Code Transformation for Distributed Python ML Code

1   Python Abstract Syntax

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

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

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

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

$$
\begin{array}{lll}
\tau & ::= & Module & \text{(MODULES)} \\
& & Stmt & \text{(STATEMENTS)} \\
& & Expr & \text{(EXPRESSIONS)} \\
& & Comprehension & \text{(COMPREHENSIONS)} \\
& & ExcHandler & \text{(EXCEPTION HANDLERS)} \\
& & Alias & \text{(ALIASES)} \\
& & WithItem & \text{(WITHITEMS)} \\
& & Pattern & \text{(PATTERNS)} \\
& & Id & \text{(IDENTIFIERS)} \\
& & Str & \text{(STRINGS)} \\
& & \tau \text{ list} & \text{(LIST OF ELEMENTS TYPED } \tau)
\end{array}
$$

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

`._1`	:	$\tau \times \tau \to \tau$	GET THE FIRST ELEMENT OF THE GIVEN PAIR
`@`	:	$\tau \text{ list} \to \tau \text{ list} \to \tau \text{ list}$	CONCATENATE TWO LISTS
`::`	:	$\tau \to \tau \text{ list} \to \tau \text{ list}$	APPEND AN ELEMENT TO A LIST (RIGHT-ASSOCIATIVE)
`<:`	:	$Expr \times Expr \to \mathbb{B}$	WHETHER CLASSES SPECIFIED BY THE EXPRESSIONS ARE SUBCLASSES

2.3 Rule: TensorFlow 2.x Keras API Pattern

2.3.1 Transformation Rules

$$\boxed{trans_M \quad : \quad Module \quad \to \quad Module}$$

$$trans_M [\![\; stmt^* \quad type\_ignore \;]\!] \quad = \quad trans_{\overline{S}}[\![\; stmt^* \;]\!](\sigma).\_1 \;\; type\_ignore$$

$$\boxed{trans_{\overline{S}} \quad : \quad Stmt \;\; \textbf{list} \quad \to \quad \Sigma \quad \to \quad (Stmt \;\; \textbf{list} \quad \times \quad \Sigma)}$$

$$
\begin{aligned}
trans_{\overline{S}}[\![\; stmt_1 \quad stmt_2 \;...\; stmt_n \;]\!](\sigma) \quad = \quad & \textbf{LET} \;\; stmt_1^{*\prime}, \sigma_1 \;\; = \;\; trans_S[\![\; stmt_1 \;]\!](\sigma) \quad \textbf{IN} \\
& \textbf{LET} \;\; stmt_2^{*\prime}, \sigma_2 \;\; = \;\; trans_S[\![\; stmt_2 \;]\!](\sigma_1) \quad \textbf{IN} \\
& \cdots \\
& \textbf{LET} \;\; stmt_n^{*\prime}, \sigma_n \;\; = \;\; trans_S[\![\; stmt_n \;]\!](\sigma_{n-1}) \quad \textbf{IN} \\
& (stmt_1^{*\prime} \; @ \; stmt_2^{*\prime} \; @ \;...\; @ \; stmt_n^{*\prime}, \; \sigma_n)
\end{aligned}
$$

$$\boxed{trans_S \quad : \quad Stmt \quad \to \quad \Sigma \quad \to \quad (Stmt \;\; \textbf{list} \quad \times \quad \Sigma)}$$

$$trans_S[\![\; (@expr_1)^* \;\; \texttt{def} \;\; id \;\; (args) \;\; (\texttt{->} \; expr_2)? \quad : \;\; (\texttt{\#type:}s)? \;\; stmt^* \;]\!](\sigma) =$$
$$([(@expr_1)^* \;\; \texttt{def} \;\; id \;\; (args) \;\; (\texttt{->} \; expr_2)? \quad : \;\; (\texttt{\#type:}s)? \;\; trans_{\overline{S}}[\![\; stmt^* \;]\!](\sigma).\_1], \; \sigma)$$

$$trans_S[\![\; (@expr_1)^* \;\; \texttt{async def} \;\; id \;\; (args) \;\; (\texttt{->} \; expr_2)? \quad : \;\; (\texttt{\#type:}s)? \;\; stmt^* \;]\!](\sigma) =$$
$$([(@expr_1)^* \;\; \texttt{async def} \;\; id \;\; (args) \;\; (\texttt{->} \; expr_2)? \quad : \;\; (\texttt{\#type:}s)? \;\; trans_{\overline{S}}[\![\; stmt^* \;]\!](\sigma).\_1], \; \sigma)$$

$$trans_S[\![\; (@expr_1)^* \;\; \texttt{class} \;\; id \;\; (expr_2^* \; keyword^*) \quad : \;\; stmt^* \;]\!](\sigma) =$$
$$([(@expr_1)^* \;\; \texttt{class} \;\; id \;\; (expr_2^* \; keyword^*) \quad : \;\; trans_{\overline{S}}[\![\; stmt^* \;]\!](\sigma).\_1], \; \sigma) =$$

$$trans_S[\![\; \texttt{return} \;\; expr? \;]\!](\sigma) = ([\texttt{return} \;\; (trans_E[\![\; expr \;]\!](\sigma))?], \; \sigma)$$

$$trans_S[\![\; \texttt{delete} \;\; expr^* \;]\!](\sigma) = ([\texttt{delete} \;\; expr^*], \; \sigma)$$

A strict form of assignment statements

$$trans_S[\![\; id_r = expr_1 \; (expr_{11} \;...\; expr_{1n} \quad (id_1 =)? \; expr_{21} \;...\; (id_k =)? \; expr_{2k}) \; (\texttt{\#type:}s)? \;]\!](\sigma) =$$

    # Learning rate scheduler scaling

  IF $expr_1 <:$ `"tensorflow.keras.optimizers.schedules.PiecewiseConstantDecay"` **THEN**

    $([id_r = expr_1 \; (expr_{11} \;...\; expr_{1n} \quad (id_1 =)? \; expr_{21} \;...\; (id_k =)? \; expr_{2k}) \; (\texttt{\#type:}s)?],$

      $\sigma[\textbf{"lr\_scheduler"} \mapsto id_r])$

  ELIF $expr_1 <:$ `tensorflow.keras.optimizers.schedules.LearningRateSchedule` **THEN**

    IF $id_i = $ `initial_learning_rate` **WHEN** $1 \le i \le k$ **THEN**

      $([id_r = expr_1 \; (expr_{11} \;...\; expr_{1n} \quad (id_1 =)? \; expr_{21} \;...\; id_i = expr_{2i} *$ `hvd.size()`

        $...\; (id_k =)? \; expr_{2k}) \; (\texttt{\#type:}s)?], \; \sigma[\textbf{"lr\_scheduler"} \mapsto id_r])$

    ELSE

      $([id_r = expr_1 \; (expr_{11} *$ `hvd.size()` $...\; expr_{1n} \quad (id_1 =)? \; expr_{21} \;...\; (id_k =)? \; expr_{2k}) \; (\texttt{\#type:}s)?],$

        $\sigma[\textbf{"lr\_scheduler"} \mapsto id_r])$

  ELIF $expr_1 <:$ `tensorflow.compat.v1.train.exponential_decay` **THEN**

    IF $id_i = $ `learning_rate` **WHEN** $1 \le i \le k$ **THEN**

      $([id_r = expr_1 \; (expr_{11} \;...\; expr_{1n} \quad (id_1 =)? \; expr_{21} \;...\; id_i = expr_{2i} *$ `hvd.size()`

        $...\; (id_k =)? \; expr_{2k}) \; (\texttt{\#type:}s)?], \; \sigma[\textbf{"lr\_scheduler"} \mapsto id_r])$

ELSE

$([id_r = expr_1 \ (expr_{11} * \texttt{hvd.size()} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?],$
$\sigma[\textbf{``lr\_scheduler''}\mapsto id_r])$

 # Optimizer learning rate scaling and wrapping

ELIF $expr_1 <:$ `tensorflow.keras.optimizers.Optimizer` **THEN**

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

  $([id_r = expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ id_i = expr_{2i} * \texttt{hvd.size()}$
   $... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?$
  $id_r = \texttt{hvd.DistributedOptimizer}(id_r)], \ \sigma[\textbf{``optimizer''}\mapsto id_r])$

 ELSE

  $([id_r = expr_1 \ (expr_{11} * \texttt{hvd.size()} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?,$
  $id_r = \texttt{hvd.DistributedOptimizer}(id_r)], \ \sigma[\textbf{``optimizer''}\mapsto id_r])$

 # Model related

ELIF $expr_1 <:$ `tensorflow.keras.Model` **THEN**

 $([id_r = expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_2 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k})], \ \sigma[\textbf{``model''}\mapsto id_r])$

ELIF $id_m = $ **``model''** **AND** $expr_1 = id_t.\texttt{evaluate}$ **THEN**

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

  $([id_r = expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ id_i = \texttt{1 if hvd.rank() == 0 else 0}$
   $... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?], \ \sigma)$

 ELSE

  $([id_r = expr_1 \ (expr_{11} \quad expr_{12} \quad \texttt{1 if hvd.rank() == 0 else 0} \ ... \ expr_{1n}$
   $(id_1 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?], \ \sigma)$

 # Checkpoint

ELIF $\sigma(\textbf{``tensor\_flow''}) = id_t$ **AND** $expr_1 = id_t.\texttt{train.Checkpoint}$ **THEN**

 $([id_r = expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?],$
 $\sigma[\textbf{``checkpoint''}\mapsto id_r])$

 # Default

ELSE $([id_r = trans_E[\![\ expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_2 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k}) \ ]\!](\sigma) \ (\texttt{\#type:}s)?],$
 $\sigma)$

$trans_S[\![\ expr_0 = expr_1 \ (\texttt{\#type:}s)? \ ]\!](\sigma) =$
 IF $\sigma(\textbf{``os''}) = id_{os}$ **AND** $expr_0 = id_{os}.\texttt{environ ['CUDA\_VISIBLE\_DEVICES']}$
 THEN $([], \ \sigma)$

A general form of assignment statements except for the strict form
$trans_S[\![\ expr_1^* = expr_2 \ (\texttt{\#type:}s)? \ ]\!](\sigma) = ([expr_1^* = trans_E[\![\ expr_2 \ ]\!](\sigma) \ (\texttt{\#type:}s)?], \ \sigma)$

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

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

$trans_S[\![\ (\texttt{\#type:}s)? \ \texttt{for} \ expr_1 \ \texttt{in} \ expr_2 \ : \ stmt_1^* \ (\texttt{else} : \ stmt_2^*)? \ ]\!](\sigma) =$
 $([(\texttt{\#type:}s)? \ \texttt{for} \ expr_1 \ \texttt{in} \ trans_E[\![\ expr_2 \ ]\!](\sigma) \ :$
  $trans_{\overline{S}}[\![\ stmt_1^* \ ]\!](\sigma).\_1 \ (\texttt{else} : \ trans_{\overline{S}}[\![\ stmt_2^* \ ]\!](\sigma).\_1)?], \ \sigma)$

6

$trans_S[\![$ (#type:s)? async for $expr_1$ in $expr_2$: $stmt_1^*$ (else : $stmt_2^*$)? $]\!](\sigma) =$
    $([$(#type:s)? async for $expr_1$ in $trans_E[\![$ $expr_2$ $]\!](\sigma)$:
      $trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma).\_1$ (else : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma).\_1$)?$], \sigma)$

$trans_S[\![$ while ($expr$) : $stmt_1^*$ (else : $stmt_2^*$)? $]\!](\sigma) =$
    $([$while ($trans_E[\![$ $expr$ $]\!](\sigma)$) : $trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma).\_1$ (else : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma).\_1$)?$], \sigma)$

$trans_S[\![$ if ($expr$) : $stmt^*$ (else : $stmt^*$)? $]\!](\sigma) =$
    $([$if ($trans_E[\![$ $expr$ $]\!](\sigma)$) : $trans_{\overline{S}}[\![$ $stmt^*$ $]\!](\sigma).\_1$ (else : $trans_{\overline{S}}[\![$ $stmt^*$ $]\!](\sigma).\_1$)?$], \sigma)$

$trans_S[\![$ (#type:s)? with $with\_item^*$: $stmt^*$ $]\!](\sigma) = ([$(#type:s)? with $with\_item^{*\prime}$: $stmt^{*\prime}], \sigma_2)$

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

$trans_S[\![$ match $expr$: $match\_case^*$ $]\!](\sigma) =$
    $([$match $trans_E[\![$ $expr$ $]\!](\sigma)$: ($trans_C[\![match\_case]\!](\sigma))^*], \sigma)$

$trans_S[\![$ raise $expr_1$? (from $expr_2$)? $]\!](\sigma) = ([$raise $expr_1$? (from $expr_2$)?$], \sigma)$

$trans_S[\![$ try : $stmt_1^*$ $exc\_handler^*$ (else : $stmt_2^*$)? (finally : $stmt_3^*$)? $]\!](\sigma) =$
    $([$try : $trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma).\_1$ ($trans_H[\![exc\_handler]\!](\sigma))^*$
    (else : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma).\_1$)? (finally : $trans_{\overline{S}}[\![$ $stmt_3^*$ $]\!](\sigma).\_1$)?$], \sigma)$

$trans_S[\![$ assert $expr_1$ $expr_2$? $]\!](\sigma) = ([$assert $trans_E[\![$ $expr_1$ $]\!](\sigma)$ $expr_2$?$], \sigma)$

$trans_S[\![$ import $alias^*$ $]\!](\sigma) =$
    LET σ_1 = $trans_{\overline{A}}[\![$ $alias^*$ $]\!](\sigma)$ **IN**
    IF σ_1 \ σ = [**"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 $alias^*], \sigma_1)$

$trans_S[\![$ from i id? import $alias^*$ $]\!](\sigma) = ([$from i id? import $alias^*], \sigma_1)$

$trans_S[\![$ global id^* $]\!](\sigma) = ([$global $id^*], \sigma)$

$trans_S[\![$ nonlocal id^* $]\!](\sigma) = ([$nonlocal $id^*], \sigma)$

A strict form of expr statements

$trans_S[\![\ expr_1\ (expr_{11}\ ...\ expr_{1n}\quad (id_1\ =\ )?\ expr_{21}\ ...\ (id_k\ =\ )?\ expr_{2k})\ ]\!](\sigma) =$

    # Config

    IF $id_t\ =\ \sigma($**"tensor_flow"**$)$ **AND**

        $expr_1\ =\ id_t.$`config.experimental.set_visible_devices` **THEN** $([], \sigma)$

    # model.compile

    ELIF $id_m\ =\ \sigma($**"model"**$)$ **AND** $expr_1\ =\ id_m.$`compile` **THEN**

        # string "adam" case

        IF $id_i\ =$ `optimizer` **WHEN** $2 \le i \le k$ **AND** $expr_{2i}\ =$ `"adam"` **THEN**

          ([`optim = tf.optimizers.Adam(learning_rate=0.001 * hvd.size)`,

           `optim = hvd.DistributedOptimizer(optim)`,

           $expr_1$ (`$expr_{11}$ ...  $expr_{1n}$ ($id_1$ = )? $expr_{21}$ ... $id_i$ = optim ... ($id_k$ = )? $expr_{2k}$)`], \sigma$)

        ELIF $expr_{11}\ =$ `"adam"` **THEN**

          ([`optim = tf.optimizers.Adam(learning_rate=0.001 * hvd.size)`,

           `optim = hvd.DistributedOptimizer(optim)`,

           $expr_1$ (`optim ...  $expr_{1n}$ ($id_1$ = )? $expr_{21}$ ... ($id_k$ = )? $expr_{2k}$)`], \sigma$)

        ELSE

          [$expr_1$ (`$expr_{11}$ ...  $expr_{1n}$ ($id_1$ = )? $expr_{21}$ ... ($id_k$ = )? $expr_{2k}$)`], \sigma$)

    # model.fit

    ELIF $id_m\ =\ \sigma($**"model"**$)$ **AND** $expr_1\ =\ id_m.$`fit` **THEN**

      IF $id_i\ =$ `callbacks` **WHEN** $2 \le i \le k$ **THEN**

    ([`callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(root_rank=0)]`,

     `if hvd.rank() == 0:  callbacks.append($expr_{2i}$)`,

     $expr_1$ (`optim ...  $expr_{1n}$ ($id_1$ = )? $expr_{21}$ ... callbacks = callbacks ... ($id_k$ = )? $expr_{2k}$)`], \sigma$)

    ELSE

        ([`callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(root_rank=0)]`,

        $expr_1$ (`optim ...  $expr_{1n}$ ($id_1$ = )? $expr_{21}$ ... ($id_k$ = )? $expr_{2k}$... callbacks = callbacks )`], \sigma$)

  # Root Rank Blocking

  ELIF $id_m\ =\ \sigma($**"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**

    $id_c\ =\ \sigma($**"checkpoint"**$)$ **AND** $expr_1\ =\ id_c.$`save` **OR**

    $id_t\ =\ \sigma($**"tensor_flow"**$)$ **AND** $expr_1\ =\ id_t.$`print` **OR**

    $expr_1\ =$ `print` **THEN**

      ([`if hvd.rank() == 0:` $expr_1$ (`$expr_{11}$ ... $expr_{1n}$   ($id_1$ = )? $expr_{21}$ ... ($id_k$ = )? $expr_{2k}$)`], \sigma$)

  ELSE

    [$trans_E[\![\ expr_1\ (expr_{11}\ ...\ expr_{1n}\quad (id_1\ =\ )?\ expr_{21}\ ...\ (id_k\ =\ )?\ expr_{2k})\ ]\!](\sigma)], \sigma$)

A general form of expr statements except for the strict form

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

$trans_S[\![$ `pass` $]\!](\sigma) = ([$`pass`$], \sigma)$

$trans_S[\![$ `break` $]\!](\sigma) = ([$`break`$], \sigma)$

$trans_S [\![$ continue $]\!](\sigma) = ([$continue$], \sigma)$

$$\boxed{trans_E \quad : \quad Expr \quad \to \quad \Sigma \quad \to \quad Expr}$$

$trans_E [\![expr_1 \quad boolop \quad expr_2]\!](\sigma) = trans_E [\![expr_1]\!](\sigma) \quad boolop \quad trans_E [\![expr_2]\!](\sigma)$

$trans_E [\![expr_1 \quad := \quad expr_2]\!](\sigma) = expr_1 \quad := \quad trans_E [\![expr_2]\!](\sigma)$

$trans_E [\![expr_1 \quad binop \quad expr_2]\!](\sigma) = trans_E [\![expr_1]\!](\sigma) \quad binop \quad trans_E [\![expr_2]\!](\sigma)$

$trans_E [\![unop \quad expr]\!](\sigma) = unop \quad trans_E [\![expr]\!](\sigma)$

$trans_E [\![$ lambda $args \quad : \quad expr]\!](\sigma) =$ lambda $args \quad : \quad trans_E [\![expr]\!](\sigma)$

$trans_E [\![expr_1 \quad$ if $expr_2 \quad$ else $expr_3]\!](\sigma) =$
    $trans_E [\![expr_1]\!](\sigma) \quad$ if $\quad trans_E [\![expr_2]\!](\sigma) \quad$ else $\quad trans_E [\![expr_3]\!](\sigma)$

$trans_E [\![expr_1 \quad$ if $expr_2 \quad$ else $expr_3]\!](\sigma) =$
    $trans_E [\![expr_1]\!](\sigma) \quad$ if $\quad trans_E [\![expr_2]\!](\sigma) \quad$ else $\quad trans_E [\![expr_3]\!](\sigma)$

$trans_E [\![\{(expr_1 : expr_2)^* \ (**expr_3)^*\}]\!](\sigma) = \{(expr_1 : trans_E[\![expr_2]\!](\sigma))^* \ (**expr_3)^*\}$

$trans_E [\![\{expr^*\}]\!](\sigma) = \{(trans_E[\![expr]\!](\sigma))^*\}$

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

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

$trans_E [\![\{expr_1 : expr_2 \ comprehension^*\}]\!](\sigma) = \{expr_1 : trans_E [\![expr_2]\!](\sigma) \ (trans_O[\![comprehension]\!](\sigma))^*\}$

$trans_E [\![\{expr \quad comprehension^*\}]\!](\sigma) = \{trans_E [\![expr]\!](\sigma) \ (trans_O[\![comprehension]\!](\sigma))^*\}$

$trans_E [\![[expr \quad comprehension^*]]\!](\sigma) = [trans_E [\![expr]\!](\sigma) \ (trans_O[\![comprehension]\!](\sigma))^*]$

$trans_E [\![(expr \ comprehension^*)]\!](\sigma) = (trans_E [\![expr]\!](\sigma) \ (trans_O[\![comprehension]\!](\sigma))^*)$

$trans_E [\![$ await $expr]\!](\sigma) =$ await $\ trans_E [\![expr]\!](\sigma)$

$trans_E [\![$ yield $expr?]\!](\sigma) =$ yield $\ (trans_E [\![expr]\!](\sigma))?$

$trans_E[\![\ \texttt{yield from}\ expr\ ]\!](\sigma) = \texttt{yield from}\ trans_E[\![\ expr\ ]\!](\sigma)$

$trans_E[\![\ expr_1\ (compop\ expr_2)^*\ ]\!](\sigma) = trans_E[\![\ expr_1\ ]\!](\sigma)\ (compop\ trans_E[\![expr_2]\!](\sigma))^*$

$trans_E[\![\ expr_1\ (expr_{11}\ ...\ expr_{1n}\quad (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ ]\!](\sigma) =$
$\quad trans_E[\![\ expr_1\ ]\!](\sigma)\ (trans_E[\![\ expr_{11}\ ]\!](\sigma)\ ...\ trans_E[\![\ expr_{1n}\ ]\!](\sigma)$
$\quad\quad (id_1\ \texttt{=}\ )?\ trans_E[\![\ expr_{21}\ ]\!](\sigma)\ ...\ (id_k\ \texttt{=}\ )?\ trans_E[\![\ expr_{2k}\ ]\!](\sigma))$

$trans_E[\![\ \{expr_1\quad (\texttt{!}i)?\quad (\texttt{:}expr_2)?\}\ ]\!](\sigma) = \{expr_1\quad (\texttt{!}i)?\quad (\texttt{:}expr_2)?\}$

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

$trans_E[\![\ constant\ ]\!](\sigma) = constant$

$trans_E[\![\ expr.id\ ]\!](\sigma) = (trans_E[\![\ expr\ ]\!](\sigma)).id$

$trans_E[\![\ expr_1[expr_2]\ ]\!](\sigma) = trans_E[\![\ expr_1\ ]\!](\sigma)[trans_E[\![\ expr_2\ ]\!](\sigma)]$

$trans_E[\![\ \texttt{*}expr\ ]\!](\sigma) = \texttt{*}expr$

$trans_E[\![\ \texttt{**}expr\ ]\!](\sigma) = \texttt{**}expr$

$trans_E[\![\ id\ ]\!](\sigma) = id$

$trans_E[\![\ expr_1?\ (\texttt{:}expr_2)?\ (\texttt{:}expr_3)?\ ]\!](\sigma) = trans_E[\![\ expr_1\ ]\!](\sigma)?\ (\texttt{:}trans_E[\![\ expr_2\ ]\!](\sigma))?\ (\texttt{:}trans_E[\![\ expr_3\ ]\!](\sigma))?$

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

$trans_O[\![\ \texttt{for}\ expr_1\ \texttt{in}\ expr_2\ (\texttt{if}\ expr_3)^*\ ]\!](\sigma) = \texttt{for}\ expr_1\ \texttt{in}\ trans_E[\![\ expr_2\ ]\!](\sigma)\ (\texttt{if}\ trans_E[\![expr_3]\!](\sigma))^*$

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

$trans_H[\![\ \texttt{except}\ expr?\ (\texttt{as}\ id)?\ \texttt{:}\ stmt^*\ ]\!](\sigma) = \texttt{except}\ expr?\ (\texttt{as}\ id)?\ \texttt{:}\ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma).\_\mathbf{1}$

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

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

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

$trans_A[\![\ id\ ]\!](\sigma) =$
    LET id = `tensorflow` **THEN** $\sigma[\textbf{"tensor\_flow"} \mapsto id]$
    LET id = `os` **THEN** $\sigma[\textbf{"os"} \mapsto id]$
    ELSE σ

$trans_A[\![\ id_1\ \text{as}\ id_2\ ]\!](\sigma) =$
    LET id_1 = `tensorflow` **THEN** $\sigma[\textbf{"tensor\_flow"} \mapsto id_2]$
    LET id_1 = `os` **THEN** $\sigma[\textbf{"os"} \mapsto id_2]$
    ELSE σ

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

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

$trans_{\overline{W}}[\![\ with\_item_1\ with\_item_2\ ...\ with\_item_n\ ]\!](\sigma) =$
    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**
    $\ldots$
    LET $with\_item_n',\ \sigma_n\ =\ trans_W[\![\ with\_item_n\ ]\!](\sigma_{n-1})$ **IN**
    $(with\_item_1'\ \text{::}\ with\_item_2'\ \text{::}\ ...\ \text{::}\ [with\_item_n'],\ \sigma_n)$

$\boxed{trans_W\ :\ WithItem\ \rightarrow\ \Sigma\ \rightarrow\ (WithItem\ \times \Sigma)}$

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

$trans_W[\![\ expr_1\ \text{as}\ expr_2\ ]\!](\text{smodenv}) =$
    $(trans_E[\![\ expr_1\ ]\!](\sigma)\ \text{as}\ expr_2,\ \sigma)$

$\boxed{trans_C\ :\ MatchCase\ \rightarrow\ \Sigma\ \rightarrow\ MatchCase}$

$trans_C[\![\ \text{case}\ pattern\ (\text{if}\ expr)?\ :\ stmt^*\ ]\!](\sigma) =$
    `case` $trans_P[\![\ pattern\ ]\!](\sigma)\ (\text{if}\ trans_E[\![\ expr\ ]\!](\sigma))?\ :\ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma).\_1$

$\boxed{trans_P\ :\ Pattern\ \rightarrow\ \Sigma\ \rightarrow\ Pattern}$

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

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

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

$trans_P[\![\ *(id)?\ ]\!](\sigma) = *(id)?$

$trans_P[\![\ \{(expr\ :\ pattern)^*\ id?\}\ ]\!](\sigma) = \{(expr\ :\ trans_P[\![pattern]\!](\sigma))^*\ id?\}$

$trans_P[\![\ expr\ (pattern_1^*\ (id = pattern_2)^*)\ ]\!](\sigma) = expr\ (trans_P[\![pattern_1]\!](\sigma)^*\ (id = trans_P[\![pattern_2]\!](\sigma))^*)$

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

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

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

2.4 Rule: TensorFlow 2.x GradientTape API Pattern

2.4.1 Transformation Rules

$$\boxed{trans_M \ : \ Module \ \rightarrow \ Module}$$
$$trans_M \llbracket \ stmt^* \ \ type\_ignore \ \rrbracket \ = \ trans_{\overline{S}} \llbracket \ stmt^* \ \rrbracket(\sigma).\_\mathbf{1} \ \ type\_ignore$$

$$\boxed{trans_{\overline{S}} \ : \ Stmt \ \mathbf{list} \ \rightarrow \ \Sigma \ \rightarrow \ (Stmt \ \mathbf{list} \ \times \ \Sigma)}$$
$$
\begin{aligned}
trans_{\overline{S}} \llbracket \ stmt_1 \ \ stmt_2 \ ... \ stmt_n \ \rrbracket(\sigma) \ = \ & \mathbf{LET} \ \ stmt_1^{*\prime}, \sigma_1 \ = \ trans_S \llbracket \ stmt_1 \ \rrbracket(\sigma) \ \ \mathbf{IN} \\
& \mathbf{LET} \ \ stmt_2^{*\prime}, \sigma_2 \ = \ trans_S \llbracket \ stmt_2 \ \rrbracket(\sigma_1) \ \ \mathbf{IN} \\
& \ ... \\
& \mathbf{LET} \ \ stmt_n^{*\prime}, \sigma_n \ = \ trans_S \llbracket \ stmt_n \ \rrbracket(\sigma_{n-1}) \ \ \mathbf{IN} \\
& (stmt_1^{*\prime} \ @ \ stmt_2^{*\prime} \ @ \ ... \ @ \ stmt_n^{*\prime}, \ \sigma_n)
\end{aligned}
$$

$$\boxed{trans_S \ : \ Stmt \ \rightarrow \ \Sigma \ \rightarrow \ (Stmt \ \mathbf{list} \ \times \ \Sigma)}$$

$trans_S \llbracket \ (@expr_1)^* \ \ \mathtt{def} \ id \ (args) \ \ (\mathtt{->} \ expr_2)? \ \ : \ (\mathtt{\#type:}s)? \ stmt^* \ \rrbracket(\sigma) =$
$\quad ([(@expr_1)^* \ \ \mathtt{def} \ id \ (args) \ \ (\mathtt{->} \ expr_2)? \ \ : \ (\mathtt{\#type:}s)? \ trans_{\overline{S}} \llbracket \ stmt^* \ \rrbracket(\sigma).\_\mathbf{1}], \sigma)$

$trans_S \llbracket \ (@expr_1)^* \ \ \mathtt{async \ def} \ id \ (args) \ \ (\mathtt{->} \ expr_2)? \ \ : \ (\mathtt{\#type:}s)? \ stmt^* \ \rrbracket(\sigma) =$
$\quad ([(@expr_1)^* \ \ \mathtt{async \ def} \ id \ (args) \ \ (\mathtt{->} \ expr_2)? \ \ : \ (\mathtt{\#type:}s)? \ trans_{\overline{S}} \llbracket \ stmt^* \ \rrbracket(\sigma).\_\mathbf{1}], \sigma)$

$trans_S \llbracket \ (@expr_1)^* \ \ \mathtt{class} \ id \ (expr_2^* \ keyword^*) \ \ : \ stmt^* \ \rrbracket(\sigma) =$
$\quad ([(@expr_1)^* \ \ \mathtt{class} \ id \ (expr_2^* \ keyword^*) \ \ : \ trans_{\overline{S}} \llbracket \ stmt^* \ \rrbracket(\sigma).\_\mathbf{1}], \sigma) =$

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

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

A strict form of assignment statements

$trans_S \llbracket \ id_r \ \mathtt{=} \ expr_1 \ (expr_{11} \ ... \ expr_{1n} \ \ (id_1 \ \mathtt{=} \ )? \ expr_{21} \ ... \ (id_k \ \mathtt{=} \ )? \ expr_{2k}) \ (\mathtt{\#type:}s)? \ \rrbracket(\sigma) =$
$\quad \textcolor{blue}{\# \ Dataset}$
$\quad \mathbf{IF} \ \ \sigma(\text{``}\mathbf{tensor\_flow}\text{''}) \ = \ id_t \ \ \mathbf{AND} \ \ expr_1 \ = \ id_t.\mathtt{data.Dataset}.expr_3 \ \ \mathbf{THEN}$
$\quad\quad ([id_r \ \mathtt{=} \ expr_1 \ (expr_{11} \ ... \ expr_{1n} \ \ (id_1 \ \mathtt{=} \ )? \ expr_{21} \ ... \ (id_k \ \mathtt{=} \ )? \ expr_{2k}) \ (\mathtt{\#type:}s)?],$
$\quad\quad\quad \sigma[\text{``}\mathbf{dataset}\text{''} \mapsto id_r])$
$\quad \textcolor{blue}{\# \ Variable \ Broadcasting}$

ELIF $\sigma(\text{``optimizer''}) = id_t$ **AND** $expr_1 = id_t.\texttt{apply\_gradients}$ **THEN**

    IF $id_i = \texttt{grads\_and\_vars}$ **WHEN** $1 \leq i \leq k$ **THEN**

        LET $id_z = \textbf{NewID()}$ **IN**

        $([id_z = expr_{2i},$

        $id_r = expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ id_i = id_z \ ... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?,$

        `global hvd_broadcast_done,`

        `if not hvd_broadcast_done:` $\lceil$ `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` $\rceil], \sigma)$

    ELSE

        LET $id_z = \textbf{NewID()}$ **IN**

        $([id_z = expr_{11},$

        $id_r = expr_1 \ (id_z \ expr_{12} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?,$

        `global hvd_broadcast_done,`

        `if not hvd_broadcast_done:` $\lceil$ `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` $\rceil], \sigma)$

 # Learning rate scheduler scaling

IF $expr_1 <:$ `"tensorflow.keras.optimizers.schedules.PiecewiseConstantDecay"` **THEN**

    $([id_r = expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?],$

      $\sigma[\text{``lr\_scheduler''} \mapsto id_r])$

ELIF $expr_1 <:$ `tensorflow.keras.optimizers.schedules.LearningRateSchedule` **THEN**

    IF $id_i = \texttt{initial\_learning\_rate}$ **WHEN** $1 \leq i \leq k$ **THEN**

        $([id_r = expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ id_i = expr_{2i} \ * \ \texttt{hvd.size()}$

           $... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?], \sigma[\text{``lr\_scheduler''} \mapsto id_r])$

    ELSE

        $([id_r = expr_1 \ (expr_{11} \ * \ \texttt{hvd.size()} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?],$

          $\sigma[\text{``lr\_scheduler''} \mapsto id_r])$

ELIF $expr_1 <:$ `tensorflow.compat.v1.train.exponential_decay` **THEN**

    IF $id_i = \texttt{learning\_rate}$ **WHEN** $1 \leq i \leq k$ **THEN**

        $([id_r = expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ id_i = expr_{2i} \ * \ \texttt{hvd.size()}$

           $... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?], \sigma[\text{``lr\_scheduler''} \mapsto id_r])$

    ELSE

        $([id_r = expr_1 \ (expr_{11} \ * \ \texttt{hvd.size()} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?],$

          $\sigma[\text{``lr\_scheduler''} \mapsto id_r])$

 # Optimizer learning rate scaling and wrapping

IF $expr_1 <:$ `tensorflow.keras.optimizers.Optimizer` **THEN**

    $([id_r = expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_1 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k}) \ (\texttt{\#type:}s)?],$

      $\sigma[\text{``optimizer''} \mapsto id_r])$

 # Model related

ELIF $expr_1 <:$ `tensorflow.keras.Model` **THEN**

    $([id_r = expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_2 =)? \ expr_{21} \ ... \ (id_k =)? \ expr_{2k})], \sigma[\text{``model''} \mapsto id_r])$

ELIF $id_m = \text{``model''}$ **AND** $expr_1 = id_t.\texttt{evaluate}$ **THEN**

    IF $id_i = \texttt{verbose}$ **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)?], σ)

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)?], σ)

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

A general form of assignment statements except for the strict form

$trans_S[\![$ $expr_1^*$ = $expr_2$ (#type:s)? $]\!](\sigma)$ =

 # CUDA Visible Devices

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

$([], \sigma)$

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

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

$trans_S[\![$ $expr_1$: $expr_2$ (= $expr_3$)? $]\!](\sigma)$ =

 # Dataset

IF $expr_1$ = id_1 **AND** $\sigma($**"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)?], $\sigma[$**"dataset"**$\mapsto id_1])$

ELSE $([expr_1$: $expr_2$ (= $trans_E[\![$ $expr_3$ $]\!](\sigma))$?], $\sigma)$

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

 # Adjust Steps

IF $\sigma($**"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()) :
   $trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma).\_1$ (else : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma).\_1$)?], $\sigma)$

ELSE $([$(#type:s)? for $expr_1$ in $trans_E[\![$ $expr_2$ $]\!](\sigma)$:
   $trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma).\_1$ (else : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma).\_1$)?], $\sigma)$

$trans_S[\![$ (#type:s)? `async for` $expr_1$ `in` $expr_2$: $stmt_1^*$ (`else` : $stmt_2^*$)? $]\!](\sigma) =$

    # *Adjust Steps*

      IF $\sigma($**"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())` :

            $trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma)._{-}\mathbf{1}$ (`else` : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma)._{-}\mathbf{1})?], \sigma)$

      ELSE $([$(#type:s)? `async for` $expr_1$ `in` $trans_E[\![$ $expr_2$ $]\!](\sigma)$:

        $trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma)._{-}\mathbf{1}$ (`else` : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma)._{-}\mathbf{1})?], \sigma)$

$trans_S[\![$ `while` ($expr$) : $stmt_1^*$ (`else` : $stmt_2^*$)? $]\!](\sigma) =$

    $([$`while` ($trans_E[\![$ $expr$ $]\!](\sigma)$) : $trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma)._{-}\mathbf{1}$ (`else` : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma)._{-}\mathbf{1})?], \sigma)$

$trans_S[\![$ `if` ($expr$) : $stmt^*$ (`else` : $stmt^*$)? $]\!](\sigma) =$

    $([$`if` ($trans_E[\![$ $expr$ $]\!](\sigma)$) : $trans_{\overline{S}}[\![$ $stmt^*$ $]\!](\sigma)._{-}\mathbf{1}$ (`else` : $trans_{\overline{S}}[\![$ $stmt^*$ $]\!](\sigma)._{-}\mathbf{1})?], \sigma)$

$trans_S[\![$ (#type:s)? `with` $with\_item^*$: $stmt^*$ $]\!](\sigma) =$

    LET $with\_item^{*\prime}, \sigma_1 = trans_{\overline{W}}[\![$ $with\_item^*$ $]\!](\sigma)$ **IN**

    LET $stmt^{*\prime}, \sigma_2 = trans_{\overline{S}}[\![$ $stmt^*$ $]\!](\sigma_1)$ **IN**

    # *Wrapping Gradient Tape*

    IF $\sigma_1 \setminus \sigma = [$**"gradient_tape"**$\mapsto id]$ **THEN**

      $([$(#type:s)? `with` $with\_item^{*\prime}$: $stmt^{*\prime}$,

      id `= hvd.DistributedGradientTape`(id)$], \sigma_2)$

    ELSE $([$(#type:s)? `with` $with\_item^{*\prime}$: $stmt^{*\prime}], \sigma_2)$

$trans_S[\![$ (#type:s)? `async with` $with\_item^*$: $stmt^*$ $]\!](\sigma) =$

    LET $with\_item^{*\prime}, \sigma_1 = trans_{\overline{W}}[\![$ $with\_item^*$ $]\!](\sigma)$ **IN**

    LET $stmt^{*\prime}, \sigma_2 = trans_{\overline{S}}[\![$ $stmt^*$ $]\!](\sigma_1)$ **IN**

    # *Wrapping Gradient Tape*

    IF $\sigma_1 \setminus \sigma = [$**"gradient_tape"**$\mapsto id]$ **THEN**

      $([$(#type:s)? `async with` $with\_item^{*\prime}$: $stmt^{*\prime}$,

      id `= hvd.DistributedGradientTape`(id)$], \sigma_2)$

    ELSE $([$(#type:s)? `async with` $with\_item^{*\prime}$: $stmt^{*\prime}], \sigma_2)$

$trans_S[\![$ `match` $expr$: $match\_case^*$ $]\!](\sigma) =$

    $([$`match` $trans_E[\![$ $expr$ $]\!](\sigma)$: ($trans_C[\![match\_case]\!](\sigma))^*], \sigma)$

$trans_S[\![$ `raise` $expr_1$? (`from` $expr_2$)? $]\!](\sigma) = ([$`raise` $expr_1$? (`from` $expr_2$)?$], \sigma)$

$trans_S[\![$ `try` : $stmt_1^*$ $exc\_handler^*$ (`else` : $stmt_2^*$)? (`finally` : $stmt_3^*$)? $]\!](\sigma) =$

    $([$`try` : $trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma)._{-}\mathbf{1}$ ($trans_H[\![exc\_handler]\!](\sigma))^*$

    (`else` : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma)._{-}\mathbf{1})?$ (`finally` : $trans_{\overline{S}}[\![$ $stmt_3^*$ $]\!](\sigma)._{-}\mathbf{1})?], \sigma)$

$trans_S[\![$ `assert` $expr_1$ $expr_2$? $]\!](\sigma) = ([$`assert` $trans_E[\![$ $expr_1$ $]\!](\sigma)$ $expr_2$?$], \sigma)$

$trans_S[\![$ import $alias^*$ $]\!](\sigma) =$
    LET σ_1 $=$ $trans_{\overline{A}}[\![$ $alias^*$ $]\!](\sigma)$ **IN**
    # import
    IF σ_1 $\setminus$ σ $=$ [**"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')], σ_1)
    ELSE ([import $alias^*$], σ_1)

$trans_S[\![$ global id^* $]\!](\sigma) = ([$global $id^*]$, $\sigma)$

$trans_S[\![$ nonlocal id^* $]\!](\sigma) = ([$nonlocal $id^*]$, $\sigma)$

A strict form of expr statements

$trans_S[\![$ $expr_1$ ($expr_{11}$ ... $expr_{1n}$ $(id_1$ =)? $expr_{21}$ ... $(id_k$ =)? $expr_{2k}$) $]\!](\sigma) =$
    # Variable Broadcasting
    IF $\sigma($**"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 $=$ **NewID()** **IN**
        ([id_z = $expr_{2i}$,
        $expr_1$ ($expr_{11}$ ... $expr_{1n}$ $(id_1$ =)? $expr_{21}$ ... id_i = id_z ... $(id_k$ =)? $expr_{2k}$),
        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]], σ)
      ELSE
        LET id_z $=$ **NewID()** **IN**
        ([id_z = $expr_{11}$,
        $expr_1$ (id_z $expr_{12}$ ... $expr_{1n}$ $(id_1$ =)? $expr_{21}$ ... $(id_k$ =)? $expr_{2k}$),
        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]], σ)
    # Config
    IF id_t $=$ $\sigma($**"tensor_flow"**$)$ **AND**
      $expr_1$ $=$ id_t.config.experimental.set_visible_devices **THEN** ([], σ)
    # Root Rank Blocking
    ELIF id_m $=$ $\sigma($**"model"**$)$ **AND** ($expr_1$ $=$ id_m.write **OR**
      $expr_1$ $=$ id_m.summary **OR** $expr_1$ $=$ id_m.save_weights **OR**

$$expr_1 \ = \ expr\texttt{.load\_weights} \ \ \textbf{OR} \ \ expr_1 \ = \ id_m\texttt{.save}) \ \ \textbf{OR}$$
$$id_c \ = \ \sigma(\textbf{``checkpoint''}) \ \ \textbf{AND} \ \ expr_1 \ = \ id_c\texttt{.save} \ \ \textbf{OR}$$
$$id_t \ = \ \sigma(\textbf{``tensor\_flow''}) \ \ \textbf{AND} \ \ expr_1 \ = \ id_t\texttt{.print} \ \ \textbf{OR}$$
$$expr_1 \ = \ \texttt{print} \ \ \textbf{THEN}$$
$$([\texttt{if hvd.rank() == 0:} \ expr_1 \ (expr_{11} \ ... \ expr_{1n} \ \ (id_1 \ \texttt{=} \ )? \ expr_{21} \ ... \ (id_k \ \texttt{=} \ )? \ expr_{2k})], \ \sigma)$$
$$\textbf{ELSE}$$
$$[trans_E[\![\ expr_1 \ (expr_{11} \ ... \ expr_{1n} \ \ (id_1 \ \texttt{=} \ )? \ expr_{21} \ ... \ (id_k \ \texttt{=} \ )? \ expr_{2k}) \ ]\!](\sigma)], \ \sigma)$$

A general form of expr statements except for the strict form

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

$$trans_S[\![\ \texttt{pass} \ ]\!](\sigma) = ([\texttt{pass}], \ \sigma)$$

$$trans_S[\![\ \texttt{break} \ ]\!](\sigma) = ([\texttt{break}], \ \sigma)$$

$$trans_S[\![\ \texttt{continue} \ ]\!](\sigma) = ([\texttt{continue}], \ \sigma)$$

$$\boxed{trans_E \ \ : \ \ Expr \ \ \rightarrow \ \ \Sigma \ \ \rightarrow \ \ Expr}$$
$$trans_E[\![\ expr_1 \ \ boolop \ expr_2 \ ]\!](\sigma) = trans_E[\![\ expr_1 \ ]\!](\sigma) \ \ boolop \ \ trans_E[\![\ expr_2 \ ]\!](\sigma)$$

$$trans_E[\![\ expr_1 \ \ \texttt{:=} \ \ expr_2 \ ]\!](\sigma) = expr_1 \ \ \texttt{:=} \ \ trans_E[\![\ expr_2 \ ]\!](\sigma)$$

$$trans_E[\![\ expr_1 \ \ binop \ expr_2 \ ]\!](\sigma) = trans_E[\![\ expr_1 \ ]\!](\sigma) \ \ binop \ \ trans_E[\![\ expr_2 \ ]\!](\sigma)$$

$$trans_E[\![\ unop \ expr \ ]\!](\sigma) = unop \ \ trans_E[\![\ expr \ ]\!](\sigma)$$

$$trans_E[\![\ \texttt{lambda} \ args \ : \ expr \ ]\!](\sigma) = \texttt{lambda} \ \ args \ : \ trans_E[\![\ expr \ ]\!](\sigma)$$

$$trans_E[\![\ expr_1 \ \ \texttt{if} \ expr_2 \ \ \texttt{else} \ expr_3 \ ]\!](\sigma) =$$
$$trans_E[\![\ expr_1 \ ]\!](\sigma) \ \ \texttt{if} \ \ trans_E[\![\ expr_2 \ ]\!](\sigma) \ \ \texttt{else} \ \ trans_E[\![\ expr_3 \ ]\!](\sigma)$$

$$trans_E[\![\ expr_1 \ \ \texttt{if} \ expr_2 \ \ \texttt{else} \ expr_3 \ ]\!](\sigma) =$$
$$trans_E[\![\ expr_1 \ ]\!](\sigma) \ \ \texttt{if} \ \ trans_E[\![\ expr_2 \ ]\!](\sigma) \ \ \texttt{else} \ \ trans_E[\![\ expr_3 \ ]\!](\sigma)$$

$$trans_E[\![\ \{(expr_1 \ : \ expr_2)^* \ (**expr_3)^*\} \ ]\!](\sigma) = \{(expr_1 \ : \ trans_E[\![expr_2]\!](\sigma))^* \ (**expr_3)^*\}$$

$$trans_E[\![\ \{expr^*\} \ ]\!](\sigma) = \{(trans_E[\![expr]\!](\sigma))^*\}$$

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

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

$trans_E[\![\ \{expr_1 : expr_2\ comprehension^*\}\ ]\!](\sigma) = \{expr_1 : trans_E[\![\ expr_2\ ]\!](\sigma)\ (trans_O[\![comprehension]\!](\sigma))^*\}$

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

$trans_E[\![\ [expr\ \ comprehension^*]\ ]\!](\sigma) = [trans_E[\![\ expr\ ]\!](\sigma)\ (trans_O[\![comprehension]\!](\sigma))^*]$

$trans_E[\![\ (expr\ comprehension^*)\ ]\!](\sigma) = (trans_E[\![\ expr\ ]\!](\sigma)\ (trans_O[\![comprehension]\!](\sigma))^*)$

$trans_E[\![\ \texttt{await}\ expr\ ]\!](\sigma) = \texttt{await}\ trans_E[\![\ expr\ ]\!](\sigma)$

$trans_E[\![\ \texttt{yield}\ expr?\ ]\!](\sigma) = \texttt{yield}\ (trans_E[\![\ expr\ ]\!](\sigma))?$

$trans_E[\![\ \texttt{yield from}\ expr\ ]\!](\sigma) = \texttt{yield from}\ trans_E[\![\ expr\ ]\!](\sigma)$

$trans_E[\![\ expr_1\ (compop\ expr_2)^*\ ]\!](\sigma) = trans_E[\![\ expr_1\ ]\!](\sigma)\ (compop\ trans_E[\![expr_2]\!](\sigma))^*$

$trans_E[\![\ expr_1\ (expr_{11}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ ]\!](\sigma) =$
    # *Adjust Steps*
  IF $\sigma(\text{``\textbf{dataset}''})\ \texttt{=}\ id_t$ **AND** $expr_1\ \texttt{=}\ id_t.\texttt{take}$ **THEN**
    IF $id_i\ \texttt{=}\ \texttt{count}$ **WHEN** $1 \leq \text{i} \leq \text{k}$ **THEN**
      $expr_1\ (expr_{11}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ id_i\ \texttt{=}\ expr_{2i}\ \texttt{// hvd.size()}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})$
    ELSE
      $expr_1\ (expr_{11}\ \texttt{// hvd.size()}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})$
  ELSE
    $trans_E[\![\ expr_1\ ]\!](\sigma)\ (trans_E[\![\ expr_{11}\ ]\!](\sigma)\ ...\ trans_E[\![\ expr_{1n}\ ]\!](\sigma)$
      $(id_1\ \texttt{=}\ )?\ trans_E[\![\ expr_{21}\ ]\!](\sigma)\ ...\ (id_k\ \texttt{=}\ )?\ trans_E[\![\ expr_{2k}\ ]\!](\sigma))$

$trans_E[\![\ \{expr_1\ \ \ (\texttt{!}i)?\ \ (\texttt{:}expr_2)?\}\ ]\!](\sigma) = \{expr_1\ \ \ (\texttt{!}i)?\ \ (\texttt{:}expr_2)?\}$

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

$trans_E[\![\ constant\ ]\!](\sigma) = constant$

$trans_E[\![\ expr.id\ ]\!](\sigma) = (trans_E[\![\ expr\ ]\!](\sigma)).id$

$trans_E[\![\ expr_1[expr_2]\ ]\!](\sigma) = trans_E[\![\ expr_1\ ]\!](\sigma)[trans_E[\![\ expr_2\ ]\!](\sigma)]$

$trans_E[\![\ {*}expr\ ]\!](\sigma) = {*}expr$

$trans_E[\![\ **expr \ ]\!](\sigma) = **expr$

$trans_E[\![\ id \ ]\!](\sigma) = id$

$trans_E[\![\ expr_1? \ (:expr_2)? \ (:expr_3)? \ ]\!](\sigma) = trans_E[\![\ expr_1 \ ]\!](\sigma)? \ (:trans_E[\![\ expr_2 \ ]\!](\sigma))? \ (:trans_E[\![\ expr_3 \ ]\!](\sigma))?$

$$\boxed{trans_O \quad : \quad Comprehension \quad \rightarrow \quad \Sigma \quad \rightarrow \quad Comprehension}$$
$trans_O[\![\ \texttt{for} \ expr_1 \ \texttt{in} \ expr_2 \ (\texttt{if} \ expr_3)^* \ ]\!](\sigma) = \texttt{for} \ expr_1 \ \texttt{in} \ trans_E[\![\ expr_2 \ ]\!](\sigma) \ (\texttt{if} \ trans_E[\![expr_3]\!](\sigma))^*$

$$\boxed{trans_H \quad : \quad ExcHandler \quad \rightarrow \quad \Sigma \quad \rightarrow \quad ExcHandler}$$
$trans_H[\![\ \texttt{except} \ expr? \ (\texttt{as} \ id)? \ : \ stmt^* \ ]\!](\sigma) = \texttt{except} \ expr? \ (\texttt{as} \ id)? \ : \ trans_{\overline{S}}[\![\ stmt^* \ ]\!](\sigma).\_\mathbf{1}$

$$\boxed{trans_{\overline{A}} \quad : \quad Alias \ \textbf{list} \quad \rightarrow \quad \Sigma \quad \rightarrow \quad \Sigma}$$
$trans_{\overline{A}}[\![\ alias_1 \ ... \ alias_{n-1} \ alias_n \ ]\!](\sigma) = trans_A[\![\ alias_n \ ]\!](trans_A[\![\ alias_{n-1} \ ]\!](... \ trans_A[\![\ alias_1 \ ]\!](\sigma)))$

$$\boxed{trans_A \quad : \quad Alias \quad \rightarrow \quad \Sigma \quad \rightarrow \quad \Sigma}$$
$trans_A[\![\ id \ ]\!](\sigma) =$
    LET $id = \texttt{tensorflow}$ **THEN** $\sigma[\text{``}\textbf{tensor\_flow''}\mapsto id]$
    LET $id = \texttt{os}$ **THEN** $\sigma[\text{``}\textbf{os''}\mapsto id]$
    LET $id = \texttt{config}$ **THEN** $\sigma[\text{``}\textbf{config''}\mapsto id]$
    ELSE σ

$trans_A[\![\ id_1 \ \texttt{as} \ id_2 \ ]\!](\sigma) =$
    LET $id_1 = \texttt{tensorflow}$ **THEN** $\sigma[\text{``}\textbf{tensor\_flow''}\mapsto id_2]$
    LET $id_1 = \texttt{os}$ **THEN** $\sigma[\text{``}\textbf{os''}\mapsto id_2]$
    LET $id = \texttt{config}$ **THEN** $\sigma[\text{``}\textbf{config''}\mapsto id_2]$
    ELSE σ

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

$$\boxed{trans_{\overline{W}} \quad : \quad WithItem \ \textbf{list} \quad \rightarrow \quad \Sigma \quad \rightarrow \quad (WithItem \ \textbf{list} \times \Sigma)}$$
$trans_{\overline{W}}[\![\ with\_item_1 \ with\_item_2 \ ... \ with\_item_n \ ]\!](\sigma) =$
    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**
    $(with\_item_1' :: with\_item_2' :: ... :: [with\_item_n'], \sigma_n)$

$$\boxed{trans_W \quad : \quad WithItem \quad \rightarrow \quad \Sigma \quad \rightarrow \quad (WithItem \ \times \Sigma)}$$
$trans_W[\![\ expr \ ]\!](\sigma) = (trans_E[\![\ expr \ ]\!](\sigma), \sigma)$

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

    IF $\sigma(\textbf{``tensor\_flow''}) = id_1$ **AND** $expr_1 = id_1.\texttt{GradientTape()}$ **AND** $expr_2 = id_2$ **THEN**

        $(expr_1 \ \texttt{as} \ expr_2, \sigma[\textbf{``gradient\_tape''}\mapsto id_2])$

    ELSE $(trans_E[\![\ expr_1 \ ]\!](\sigma) \ \texttt{as} \ expr_2, \sigma)$

$\boxed{trans_C \quad : \quad MatchCase \quad \rightarrow \quad \Sigma \quad \rightarrow \quad MatchCase}$

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

    $\texttt{case} \ trans_P[\![\ pattern \ ]\!](\sigma) \ (\texttt{if} \ trans_E[\![\ expr \ ]\!](\sigma))? \ : \ trans_{\overline{S}}[\![\ stmt^* \ ]\!](\sigma)\textbf{.\_1}$

$\boxed{trans_P \quad : \quad Pattern \quad \rightarrow \quad \Sigma \quad \rightarrow \quad Pattern}$

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

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

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

$trans_P[\![\ \texttt{*}(id)? \ ]\!](\sigma) = \texttt{*}(id)?$

$trans_P[\![\ \{(expr \ : \ pattern)^* \ id?\} \ ]\!](\sigma) = \{(expr \ : \ trans_P[\![pattern]\!](\sigma))^* \ id?\}$

$trans_P[\![\ expr \ (pattern_1^* \ (id = pattern_2)^*) \ ]\!](\sigma) = expr \ (trans_P[\![pattern_1]\!](\sigma)^* \ (id = trans_P[\![pattern_2]\!](\sigma))^*)$

$trans_P[\![\ (pattern \ \texttt{as})? \ id \ ]\!](\sigma) = (trans_P[\![\ pattern \ ]\!](\sigma) \ \texttt{as})? \ id$

$trans_P[\![\ pattern_1 \ | \ pattern_2 \ ]\!](\sigma) = trans_P[\![\ pattern_1 \ ]\!](\sigma) \ | \ trans_P[\![\ pattern_2 \ ]\!](\sigma)$

$trans_P[\![\ \_ \ ]\!](\sigma) = \_$

2.5   Rule: TensorFlow 1.x Mainscript API Pattern

2.5.1   Transformation Rules

$\boxed{trans_M \quad : \quad Module \quad \rightarrow \quad Module}$

$trans_M[\![\ stmt^* \ type\_ignore \ ]\!] \quad = \quad trans_{\overline{S}}[\![\ stmt^* \ ]\!](\sigma)\textbf{.\_1} \ type\_ignore$

$\boxed{trans_{\overline{S}} \quad : \quad Stmt \ \textbf{list} \quad \rightarrow \quad \Sigma \quad \rightarrow \quad (Stmt \ \textbf{list} \quad \times \quad \Sigma)}$

$$trans_{\overline{S}}[\![\ stmt_1 \quad stmt_2\ ...\ stmt_n\ ]\!](\sigma) \quad = \quad \textbf{LET} \quad stmt_1^{*\prime},\, \sigma_1 \quad = \quad trans_S[\![\ stmt_1\ ]\!](\sigma) \quad \textbf{IN}$$
$$\textbf{LET} \quad stmt_2^{*\prime},\, \sigma_2 \quad = \quad trans_S[\![\ stmt_2\ ]\!](\sigma_1) \quad \textbf{IN}$$
$$\ldots$$
$$\textbf{LET} \quad stmt_n^{*\prime},\, \sigma_n \quad = \quad trans_S[\![\ stmt_n\ ]\!](\sigma_{n-1}) \quad \textbf{IN}$$
$$(stmt_1^{*\prime}\ @\ stmt_2^{*\prime}\ @\ ...\ @\ stmt_n^{*\prime},\, \sigma_n)$$

$$\boxed{trans_S \quad : \quad Stmt \quad \rightarrow \quad \Sigma \quad \rightarrow \quad (Stmt\ \textbf{list} \quad \times \quad \Sigma)}$$

$trans_S[\![\ (@expr_1)^*\ \texttt{def}\ id\ (args)\ (\texttt{->}\ expr_2)?\ :\ (\texttt{\#type}:s)?\ stmt^*\ ]\!](\sigma) =$
$\quad ([(@expr_1)^*\ \texttt{def}\ id\ (args)\ (\texttt{->}\ expr_2)?\ :\ (\texttt{\#type}:s)?\ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma).\_\mathbf{1}],\, \sigma)$

$trans_S[\![\ (@expr_1)^*\ \texttt{async def}\ id\ (args)\ (\texttt{->}\ expr_2)?\ :\ (\texttt{\#type}:s)?\ stmt^*\ ]\!](\sigma) =$
$\quad ([(@expr_1)^*\ \texttt{async def}\ id\ (args)\ (\texttt{->}\ expr_2)?\ :\ (\texttt{\#type}:s)?\ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma).\_\mathbf{1}],\, \sigma)$

$trans_S[\![\ (@expr_1)^*\ \texttt{class}\ id\ (expr_2^*\ keyword^*)\ :\ stmt^*\ ]\!](\sigma) =$
$\quad ([(@expr_1)^*\ \texttt{class}\ id\ (expr_2^*\ keyword^*)\ :\ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma).\_\mathbf{1}],\, \sigma) =$

$trans_S[\![\ \texttt{return}\ expr?\ ]\!](\sigma) = ([\texttt{return}\ (trans_E[\![\ expr\ ]\!](\sigma))?],\, \sigma)$

$trans_S[\![\ \texttt{delete}\ expr^*\ ]\!](\sigma) = ([\texttt{delete}\ expr^*],\, \sigma)$

A strict form of assignment statements

$trans_S[\![\ id_r = expr_1\ (expr_{11}\ ...\ expr_{1j}\quad (id_{11} =)?\ expr_{21}\ ...\ (id_{1k} =)?\ expr_{2k})$
$\quad$ *# Minimized Optimizer*
$\quad .id_m\ (expr_{31}\ ...\ expr_{3l}\quad (id_{41} =)?\ expr_{41}\ ...\ (id_{4n} =)?\ expr_{4n})\ (\texttt{\#type}:s)?\ ]\!](\sigma) =$
$\qquad \textbf{IF}\ expr_1\ \texttt{<:}\ \texttt{tensorflow.compat.v1.train.Optimizer}\ \textbf{AND}\ id_m = \texttt{minimize}\ \textbf{THEN}$
$\qquad\quad ([id_r = expr_1\ (expr_{11}\ ...\ expr_{1j}\quad (id_1 =)?\ expr_{21}\ ...\ id_i = expr_{2i}\ \texttt{* hvd.size()}$
$\qquad\qquad ...\ (id_k =)?\ expr_{2k}).id_m\ (expr_{31}\ ...\ expr_{3l}\quad (id_{41} =)?\ expr_{41}\ ...\ (id_{4n} =)?\ expr_{4n})\ (\texttt{\#type}:s)?,$
$\qquad\quad id_r = \texttt{hvd.DistributedOptimizer}(id_r)],\, \sigma[\textbf{"optimizer"} \mapsto id_r])$
$\qquad \textbf{ELSE}$
$\qquad\quad ([id_r = expr_1\ (expr_{11}\quad \texttt{* hvd.size()}\ ...\ expr_{1j}\quad (id_1 =)?\quad expr_{21}\ ...\ (id_k =)?\ expr_{2k})$
$\qquad\qquad .id_m\ (expr_{31}\ ...\ expr_{3l}\quad (id_{41} =)?\ expr_{41}\ ...\ (id_{4n} =)?\ expr_{4n})\ (\texttt{\#type}:s)?,$
$\qquad\quad id_r = \texttt{hvd.DistributedOptimizer}(id_r)],\, \sigma[\textbf{"optimizer"} \mapsto id_r])$

$trans_S[\![\ id_r = expr_1\ (expr_{11}\ ...\ expr_{1n}\quad (id_1 =)?\ expr_{21}\ ...\ (id_k =)?\ expr_{2k})\ (\texttt{\#type}:s)?\ ]\!](\sigma) =$
$\quad$ *# Config Proto*
$\quad \textbf{IF}\ id_t = \textbf{"tensor\_flow"}\ \textbf{AND}\ expr_1 = id_t.\texttt{ConfigProto}\ \textbf{THEN}$
$\qquad (id_r = expr_1\ (expr_{11}\ ...\ expr_{1n}\quad (id_1 =)?\ expr_{21}\ ...\ (id_k =)?\ expr_{2k})\ (\texttt{\#type}:s)?,$
$\qquad\quad \sigma[id_r \mapsto \textbf{"config\_proto"}])$
$\quad$ *# Learning rate scheduler scaling*
$\quad \textbf{IF}\ expr_1\ \texttt{<:}\ \texttt{"tensorflow.keras.optimizers.schedules.PiecewiseConstantDecay"}\ \textbf{THEN}$
$\qquad ([id_r = expr_1\ (expr_{11}\ ...\ expr_{1n}\quad (id_1 =)?\ expr_{21}\ ...\ (id_k =)?\ expr_{2k})\ (\texttt{\#type}:s)?],$
$\qquad\quad \sigma[\textbf{"lr\_scheduler"} \mapsto id_r])$
$\quad \textbf{ELIF}\ expr_1\ \texttt{<:}\ \texttt{tensorflow.keras.optimizers.schedules.LearningRateSchedule}\ \textbf{THEN}$
$\qquad \textbf{IF}\ id_i = \texttt{initial\_learning\_rate}\ \textbf{WHEN}\ 1 \le \text{i} \le \text{k}\ \textbf{THEN}$
$\qquad\quad ([id_r = expr_1\ (expr_{11}\ ...\ expr_{1n}\quad (id_1 =)?\ expr_{21}\ ...\ id_i = expr_{2i}\ \texttt{* hvd.size()}$

$$\ldots \ (id_k \ \texttt{= })? \ expr_{2k}) \ (\texttt{\#type:}s)?], \ \sigma[\textbf{``lr\_scheduler''} \mapsto id_r])$$

ELSE

$$([id_r \ \texttt{= } expr_1 \ (expr_{11} \ \texttt{* hvd.size()} \ \ldots \ expr_{1n} \quad (id_1 \ \texttt{= })? \ expr_{21} \ \ldots \ (id_k \ \texttt{= })? \ expr_{2k}) \ (\texttt{\#type:}s)?],$$
$$\sigma[\textbf{``lr\_scheduler''} \mapsto id_r])$$

ELIF $expr_1 <: $ `tensorflow.compat.v1.train.exponential_decay` **THEN**

IF $id_i \ \texttt{= learning\_rate}$ **WHEN** $1 \le \text{i} \le \text{k}$ **THEN**

$$([id_r \ \texttt{= } expr_1 \ (expr_{11} \ \ldots \ expr_{1n} \quad (id_1 \ \texttt{= })? \ expr_{21} \ \ldots \ id_i \ \texttt{= } expr_{2i} \ \texttt{* hvd.size()}$$
$$\ldots \ (id_k \ \texttt{= })? \ expr_{2k}) \ (\texttt{\#type:}s)?], \ \sigma[\textbf{``lr\_scheduler''} \mapsto id_r])$$

ELSE

$$([id_r \ \texttt{= } expr_1 \ (expr_{11} \ \texttt{* hvd.size()} \ \ldots \ expr_{1n} \quad (id_1 \ \texttt{= })? \ expr_{21} \ \ldots \ (id_k \ \texttt{= })? \ expr_{2k}) \ (\texttt{\#type:}s)?],$$
$$\sigma[\textbf{``lr\_scheduler''} \mapsto id_r])$$

Optimizer learning rate scaling and wrapping

ELIF $expr_1 <: $ `tensorflow.keras.optimizers.Optimizer` **THEN**

IF $id_i \ \texttt{= learning\_rate}$ **WHEN** $1 \le \text{i} \le \text{k}$ **THEN**

$$([id_r \ \texttt{= } expr_1 \ (expr_{11} \ \ldots \ expr_{1n} \quad (id_1 \ \texttt{= })? \ expr_{21} \ \ldots \ id_i \ \texttt{= } expr_{2i} \ \texttt{* hvd.size()}$$
$$\ldots \ (id_k \ \texttt{= })? \ expr_{2k}) \ (\texttt{\#type:}s)?$$
$$id_r \ \texttt{= hvd.DistributedOptimizer(}id_r\texttt{)}], \ \sigma[\textbf{``optimizer''} \mapsto id_r])$$

ELSE

$$([id_r \ \texttt{= } expr_1 \ (expr_{11} \ \texttt{* hvd.size()} \ \ldots \ expr_{1n} \quad (id_1 \ \texttt{= })? \ expr_{21} \ \ldots \ (id_k \ \texttt{= })? \ expr_{2k}) \ (\texttt{\#type:}s)?,$$
$$id_r \ \texttt{= hvd.DistributedOptimizer(}id_r\texttt{)}], \ \sigma[\textbf{``optimizer''} \mapsto id_r])$$

Model related

ELIF $expr_1 <: $ `tensorflow.keras.Model` **THEN**

$$([id_r \ \texttt{= } expr_1 \ (expr_{11} \ \ldots \ expr_{1n} \quad (id_2 \ \texttt{= })? \ expr_{21} \ \ldots \ (id_k \ \texttt{= })? \ expr_{2k})], \ \sigma[\textbf{``model''} \mapsto id_r])$$

ELIF $id_m \ \texttt{= ``model''}$ **AND** $expr_1 \ \texttt{= } id_t\texttt{.evaluate}$ **THEN**

IF $id_i \ \texttt{= verbose}$ **WHEN** $1 \le \text{i} \le \text{k}$ **THEN**

$$([id_r \ \texttt{= } expr_1 \ (expr_{11} \ \ldots \ expr_{1n} \quad (id_1 \ \texttt{= })? \ expr_{21} \ \ldots \ id_i \ \texttt{= 1 if hvd.rank() == 0 else 0}$$
$$\ldots \ (id_k \ \texttt{= })? \ expr_{2k}) \ (\texttt{\#type:}s)?], \ \sigma)$$

ELSE

$$([id_r \ \texttt{= } expr_1 \ (expr_{11} \quad expr_{12} \quad \texttt{1 if hvd.rank() == 0 else 0} \ \ldots \ expr_{1n}$$
$$(id_1 \ \texttt{= })? \ expr_{21} \ \ldots \ (id_k \ \texttt{= })? \ expr_{2k}) \ (\texttt{\#type:}s)?], \ \sigma)$$

Checkpoint

ELIF $\sigma(\textbf{``tensor\_flow''}) \ \texttt{= } id_t$ **AND** $expr_1 \ \texttt{= } id_t\texttt{.train.Checkpoint}$ **THEN**

$$([id_r \ \texttt{= } expr_1 \ (expr_{11} \ \ldots \ expr_{1n} \quad (id_1 \ \texttt{= })? \ expr_{21} \ \ldots \ (id_k \ \texttt{= })? \ expr_{2k}) \ (\texttt{\#type:}s)?],$$
$$\sigma[\textbf{``checkpoint''} \mapsto id_r])$$

Default

ELSE $([id_r \ \texttt{= } trans_E[\![\ expr_1 \ (expr_{11} \ \ldots \ expr_{1n} \quad (id_2 \ \texttt{= })? \ expr_{21} \ \ldots \ (id_k \ \texttt{= })? \ expr_{2k}) \ ]\!](\sigma) \ (\texttt{\#type:}s)?],$
$$\sigma)$$

$trans_S[\![\ expr_0 \ \texttt{= } expr_1 \ (\texttt{\#type:}s)? \ ]\!](\sigma) =$

IF $\sigma(\textbf{``os''}) \ \texttt{= } id_{os}$ **AND** $expr_0 \ \texttt{= } id_{os}\texttt{.environ ['CUDA\_VISIBLE\_DEVICES']}$

THEN $([], \ \sigma)$

A general form of assignment statements except for the strict form

$trans_S[\![\ expr_1^* \ \texttt{= } expr_2 \ (\texttt{\#type:}s)? \ ]\!](\sigma) = ([expr_1^* \ \texttt{= } trans_E[\![\ expr_2 \ ]\!](\sigma) \ (\texttt{\#type:}s)?], \ \sigma)$

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

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

$trans_S[\![\ (\texttt{\#type:}s)? \ \texttt{for} \ expr_1 \ \texttt{in} \ expr_2 \ : \ stmt_1^* \ (\texttt{else} : stmt_2^*)? \ ]\!](\sigma) =$
$\quad ([(\texttt{\#type:}s)? \ \texttt{for} \ expr_1 \ \texttt{in} \ trans_E[\![\ expr_2 \ ]\!](\sigma) \ :$
$\qquad trans_{\overline{S}}[\![\ stmt_1^* \ ]\!](\sigma).\_1 \ (\texttt{else} : trans_{\overline{S}}[\![\ stmt_2^* \ ]\!](\sigma).\_1)?], \sigma)$

$trans_S[\![\ (\texttt{\#type:}s)? \ \texttt{async for} \ expr_1 \ \texttt{in} \ expr_2 \ : \ stmt_1^* \ (\texttt{else} : stmt_2^*)? \ ]\!](\sigma) =$
$\quad ([(\texttt{\#type:}s)? \ \texttt{async for} \ expr_1 \ \texttt{in} \ trans_E[\![\ expr_2 \ ]\!](\sigma) \ :$
$\qquad trans_{\overline{S}}[\![\ stmt_1^* \ ]\!](\sigma).\_1 \ (\texttt{else} : trans_{\overline{S}}[\![\ stmt_2^* \ ]\!](\sigma).\_1)?], \sigma)$

$trans_S[\![\ \texttt{while} \ (expr) \ : \ stmt_1^* \ (\texttt{else} : stmt_2^*)? \ ]\!](\sigma) =$
$\quad ([\texttt{while} \ (trans_E[\![\ expr \ ]\!](\sigma)) \ : \ trans_{\overline{S}}[\![\ stmt_1^* \ ]\!](\sigma).\_1 \ (\texttt{else} : trans_{\overline{S}}[\![\ stmt_2^* \ ]\!](\sigma).\_1)?], \sigma)$

$trans_S[\![\ \texttt{if} \ (expr) \ : \ stmt^* \ (\texttt{else} : stmt^*)? \ ]\!](\sigma) =$
$\quad ([\texttt{if} \ (trans_E[\![\ expr \ ]\!](\sigma)) \ : \ trans_{\overline{S}}[\![\ stmt^* \ ]\!](\sigma).\_1 \ (\texttt{else} : trans_{\overline{S}}[\![\ stmt^* \ ]\!](\sigma).\_1)?], \sigma)$

$trans_S[\![\ (\texttt{\#type:}s)? \ \texttt{with} \ with\_item^* \ : \ stmt^* \ ]\!](\sigma) = ([(\texttt{\#type:}s)? \ \texttt{with} \ with\_item^{*\prime} \ : \ stmt^{*\prime}], \sigma_2)$

$trans_S[\![\ (\texttt{\#type:}s)? \ \texttt{async with} \ with\_item^* \ : \ stmt^* \ ]\!](\sigma) =$
$\quad ([(\texttt{\#type:}s)? \ \texttt{async with} \ with\_item^{*\prime} \ : \ stmt^{*\prime}], \sigma_2)$

$trans_S[\![\ \texttt{match} \ expr \ : \ match\_case^* \ ]\!](\sigma) =$
$\quad ([\texttt{match} \ trans_E[\![\ expr \ ]\!](\sigma) \ : \ (trans_C[\![match\_case]\!](\sigma))^*], \sigma)$

$trans_S[\![\ \texttt{raise} \ expr_1? \ (\texttt{from} \ expr_2)? \ ]\!](\sigma) = ([\texttt{raise} \ expr_1? \ (\texttt{from} \ expr_2)?], \sigma)$

$trans_S[\![\ \texttt{try} \ : \ stmt_1^* \ exc\_handler^* \ (\texttt{else} : stmt_2^*)? \ (\texttt{finally} : stmt_3^*)? \ ]\!](\sigma) =$
$\quad ([\texttt{try} \ : \ trans_{\overline{S}}[\![\ stmt_1^* \ ]\!](\sigma).\_1 \ (trans_H[\![exc\_handler]\!](\sigma))^*$
$\quad (\texttt{else} : trans_{\overline{S}}[\![\ stmt_2^* \ ]\!](\sigma).\_1)? \ (\texttt{finally} : trans_{\overline{S}}[\![\ stmt_3^* \ ]\!](\sigma).\_1)?], \sigma)$

$trans_S[\![\ \texttt{assert} \ expr_1 \ expr_2? \ ]\!](\sigma) = ([\texttt{assert} \ trans_E[\![\ expr_1 \ ]\!](\sigma) \ expr_2?], \sigma)$

$trans_S[\![\ \texttt{import} \ alias^* \ ]\!](\sigma) =$
$\quad \textbf{IF} \ \sigma_1 \ \backslash \ \sigma = [\text{``}\textbf{tensor\_flow\_compat''} \mapsto id] \ \ \textbf{THEN}$
$\qquad ([\texttt{import} \ alias^*,$
$\qquad \texttt{import horovod.tensorflow as hvd},$
$\qquad \texttt{hvd.init()}], \sigma_1)$
$\quad \textbf{ELSE} \ ([\texttt{import} \ alias^*], \sigma_1)$

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

$trans_S[\![\ \texttt{global} \ id^* \ ]\!](\sigma) = ([\texttt{global} \ id^*], \sigma)$

$trans_S[\![$ `nonlocal` id^* $]\!](\sigma) = ([$`nonlocal` $id^*], \sigma)$

A strict form of expr statements

$trans_S[\![$ $expr_1$ $(expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k})$ $]\!](\sigma) =$

    # Config

   IF id_t $=$ $\sigma($**"tensor_flow"**$)$ **AND**

     $expr_1$ $=$ id_t.`config.experimental.set_visible_devices` **THEN** $([], \sigma)$

    # Root Rank Blocking

   ELIF id_m $=$ $\sigma($**"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 $=$ $\sigma($**"checkpoint"**$)$ **AND** $expr_1$ $=$ id_c.`save` **OR**

     id_t $=$ $\sigma($**"tensor_flow"**$)$ **AND** $expr_1$ $=$ id_t.`print` **OR**

     $expr_1$ $=$ `print` **THEN**

       $([$`if hvd.rank() == 0:` $expr_1$ $(expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k})], \sigma)$

   ELSE

      $[trans_E[\![$ $expr_1$ $(expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k})$ $]\!](\sigma)], \sigma)$

A general form of expr statements except for the strict form

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

$trans_S[\![$ `pass` $]\!](\sigma) = ([$`pass`$], \sigma)$

$trans_S[\![$ `break` $]\!](\sigma) = ([$`break`$], \sigma)$

$trans_S[\![$ `continue` $]\!](\sigma) = ([$`continue`$], \sigma)$

$trans_{\overline{E}}[\![$ $expr_1$ $expr_2$ ... $expr_n$ $]\!](\sigma) = trans_E[\![$ $expr_1$ $]\!](\sigma)$ $trans_E[\![$ $expr_2$ $]\!](\sigma)$ ... $trans_E[\![$ $expr_n$ $]\!](\sigma)$

$trans_E$: $Expr$ $\rightarrow$ Σ $\rightarrow$ $Expr$

$trans_E[\![$ $expr_1$ $boolop$ $expr_2$ $]\!](\sigma) = trans_E[\![$ $expr_1$ $]\!](\sigma)$ $boolop$ $trans_E[\![$ $expr_2$ $]\!](\sigma)$

$trans_E[\![$ $expr_1$ `:=` $expr_2$ $]\!](\sigma) = expr_1$ `:=` $trans_E[\![$ $expr_2$ $]\!](\sigma)$

$trans_E[\![$ $expr_1$ $binop$ $expr_2$ $]\!](\sigma) = trans_E[\![$ $expr_1$ $]\!](\sigma)$ $binop$ $trans_E[\![$ $expr_2$ $]\!](\sigma)$

$trans_E[\![$ $unop$ $expr$ $]\!](\sigma) = unop$ $trans_E[\![$ $expr$ $]\!](\sigma)$

$trans_E[\![$ `lambda` $args$ `:` $expr$ $]\!](\sigma) =$ `lambda` $args$ `:` $trans_E[\![$ $expr$ $]\!](\sigma)$

$trans_E[\![\ expr_1\ \ \texttt{if}\ \ expr_2\ \ \texttt{else}\ \ expr_3\ ]\!](\sigma) =$
    $trans_E[\![\ expr_1\ ]\!](\sigma)\ \ \texttt{if}\ \ trans_E[\![\ expr_2\ ]\!](\sigma)\ \ \texttt{else}\ \ trans_E[\![\ expr_3\ ]\!](\sigma)$

$trans_E[\![\ expr_1\ \ \texttt{if}\ \ expr_2\ \ \texttt{else}\ \ expr_3\ ]\!](\sigma) =$
    $trans_E[\![\ expr_1\ ]\!](\sigma)\ \ \texttt{if}\ \ trans_E[\![\ expr_2\ ]\!](\sigma)\ \ \texttt{else}\ \ trans_E[\![\ expr_3\ ]\!](\sigma)$

$trans_E[\![\ \{(expr_1\ :\ expr_2)^*\ (**expr_3)^*\}\ ]\!](\sigma) = \{(expr_1\ :\ trans_E[\![expr_2]\!](\sigma))^*\ (**expr_3)^*\}$

$trans_E[\![\ \{expr^*\}\ ]\!](\sigma) = \{(trans_E[\![expr]\!](\sigma))^*\}$

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

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

$trans_E[\![\ \{expr_1{:}expr_2\ \ comprehension^*\}\ ]\!](\sigma) = \{expr_1{:}trans_E[\![\ expr_2\ ]\!](\sigma)\ (trans_O[\![comprehension]\!](\sigma))^*\}$

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

$trans_E[\![\ [expr\ \ comprehension^*]\ ]\!](\sigma) = [trans_E[\![\ expr\ ]\!](\sigma)\ (trans_O[\![comprehension]\!](\sigma))^*]$

$trans_E[\![\ (expr\ \ comprehension^*)\ ]\!](\sigma) = (trans_E[\![\ expr\ ]\!](\sigma)\ (trans_O[\![comprehension]\!](\sigma))^*)$

$trans_E[\![\ \texttt{await}\ \ expr\ ]\!](\sigma) = \texttt{await}\ \ trans_E[\![\ expr\ ]\!](\sigma)$

$trans_E[\![\ \texttt{yield}\ \ expr?\ ]\!](\sigma) = \texttt{yield}\ \ (trans_E[\![\ expr\ ]\!](\sigma))?$

$trans_E[\![\ \texttt{yield from}\ \ expr\ ]\!](\sigma) = \texttt{yield from}\ \ trans_E[\![\ expr\ ]\!](\sigma)$

$trans_E[\![\ expr_1\ (compop\ expr_2)^*\ ]\!](\sigma) = trans_E[\![\ expr_1\ ]\!](\sigma)\ (compop\ trans_E[\![expr_2]\!](\sigma))^*$

$trans_E[\![\ expr_1\ (expr_{11}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ ]\!](\sigma) =$
    $trans_E[\![\ expr_1\ ]\!](\sigma)\ (trans_E[\![\ expr_{11}\ ]\!](\sigma)\ ...\ trans_E[\![\ expr_{1n}\ ]\!](\sigma)$
      $(id_1\ \texttt{=}\ )?\ trans_E[\![\ expr_{21}\ ]\!](\sigma)\ ...\ (id_k\ \texttt{=}\ )?\ trans_E[\![\ expr_{2k}\ ]\!](\sigma))$

$trans_E[\![\ \{expr_1\ \ \ (!i)?\ \ \ (:expr_2)?\}\ ]\!](\sigma) = \{expr_1\ \ \ (!i)?\ \ \ (:expr_2)?\}$

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

$trans_E[\![\ constant\ ]\!](\sigma) = constant$

$trans_E[\![\ expr.id\ ]\!](\sigma) = (trans_E[\![\ expr\ ]\!](\sigma)).id$

$trans_E[\![\ expr_1[expr_2]\ ]\!](\sigma) = trans_E[\![\ expr_1\ ]\!](\sigma)[trans_E[\![\ expr_2\ ]\!](\sigma)]$

$trans_E[\![\ *expr\ ]\!](\sigma) = *expr$

$trans_E[\![\ **expr\ ]\!](\sigma) = **expr$

$trans_E[\![\ id\ ]\!](\sigma) = id$

$trans_E[\![\ expr_1?\ (:expr_2)?\ (:expr_3)?\ ]\!](\sigma) = trans_E[\![\ expr_1\ ]\!](\sigma)?\ (:trans_E[\![\ expr_2\ ]\!](\sigma))?\ (:trans_E[\![\ expr_3\ ]\!](\sigma))?$

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

$trans_O[\![\ \texttt{for}\ expr_1\ \texttt{in}\ expr_2\ (\texttt{if}\ expr_3)^*\ ]\!](\sigma) = \texttt{for}\ expr_1\ \texttt{in}\ trans_E[\![\ expr_2\ ]\!](\sigma)\ (\texttt{if}\ trans_E[\![expr_3]\!](\sigma))^*$

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

$trans_H[\![\ \texttt{except}\ expr?\ (\texttt{as}\ id)?\ :\ stmt^*\ ]\!](\sigma) = \texttt{except}\ expr?\ (\texttt{as}\ id)?\ :\ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma).\_\mathbf{1}$

$\boxed{trans_{\overline{A}}\ :\ Alias\ \mathbf{list}\ \to\ \Sigma\ \to\ \Sigma}$

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

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

$trans_A[\![\ id\ ]\!](\sigma) =$
   LET $id = \texttt{tensorflow.compat.v1}$ **THEN** $\sigma[\text{“\textbf{tensor\_flow\_compat}”}\mapsto id]$
   LET $id = \texttt{tensorflow}$ **THEN** $\sigma[\text{“\textbf{tensor\_flow}”}\mapsto id]$
   LET $id = \texttt{os}$ **THEN** $\sigma[\text{“\textbf{os}”}\mapsto id]$
   ELSE σ

$trans_A[\![\ id_1\ \texttt{as}\ id_2\ ]\!](\sigma) =$
   LET $id_1 = \texttt{tensorflow.compat.v1}$ **THEN** $\sigma[\text{“\textbf{tensor\_flow\_compat}”}\mapsto id_2]$
   LET $id_1 = \texttt{tensorflow}$ **THEN** $\sigma[\text{“\textbf{tensor\_flow}”}\mapsto id_2]$
   LET $id_1 = \texttt{os}$ **THEN** $\sigma[\text{“\textbf{os}”}\mapsto id_2]$
   ELSE σ

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

$\boxed{trans_{\overline{W}}\ :\ WithItem\ \mathbf{list}\ \to\ \Sigma\ \to\ (WithItem\ \mathbf{list}\ \times\ \Sigma)}$

$trans_{\overline{W}} [\![\; with\_item_1 \; with\_item_2 \; ... \; with\_item_n \;]\!](\sigma) =$
    LET $with\_item_1', \sigma_1 \; = \; trans_W [\![\; with\_item_1 \;]\!](\sigma) \quad$ **IN**
    LET $with\_item_2', \sigma_2 \; = \; trans_W [\![\; with\_item_2 \;]\!](\sigma_1) \quad$ **IN**
    ...
    LET $with\_item_n', \sigma_n \; = \; trans_W [\![\; with\_item_n \;]\!](\sigma_{n-1}) \quad$ **IN**
    $(with\_item_1' \; \texttt{::} \; with\_item_2' \; \texttt{::} \; ... \; \texttt{::} \; [with\_item_n'], \; \sigma_n)$

$\boxed{trans_W \quad : \quad WithItem \;\; \rightarrow \;\; \Sigma \;\; \rightarrow \;\; (WithItem \;\; \times \;\; \Sigma)}$
$trans_W [\![\; expr \;]\!](\sigma) = (trans_E [\![\; expr \;]\!](\sigma), \; \sigma)$

$trans_W [\![\; expr_1 \; \texttt{as} \; expr_2 \;]\!](\sigma) =$
    $(trans_E [\![\; expr_1 \;]\!](\sigma) \; \texttt{as} \; expr_2, \; \sigma)$

$\boxed{trans_C \quad : \quad MatchCase \;\; \rightarrow \;\; \Sigma \;\; \rightarrow \;\; MatchCase}$
$trans_C [\![\; \texttt{case} \; pattern \; (\texttt{if} \; expr)? \; : \; stmt^* \;]\!](\sigma) =$
    $\texttt{case} \; trans_P [\![\; pattern \;]\!](\sigma) \;\; (\texttt{if} \; trans_E [\![\; expr \;]\!](\sigma))? \; : \;\; trans_{\overline{S}} [\![\; stmt^* \;]\!](\sigma).\_\mathbf{1}$

$\boxed{trans_P \quad : \quad Pattern \;\; \rightarrow \;\; \Sigma \;\; \rightarrow \;\; Pattern}$
$trans_P [\![\; expr \;]\!](\sigma) = trans_E [\![\; expr \;]\!](\sigma)$

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

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

$trans_P [\![\; \texttt{*}(id)? \;]\!](\sigma) = \texttt{*}(id)?$

$trans_P [\![\; \{(expr \; : \; pattern)^* \; id?\} \;]\!](\sigma) = \{(expr \; : \; trans_P [\![pattern]\!](\sigma))^* \; id?\}$

$trans_P [\![\; expr \; (pattern_1^* \; (id = pattern_2)^*) \;]\!](\sigma) = expr \;\; (trans_P [\![pattern_1]\!](\sigma)^* \; (id = trans_P [\![pattern_2]\!](\sigma))^*)$

$trans_P [\![\; (pattern \;\; \texttt{as})? \; id \;]\!](\sigma) = (trans_P [\![\; pattern \;]\!](\sigma) \; \texttt{as})? \; id$

$trans_P [\![\; pattern_1 \; | \; pattern_2 \;]\!](\sigma) = trans_P [\![\; pattern_1 \;]\!](\sigma) \; | \; trans_P [\![\; pattern_2 \;]\!](\sigma)$

$trans_P [\![\; \_ \;]\!](\sigma) = \_$

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

$$
\begin{array}{llll}
Summary & ::= & \texttt{ModuleSummary}\ id\ \sigma_\top\ tl & (\textsc{ModuleSummary}) \\
& & \texttt{FuncSummary}\ tl & (\textsc{FuncSummary}) \\
& & \texttt{ClassSummary}\ arg & (\textsc{ClassSummary}) \\
& & \texttt{ValueSummary}\ s & (\textsc{ValueSummary}) \\
arg & ::= & Model\ |\ \bot & (\textsc{Arg}) \\
tl & ::= & GradTape\ |\ Optimizer\ |\ \bot & (\textsc{TrainingLoop}) \\
\sigma_\top & ::= & \sigma\ \cup\ \top &
\end{array}
$$

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

3.2.2 Training Loop

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

$summary_M[\![\ id\ stmt^*\ type\_ignore\ ]\!] =$
    LET $\sigma, tl\ =\ summary_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)$ **IN**
    $\texttt{ModuleSummary}\ id\ \sigma\ tl$

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

$summary_{\overline{S}}[\![\ stmt_1\ stmt_2 ... stmt_n\ ]\!](\sigma)\ =$ **LET** $\sigma_1, tl_1\ =\ summary_S[\![\ stmt_1\ ]\!](\sigma)$ **IN**
                                                                          LET $\sigma_2, tl_2\ =\ summary_{\overline{S}}[\![\ stmt_2\ ]\!](\sigma_1)$ **IN**

                                                                          . . .

                                                                          LET $\sigma_n, tl_n\ =\ summary_{\overline{S}}[\![\ stmt_n\ ]\!](\sigma_{n-1})$ **IN**
                                                                          $(\sigma_n, tl_1 \sqcup tl_2 \sqcup\ ... \ tl_n)$

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

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

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

$summary_S[\![\ (@expr)^* \ \texttt{class} \ id \ (expr_{11} \ ... \ expr_{1n} \ \ (id_1 \ \texttt{=} \ )? \ expr_{21} \ ... \ (id_k \ \texttt{=} \ )? \ expr_{2k}) \ \ : \ \ stmt^* \ ]\!](\sigma) =$

    IF $\ expr_{1i} \ = \ id_c.\texttt{keras.Model} \ $ **WHEN** $\ 1 \le i \le n \ $ **AND**

        $\sigma(id_c) \ = \ \texttt{ModuleSummary} \ \text{Id("tensorflow")} \ \ [] \ \ \bot \ $ **THEN** $\ (\sigma[id \mapsto \texttt{ClassSummary} \ Model], \bot)$

    ELIF $\ expr_{2i} \ = \ id_c.\texttt{keras.Model} \ $ **WHEN** $\ 1 \le i \le k \ $ **AND**

        $\sigma(id_c) \ = \ \texttt{ModuleSummary} \ \text{Id("tensorflow")} \ \ [] \ \ \bot \ $ **THEN** $\ (\sigma[id \mapsto \texttt{ClassSummary} \ Model], \bot)$

    ELSE $\ (\sigma[id \mapsto \texttt{ClassSummary} \ \bot], \bot)$

A strict form of assignment statements

$summary_S[\![\ id_r \ \texttt{=} \ id_c \ \texttt{()} \ (\texttt{\#type:}s)? \ ]\!](\sigma) =$

    IF $\ \sigma(id_c) \ = \ \texttt{ClassSummary} \ Model \ $ **THEN** $\ (\sigma[id_r \mapsto \texttt{ValueSummary} \ \textbf{"model\_instance"}], \bot)$

    ELSE $\ (\sigma, \bot)$

$summary_S[\![\ (\texttt{\#type:}s)? \ \texttt{for} \ expr_1 \ \texttt{in} \ expr_2 \ \ : \ \ stmt_1^* \ \ (\texttt{else} \ : \ stmt_2^*)? \ ]\!](\sigma) =$

    LET $\ __, tl \ = \ summary_{\overline{S}}[\![\ stmt_1^* \ ]\!](\sigma) \ $ **IN**

    (σ, tl)

$summary_S[\![\ (\texttt{\#type:}s)? \ \texttt{async for} \ expr_1 \ \texttt{in} \ expr_2 \ \ : \ \ stmt_1^* \ \ (\texttt{else} \ : \ stmt_2^*)? \ ]\!](\sigma) =$

    LET $\ __, tl \ = \ summary_{\overline{S}}[\![\ stmt_1^* \ ]\!](\sigma) \ $ **IN**

    (σ, tl)

$summary_S[\![\ \texttt{while} \ (expr) \ \ : \ \ stmt_1^* \ \ (\texttt{else} \ : \ stmt_2^*)? \ ]\!](\sigma) =$

    LET $\ __, tl \ = \ summary_{\overline{S}}[\![\ stmt_1^* \ ]\!](\sigma) \ $ **IN**

    (σ, tl)

$summary_S[\![\ (\texttt{\#type:}s)? \ \texttt{with} \ with\_item^* \ \ : \ \ stmt^* \ ]\!](\sigma) =$

    LET $\ __, tl \ = \ summary_{\overline{S}}[\![\ stmt^* \ ]\!](\sigma) \ $ **IN**

    $(\sigma, summary_{\overline{W}}[\![\ with\_item^* \ ]\!](\sigma) \sqcup \ tl)$

$summary_S[\![\ (\texttt{\#type:}s)? \ \texttt{async with} \ with\_item^* \ \ : \ \ stmt^* \ ]\!](\sigma) =$

    LET $\ __, tl \ = \ summary_{\overline{S}}[\![\ stmt^* \ ]\!](\sigma) \ $ **IN**

    $(\sigma, summary_{\overline{W}}[\![\ with\_item^* \ ]\!](\sigma) \sqcup \ tl)$

$summary_S[\![\ \texttt{try} \ : \ stmt_1^* \ \ exc\_handler^* \ \ (\texttt{else} \ : \ stmt_2^*)? \ \ (\texttt{finally} \ : \ stmt_3^*)? \ ]\!](\sigma) =$

    $(\sigma, summary_{\overline{S}}[\![\ stmt_1^* \ ]\!](\sigma))$

$summary_S[\![\ \texttt{import} \ alias^* \ ]\!](\sigma) = (summary_{\overline{A}}[\![\ alias^* \ ]\!](\sigma), \bot)$

$summary_S[\![\ \texttt{from} \ 0 \ \ id^* \ \ \texttt{import} \ alias^* \ ]\!](\sigma) =$

    LET $\ \sigma' \ = \ summary_{\overline{A}}[\![\ alias^* \ ]\!](\sigma) \ $ **IN**

    LET $\ [id_2 \mapsto \texttt{ModuleSummary} \ id_2 \ \sigma_2 \ \bot, \ \ id_2 \mapsto \texttt{ModuleSummary} \ id_2 \ \sigma_2 \ \bot,$

       $... \ id_n \mapsto \texttt{ModuleSummary} \ id_n \ \sigma_n \ \bot] \ = \ \sigma' \ \setminus \ \sigma \ $ **IN**

    $\sigma \ \texttt{++} \ \ [id_2 \mapsto \texttt{ModuleSummary} \ (id^*\texttt{+}id_2) \ \sigma_2 \ \bot, \ \ id_2 \mapsto \texttt{ModuleSummary} \ (id^*\texttt{+}id_2) \ \sigma_2 \ \bot,$

       $... \ id_n \mapsto \texttt{ModuleSummary} \ (id^*\texttt{+}id_n) \ \sigma_n \ \bot]$

$summary_S[\![\ expr \ ]\!](\sigma) = (\sigma, summary_E[\![\ expr \ ]\!](\sigma))$

$summary_S[\![\ stmt\ ]\!](\sigma) = (\sigma,\ \bot)$

$\boxed{summary_{\overline{A}}\quad :\quad Alias\ \textbf{list}\quad \rightarrow\quad \Sigma\quad \rightarrow\quad \Sigma}$

$summary_{\overline{A}}[\![\ alias_1\ ...\ alias_{n-1}\ alias_n\ ]\!](\sigma) =$
$\quad summary_A[\![\ alias_n\ ]\!](summary_A[\![\ alias_{n-1}\ ]\!](...\ summary_A[\![\ alias_1\ ]\!](\sigma)\ ...))$

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

$trans_A[\![\ id_1\ id_2\ ...\ id_n\ ]\!](\sigma) =$
$\quad \textbf{LET}\ \ \sigma_n = [id_n \mapsto \texttt{ModuleSummary}\ id_n\quad \top\quad \bot]$
$\quad \textbf{LET}\ \ \sigma_{n-1} = [id_{n-1} \mapsto \texttt{ModuleSummary}\ id_{n-1}\quad \sigma_n\quad \bot]$
$\quad \textbf{...}$
$\quad \textbf{LET}\ \ \sigma_2 = [id_2 \mapsto \texttt{ModuleSummary}\ id_2\quad \sigma_3\quad \bot]$
$\quad \textbf{LET}\ \ \sigma_1 = [id_1 \mapsto \texttt{ModuleSummary}\ id_1\quad \sigma_2\quad \bot]$
$\quad \sigma\ \ ++\ \ \sigma_1$

$\boxed{summary_{\overline{W}}\quad :\quad WithItem\ \textbf{list}\quad \rightarrow\quad \Sigma\quad \rightarrow\quad tl}$

$summary_{\overline{W}}[\![\ with\_item_1\ with\_item_2\ ...\ with\_item_n\ ]\!](\sigma) =$
$\quad summary_W[\![\ with\_item_1\ ]\!](\sigma)\ \sqcup\ \ summary_W[\![\ with\_item_2\ ]\!](\sigma)\ \sqcup\ \textbf{...}\quad summary_W[\![\ with\_item_n\ ]\!](\sigma)$

$\boxed{summary_W\quad :\quad WithItem\quad \rightarrow\quad \Sigma\quad \rightarrow\quad tl}$

$summary_W[\![\ expr_1\ \texttt{as}\ \ expr_2?\ ]\!](\sigma) = summary_E[\![\ expr_1\ ]\!](\sigma)$

$\boxed{summary_E\quad :\quad Expr\quad \rightarrow\quad \Sigma\quad \rightarrow\quad tl}$

A strict form of call expression

$summary_E[\![\ expr_1\ (expr_{11}\ ...\ expr_{1n}\quad (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ ]\!](\sigma) =$
$\quad \textbf{IF}\ \ \sigma(id_r)\ \ =\ \texttt{ValueSummary}\ \texttt{"model\_instance"}\ \ \textbf{AND}$
$\qquad expr_1\ \ =\ id_r.\texttt{fit}\ \ \textbf{THEN}\ Optimizer$
$\quad \textbf{ELIF}\ \ \sigma(id_r)\ \ =\ \texttt{ModuleSummary}\ \text{Id("tensorflow")}\ \ \sigma'\ \ tl\ \textbf{AND}$
$\qquad expr_1\ \ =\ id_r.\texttt{GradientTape}\ \ \textbf{THEN}\ GradTape$
$\quad \textbf{ELIF}\ \ \sigma(id_r)\ \ =\ \texttt{FuncSummary}\ tl\ \textbf{AND}$
$\qquad expr_1\ \ =\ \texttt{idr}\ \ \textbf{THEN}\ tl$
$\quad \textbf{ELSE}\ \bot$

$summary_E[\![\ expr\ ]\!](\sigma) = \bot$