma) \text{ IN}$   
 $(\sigma, tl)$

$summary_S \llbracket \text{ while } (expr) : stmt_1^* (\text{else} : stmt_2^*)? \rrbracket(\sigma) =$   
 $\text{LET } \_, tl = summary_{\bar{S}} \llbracket stmt_1^* \rrbracket(\sigma) \text{ IN}$   
 $(\sigma, tl)$

$summary_S \llbracket (\#type:s)? \text{ with } with\_item^* : stmt^* \rrbracket(\sigma) =$   
 $\text{LET } \_, tl = summary_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma) \text{ IN}$   
 $(\sigma, summary_{\bar{W}} \llbracket with\_item^* \rrbracket(\sigma) \sqcup tl)$

$summary_S \llbracket (\#type:s)? \text{ async with } with\_item^* : stmt^* \rrbracket(\sigma) =$   
 $\text{LET } \_, tl = summary_{\bar{S}} \llbracket stmt^* \rrbracket(\sigma) \text{ IN}$   
 $(\sigma, summary_{\bar{W}} \llbracket with\_item^* \rrbracket(\sigma) \sqcup tl)$

$summary_S \llbracket \text{ try } : stmt_1^* \text{ exc\_handler}^* (\text{else} : stmt_2^*)? (\text{finally} : stmt_3^*)? \rrbracket(\sigma) =$   
 $(\sigma, summary_{\bar{S}} \llbracket stmt_1^* \rrbracket(\sigma))$

$summary_S \llbracket \text{ import } alias^* \rrbracket(\sigma) = (summary_{\bar{A}} \llbracket alias^* \rrbracket(\sigma), \perp)$

```

summaryS[[ from 0 id* import alias* ]](σ) =
  LET σ' = summaryA[[ alias* ]](σ) IN
  LET [id2 ↦ ModuleSummary id2 σ2 ⊥, id2 ↦ ModuleSummary id2 σ2 ⊥,
    ... idn ↦ ModuleSummary idn σn ⊥] = σ' \ σ IN
  σ ++ [id2 ↦ ModuleSummary (id*+id2) σ2 ⊥, id2 ↦ ModuleSummary (id*+id2) σ2 ⊥,
    ... idn ↦ ModuleSummary (id*+idn) σn ⊥]

```

```

summaryS[[ expr ]](σ) = (σ, summaryE[[ expr ]](σ))

```

```

summaryS[[ stmt ]](σ) = (σ, ⊥)

```

summary <sub>A</sub> : Alias list → Σ → Σ
---

```

summaryA[[ alias1 ... aliasn-1 aliasn ]](σ) =
  summaryA[[ aliasn ]](summaryA[[ aliasn-1 ]](... summaryA[[ alias1 ]](σ) ...))

```

trans <sub>A</sub> : Alias → Σ → Σ
------------------------------------

```

transA[[ id1 id2 ... idn ]](σ) =
  LET σn = [idn ↦ ModuleSummary idn ⊤ ⊥]
  LET σn-1 = [idn-1 ↦ ModuleSummary idn-1 σn ⊥]
  ...
  LET σ2 = [id2 ↦ ModuleSummary id2 σ3 ⊥]
  LET σ1 = [id1 ↦ ModuleSummary id1 σ2 ⊥]
  σ ++ σ1

```

summary <sub>W</sub> : WithItem list → Σ → tl
---

```

summaryW[[ with_item1 with_item2 ... with_itemn ]](σ) =
  summaryW[[ with_item1 ]](σ) ⊔ summaryW[[ with_item2 ]](σ) ⊔ ... summaryW[[ with_itemn ]](σ)

```

summary <sub>W</sub> : WithItem → Σ → tl
--

```

summaryW[[ expr1 as expr2? ]](σ) = summaryE[[ expr1 ]](σ)

```

summary <sub>E</sub> : Expr → Σ → tl
--------------------------------------

**A strict form of call expression**

```

summaryE[[ expr1 (expr11 ... expr1n (id1 = )? expr21 ... (idk = )? expr2k) ]](σ) =
  IF σ(idr) = ValueSummary "model_instance" AND
    expr1 = idr.fit THEN Optimizer
  ELIF σ(idr) = ModuleSummary Id("tensorflow") σ' tl AND
    expr1 = idr.GradientTape THEN GradTape
  ELIF σ(idr) = FuncSummary tl AND
    expr1 = idr THEN tl
  ELSE ⊥

```

```

summaryE[[ expr ]](σ) = ⊥

```