Code Transformation for Distributed Python ML Code

1 Python Abstract Syntax

module	$::=$	$\mathit{stmt}^*\ \ \mathit{type\_ignore}$	(ModuleDef)
stmt	$::=$	$(@\mathit{expr})^*\ \texttt{def}\ \mathit{id}\ \texttt{(}\mathit{args}\texttt{)}\ \ (\texttt{->}\ \mathit{expr})?\ \ \texttt{:}\ \ (\texttt{\#type:}s)?\ \ \mathit{stmt}^*$	(FunDef)
	$\mid$	$(@\mathit{expr})^*\ \texttt{async def}\ \mathit{id}\ \texttt{(}\mathit{args}\texttt{)}\ \ (\texttt{->}\ \mathit{expr})?\ \ \texttt{:}\ \ (\texttt{\#type:}s)?\ \ \mathit{stmt}^*$	(AsyncFunDef)
	$\mid$	$(@\mathit{expr})^*\ \texttt{class}\ \mathit{id}\ \texttt{(}\mathit{expr}^*\ \mathit{keyword}^*\texttt{)}\ \ \texttt{:}\ \ \mathit{stmt}^*$	(ClassDef)
	$\mid$	$\texttt{return}\ \mathit{expr}?$	(Return)
	$\mid$	$\texttt{delete}\ \mathit{expr}^*$	(Delete)
	$\mid$	$\mathit{expr}^*\ \ \texttt{=}\ \ \mathit{expr}\ \ (\texttt{\#type:}s)?$	(Assign)
	$\mid$	$\mathit{expr}\ \ \mathit{binop}\ \texttt{=}\ \ \mathit{expr}$	(AugAssign)
	$\mid$	$\mathit{expr}\ \ \texttt{:}\ \ \mathit{expr}\ \ (\texttt{=}\ \mathit{expr})?$	(AnnAssign)
	$\mid$	$(\texttt{\#type:}s)?\ \ \texttt{for}\ \mathit{expr}\ \texttt{in}\ \mathit{expr}\ \ \texttt{:}\ \ \mathit{stmt}^*\ \ (\texttt{else}\ \ \texttt{:}\ \ \mathit{stmt}^*)?$	(ForLoop)
	$\mid$	$(\texttt{\#type:}s)?\ \ \texttt{async for}\ \mathit{expr}\ \texttt{in}\ \mathit{expr}\ \ \texttt{:}\ \ \mathit{stmt}^*\ \ (\texttt{else}\ \ \texttt{:}\ \ \mathit{stmt}^*)?$	(AsyncForLoop)
	$\mid$	$\texttt{while}\ \ \texttt{(}\mathit{expr}\texttt{)}\ \ \texttt{:}\ \ \mathit{stmt}^*\ \ (\texttt{else}\ \ \texttt{:}\ \ \mathit{stmt}^*)?$	(WhileLoop)
	$\mid$	$\texttt{if}\ \ \texttt{(}\mathit{expr}\texttt{)}\ \ \texttt{:}\ \ \mathit{stmt}^*\ \ (\texttt{else}\ \ \texttt{:}\ \ \mathit{stmt}^*)?$	(If)
	$\mid$	$(\texttt{\#type:}s)?\ \ \texttt{with}\ \mathit{with\_item}^*\ \ \texttt{:}\ \ \mathit{stmt}^*$	(With)
	$\mid$	$(\texttt{\#type:}s)?\ \ \texttt{async with}\ \mathit{with\_item}^*\ \ \texttt{:}\ \ \mathit{stmt}^*$	(AsyncWith)
	$\mid$	$\texttt{match}\ \mathit{expr}\ \ \texttt{:}\ \ \mathit{match\_case}^*$	(Match)
	$\mid$	$\texttt{raise}\ \mathit{expr}?\ \ (\texttt{from}\ \mathit{expr})?$	(Raise)
	$\mid$	$\texttt{try}\ \ \texttt{:}\ \ \mathit{stmt}^*\ \ \mathit{exc\_handler}^*\ \ (\texttt{else}\ \ \texttt{:}\ \ \mathit{stmt}^*)?\ \ (\texttt{finally}\ \ \texttt{:}\ \ \mathit{stmt}^*)?$	(Try)
	$\mid$	$\texttt{assert}\ \mathit{expr}\ \ \mathit{expr}?$	(Assert)
	$\mid$	$\texttt{import}\ \mathit{alias}^*$	(Import)
	$\mid$	$\texttt{from}\ i\ \ \mathit{id}?\ \texttt{import}\ \mathit{alias}^*$	(ImportFrom)
	$\mid$	$\texttt{global}\ \mathit{id}^*$	(Global)
	$\mid$	$\texttt{nonlocal}\ \mathit{id}^*$	(NonLocal)
	$\mid$	expr	(ExprStmt)
	$\mid$	$\texttt{pass}$	(Pass)
	$\mid$	$\texttt{break}$	(Break)
	$\mid$	$\texttt{continue}$	(Continue)

<table>
<tr><td>expr</td><td>::=</td><td>expr boolop expr</td><td>(BoolOp)</td></tr>
<tr><td></td><td>|</td><td>expr := expr</td><td>(NamedExpr)</td></tr>
<tr><td></td><td>|</td><td>expr binop expr</td><td>(BinaryOp)</td></tr>
<tr><td></td><td>|</td><td>unop expr</td><td>(UnaryOp)</td></tr>
<tr><td></td><td>|</td><td>lambda args : expr</td><td>(Lambda)</td></tr>
<tr><td></td><td>|</td><td>expr if expr else expr</td><td>(IfExpr)</td></tr>
<tr><td></td><td>|</td><td>{(expr : expr)* (**expr)*}</td><td>(Dictionary)</td></tr>
<tr><td></td><td>|</td><td>{expr*}</td><td>(Set)</td></tr>
<tr><td></td><td>|</td><td>[expr*]</td><td>(List)</td></tr>
<tr><td></td><td>|</td><td>(expr*)</td><td>(Tuple)</td></tr>
<tr><td></td><td>|</td><td>{expr:expr comprehension*}</td><td>(DictionaryComp)</td></tr>
<tr><td></td><td>|</td><td>{expr comprehension*}</td><td>(SetComp)</td></tr>
<tr><td></td><td>|</td><td>[expr comprehension*]</td><td>(ListComp)</td></tr>
<tr><td></td><td>|</td><td>(expr comprehension*)</td><td>(GeneratorComp)</td></tr>
<tr><td></td><td>|</td><td>await expr</td><td>(Await)</td></tr>
<tr><td></td><td>|</td><td>yield expr?</td><td>(Yield)</td></tr>
<tr><td></td><td>|</td><td>yield from expr</td><td>(YieldFrom)</td></tr>
<tr><td></td><td>|</td><td>expr (compop expr)*</td><td>(CompOp)</td></tr>
<tr><td></td><td>|</td><td>expr (expr* keyword*)</td><td>(Call)</td></tr>
<tr><td></td><td>|</td><td>{expr (!i)? (:expr)?}</td><td>(FormattedValue</td></tr>
<tr><td></td><td>|</td><td>expr*</td><td>(JoinedStr)</td></tr>
<tr><td></td><td>|</td><td>constant</td><td>(Constant)</td></tr>
<tr><td></td><td>|</td><td>expr.id</td><td>(Attribute)</td></tr>
<tr><td></td><td>|</td><td>expr[expr]</td><td>(Subscript)</td></tr>
<tr><td></td><td>|</td><td>*expr</td><td>(Starred)</td></tr>
<tr><td></td><td>|</td><td>**expr</td><td>(DoubleStarred)</td></tr>
<tr><td></td><td>|</td><td>id</td><td>(Name)</td></tr>
<tr><td></td><td>|</td><td>expr? (:expr)? (:expr)?</td><td>(Slice)</td></tr>
<tr><td>boolop</td><td>::=</td><td>and | or</td><td>(BoolOperator)</td></tr>
<tr><td>binop</td><td>::=</td><td>+ | - | * | @ | / | ** | << | >> | | | ^ | & | // | %</td><td>(BinOperator)</td></tr>
<tr><td>unop</td><td>::=</td><td>~ | not | + | -</td><td>(UnOperator)</td></tr>
<tr><td>compop</td><td>::=</td><td>== | != | < | <= | > | >= | is | is not | in | not in</td><td>(CompOperator)</td></tr>
<tr><td>comprehension</td><td>::=</td><td>for expr in expr (if expr)*</td><td>(Comprehension)</td></tr>
<tr><td></td><td>|</td><td>async for expr in expr (if expr)*</td><td>(AsyncComprehen</td></tr>
<tr><td>exc_handler</td><td>::=</td><td>except expr? (as id)? : stmt*</td><td>(ExcHandler)</td></tr>
<tr><td>args</td><td>::=</td><td>(arg (= expr)?)*, (arg (= expr)?)*, arg?, (arg (= expr)?)*, arg?</td><td>(Arguments)</td></tr>
<tr><td>arg</td><td>::=</td><td>id expr? s?</td><td>(Argument)</td></tr>
<tr><td>keyword</td><td>::=</td><td>id? = expr</td><td>(Keyword)</td></tr>
<tr><td>alias</td><td>::=</td><td>id (.id)* (as id)?</td><td>(Alias)</td></tr>
<tr><td>with_item</td><td>::=</td><td>expr (as expr)?</td><td>(WithItem)</td></tr>
</table>

$$
\begin{array}{llll}
\mathit{match\_case} & ::= & \texttt{case}\ \mathit{pattern}\ (\texttt{if}\ \mathit{expr})?\ \texttt{:}\ \ \mathit{stmt}^* & (\textsc{MatchCase}) \\
\mathit{pattern} & ::= & \mathit{expr} & (\textsc{MatchValue}) \\
 & | & \mathit{constant} & (\textsc{MatchSingleton}) \\
 & | & \texttt{[}\mathit{pattern}^*\texttt{]} & (\textsc{MatchSequence}) \\
 & | & \texttt{*}(\mathit{id})? & (\textsc{MatchStar}) \\
 & | & \{(\mathit{expr}\ \texttt{:}\ \mathit{pattern})^*\ \ \mathit{id}?\} & (\textsc{MatchMapping}) \\
 & | & \mathit{expr}\ \ \texttt{(}\mathit{pattern}^*\ (\mathit{id} = \mathit{pattern})^*\texttt{)} & (\textsc{MatchClass}) \\
 & | & (\mathit{pattern}\ \ \texttt{as})?\ \mathit{id} & (\textsc{MatchAs}) \\
 & | & \mathit{pattern}\ \ \texttt{|}\ \mathit{pattern} & (\textsc{MatchOr}) \\
 & | & \texttt{\_} & (\textsc{MatchWildcard}) \\
\mathit{constant} & ::= & \texttt{None} & (\textsc{NoneLiteral}) \\
 & | & i & (\textsc{IntLiteral}) \\
 & | & f & (\textsc{FloatLiteral}) \\
 & | & c & (\textsc{ComplexLiteral}) \\
 & | & s & (\textsc{StringLiteral}) \\
 & | & b & (\textsc{BooleanLiteral}) \\
 & | & \texttt{(}\mathit{constant}^*\texttt{)} & (\textsc{TupleLiteral}) \\
 & | & \texttt{...} & (\textsc{Ellipsis}) \\
\mathit{type\_ignore} & ::= & i^* & (\textsc{TypeIgnore}) \\
\mathit{id} & \in & \mathit{Id} & \\
s & \in & \mathit{Str} & \\
b & \in & \{\texttt{True}, \texttt{False}\} & \\
i & \in & \mathbb{Z} & \\
f & \in & \mathbb{R} & \\
c & \in & \mathbb{C} &
\end{array}
$$

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

τ	$::=$	$Module$	(Modules)
		$Stmt$	(Statements)
		$Expr$	(Expressions)
		$Comprehension$	(Comprehensions)
		$ExcHandler$	(Exception handlers)
		$Alias$	(Aliases)
		$WithItem$	(Withitems)
		$Pattern$	(Patterns)
		Id	(Identifiers)
		Str	(Strings)
		τ list	(List of elements typed τ)

$\sigma \in \Sigma = Str \xrightarrow{\text{fin}} Id$ Environment storing mappings from strings to identifiers

$.\_1$	:	$\tau \times \tau \to \tau$	Get the first element of the given pair
$@$	:	τ list $\to \tau$ list $\to \tau$ list	Concatenate two lists
$::$	:	$\tau \to \tau$ list $\to \tau$ 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

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

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

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

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

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

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

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

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

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

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

A strict form of assignment statements

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

 # Learning rate scheduler scaling

$\quad$ **IF** $expr_1$ <: `"tensorflow.keras.optimizers.schedules.PiecewiseConstantDecay"` **THEN**
$\quad\quad ([id_r \;\texttt{=}\; expr_1 \;\texttt{(}expr_{11} \;...\; expr_{1n} \;\; (id_1 \;\texttt{=}\;)?\; expr_{21} \;...\; (id_k \;\texttt{=}\;)?\; expr_{2k}\texttt{)} \;(\texttt{\#type:}s)?],$
$\quad\quad\quad \sigma[\textbf{“lr\_scheduler”} \mapsto id_r])$

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

$\quad\quad$ **IF** id_i = `initial_learning_rate` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**
$\quad\quad\quad ([id_r \;\texttt{=}\; expr_1 \;\texttt{(}expr_{11} \;...\; expr_{1n} \;\; (id_1 \;\texttt{=}\;)?\; expr_{21} \;...\; id_i \;\texttt{=}\; expr_{2i} \;\texttt{* hvd.size()}$
$\quad\quad\quad\quad ... \;(id_k \;\texttt{=}\;)?\; expr_{2k}\texttt{)} \;(\texttt{\#type:}s)?], \sigma[\textbf{“lr\_scheduler”} \mapsto id_r])$

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

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

$\quad\quad$ **IF** id_i = `learning_rate` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**
$\quad\quad\quad ([id_r \;\texttt{=}\; expr_1 \;\texttt{(}expr_{11} \;...\; expr_{1n} \;\; (id_1 \;\texttt{=}\;)?\; expr_{21} \;...\; id_i \;\texttt{=}\; expr_{2i} \;\texttt{* hvd.size()}$
$\quad\quad\quad\quad ... \;(id_k \;\texttt{=}\;)?\; expr_{2k}\texttt{)} \;(\texttt{\#type:}s)?], \sigma[\textbf{“lr\_scheduler”} \mapsto id_r])$

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

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

IF id_i = `learning_rate` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **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`)`$?$

id_r `= hvd.DistributedOptimizer(`id_r`)`$]$, $\sigma[$**"optimizer"**$\mapsto id_r])$

ELSE

$([id_r$ `=` $expr_1$ `(`$expr_{11}$ `* hvd.size()` ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` `(#type:`s`)`$?$,
id_r `= hvd.DistributedOptimizer(`id_r`)`$]$, $\sigma[$**"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** $1 \leq \mathrm{i} \leq \mathrm{k}$ **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 $([id_r$ `=` $trans_E[\![$ $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_2$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $]\!](\sigma)$ `(#type:`s`)`$?]$,
$\sigma)$

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

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

THEN $([], \sigma)$

A general form of assignment statements except for the strict form

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

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

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

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

$([$`(#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**$)?], \sigma)$

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

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

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

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

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

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

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

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

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

$$trans_S[\![\ \texttt{import}\ \ alias^*\ ]\!](\sigma) =$$
$$\textbf{LET}\ \ \sigma_1\ = trans_{\overline{A}}[\![\ alias^*\ ]\!](\sigma)\ \textbf{IN}$$
$$\textbf{IF}\ \ \sigma_1\ \setminus\ \sigma\ =\ [\textbf{“tensor\_flow”}\mapsto id]\ \ \textbf{THEN}$$

([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')], σ_1)

ELSE ([import $alias^*$], σ_1)

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

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

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

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** $([], \sigma)$

 # model.compile

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

 # string "adam" case

IF id_i $=$ `optimizer` **WHEN** $2 \leq \mathrm{i} \leq \mathrm{k}$ **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 =)? $expr_{2k}$)], σ)

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 =)? $expr_{2k}$)], σ)

ELSE

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

 # model.fit

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

IF id_i $=$ `callbacks` **WHEN** $2 \leq \mathrm{i} \leq \mathrm{k}$ **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 =)? $expr_{2k}$)], σ)

ELSE

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

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

$expr_1$ $=$ $expr$`.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 =)? $expr_{2k}$)], σ)

ELSE

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$trans_H[\![\texttt{except} \; expr? \; (\texttt{as} \; id)? \; \texttt{:} \; stmt^*]\!](\sigma) = \texttt{except} \; expr? \; (\texttt{as} \; id)? \; \texttt{:} \; trans_{\overline{S}}[\![stmt^*]\!](\sigma)\textbf{.\_1}$

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

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

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

$trans_A[\![\ id\ ]\!](\sigma) =$
  LET id $=$ `tensorflow` **THEN** $\sigma[$**"tensor_flow"**$\mapsto id]$
  LET id $=$ `os` **THEN** $\sigma[$**"os"**$\mapsto id]$
  ELSE σ

$trans_A[\![\ id_1$ `as` $id_2\ ]\!](\sigma) =$
  LET id_1 $=$ `tensorflow` **THEN** $\sigma[$**"tensor_flow"**$\mapsto id_2]$
  LET id_1 $=$ `os` **THEN** $\sigma[$**"os"**$\mapsto id_2]$
  ELSE σ

$trans_A[\![\ id_1\ .id_2\ (.id_3)^*$ (`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**
  ...
  LET $with\_item_n', \sigma_n\ =\ trans_W[\![\ with\_item_n\ ]\!](\sigma_{n-1})$ **IN**
  $(with\_item_1' :: with\_item_2' :: ... :: [with\_item_n'], \sigma_n)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.4 Rule: TensorFlow 2.x GradientTape API Pattern

2.4.1 Transformation Rules

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

$$trans_M[\![\ stmt^*\ \ type\_ignore\ ]\!]\ =\ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma).\_\mathbf{1}\ \ type\_ignore$$

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

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

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

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

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

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

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

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

A strict form of assignment statements

$$\begin{aligned}&trans_S[\![\ id_r\ \texttt{=}\ expr_1\ \texttt{(}expr_{11}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}\ (\texttt{\#type:}s)?\ ]\!](\sigma) =\\ &\quad \textit{\# Dataset}\\ &\quad \textbf{IF}\ \ \sigma(\textbf{``tensor\_flow''})\ =\ id_t\ \ \textbf{AND}\ \ expr_1\ =\ id_t\texttt{.data.Dataset.}expr_3\ \ \textbf{THEN}\\ &\qquad ([id_r\ \texttt{=}\ expr_1\ \texttt{(}expr_{11}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}\ (\texttt{\#type:}s)?],\\ &\qquad\quad \sigma[\textbf{``dataset''} \mapsto id_r])\\ &\quad \textit{\# Variable Broadcasting}\end{aligned}$$

ELIF σ(**"optimizer"**) $=$ id_t **AND** $expr_1$ $=$ id_t`.apply_gradients` **THEN**
  IF id_i $=$ `grads_and_vars` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**
    LET id_z $=$ **NewID()** **IN**
    ([id_z = $expr_{2i}$,
    id_r = $expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... id_i = id_z ... (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`]], σ)
  ELSE
    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`]], σ)

 # 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)?],
    σ[**"lr_scheduler"**$\mapsto$ id_r])
ELIF $expr_1$ <: `tensorflow.keras.optimizers.schedules.LearningRateSchedule` **THEN**
  IF id_i $=$ `initial_learning_rate` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **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)?], σ[**"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)?],
      σ[**"lr_scheduler"**$\mapsto$ id_r])
ELIF $expr_1$ <: `tensorflow.compat.v1.train.exponential_decay` **THEN**
  IF id_i $=$ `learning_rate` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **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)?], σ[**"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)?],
      σ[**"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)?],
    σ[**"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 =)? $expr_{2k}$)], σ[**"model"**$\mapsto$ id_r])
ELIF id_m $=$ **"model"** **AND** $expr_1$ $=$ id_t`.evaluate` **THEN**
  IF id_i $=$ `verbose` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **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 σ(**"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`)?`$],$ σ[**"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 σ(**"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** σ(**"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`)?`$],$ σ[**"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 σ(**"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 σ(**"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 :` $trans_{\overline{S}}⟦\ stmt_2^*\ ⟧(\sigma).\_\mathbf{1}$`)?`], σ)

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

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

$trans_S⟦$ `if (`$expr$`)` `:` $stmt^*$ `(else :` $stmt^*$`)?` $⟧(\sigma) =$
([`if (`$trans_E⟦\ expr\ ⟧(\sigma)$`)` `:` $trans_{\overline{S}}⟦\ stmt^*\ ⟧(\sigma).\_\mathbf{1}$ `(else :` $trans_{\overline{S}}⟦\ stmt^*\ ⟧(\sigma).\_\mathbf{1}$`)?`], σ)

$trans_S⟦$ `(#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`)`], σ_2)

ELSE ([`(#type:`s`)?` `with` $with\_item^{*\prime}$ `:` $stmt^{*\prime}$], σ_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`)`], σ_2)

ELSE ([`(#type:`s`)?` `async with` $with\_item^{*\prime}$ `:` $stmt^{*\prime}$], σ_2)

$trans_S⟦$ `match` $expr$ `:` $match\_case^*$ $⟧(\sigma) =$
([`match` $trans_E⟦\ expr\ ⟧(\sigma)$ `:` $(trans_C⟦match\_case⟧(\sigma))^*$], σ)

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

$trans_S⟦$ `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 :` $trans_{\overline{S}}⟦\ stmt_3^*\ ⟧(\sigma).\_\mathbf{1}$`)?`], σ)

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

$trans_S[\![$ `import` $alias^*]\!](\sigma) =$

LET $\sigma_1 = trans_{\overline{A}}[\![alias^*]\!](\sigma)$ **IN**

 # import

IF $\sigma_1 \setminus \sigma$ = [**"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^*], σ)

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

A strict form of expr statements

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

 # Variable Broadcasting

IF σ(**"optimizer"**) $= id_t$ **AND** $expr_1 = id_t$`.apply_gradients` **THEN**

IF id_i = `grads_and_vars` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**

LET id_z = **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] 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 `=` $expr_{11}$,

$expr_1$ `(`$id_z\ expr_{12} \dots expr_{1n}$ $(id_1$ `=` $)?\ expr_{21} \dots (id_k$ `=` $)?\ expr_{2k}$`)`,

`global hvd_broadcast_done`,

`if not hvd_broadcast_done:` [`hvd.broadcast_variables([x[1] 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**

$expr_1 \;=\; expr$.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 =)? $expr_{2k}$)], σ)
ELSE
[$trans_E\llbracket$ $expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$) $\rrbracket(\sigma)$], σ)

A general form of expr statements except for the strict form

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

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

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

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

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

$trans_E\llbracket\; expr_1 \;\; boolop \;\; expr_2 \;\rrbracket(\sigma) = trans_E\llbracket\; expr_1 \;\rrbracket(\sigma) \;\; boolop \;\; trans_E\llbracket\; expr_2 \;\rrbracket(\sigma)$

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

$trans_E\llbracket\; expr_1 \;\; binop \;\; expr_2 \;\rrbracket(\sigma) = trans_E\llbracket\; expr_1 \;\rrbracket(\sigma) \;\; binop \;\; trans_E\llbracket\; expr_2 \;\rrbracket(\sigma)$

$trans_E\llbracket\; unop \;\; expr \;\rrbracket(\sigma) = unop \;\; trans_E\llbracket\; expr \;\rrbracket(\sigma)$

$trans_E\llbracket$ lambda $args$: $expr \;\rrbracket(\sigma) =$ lambda $args$: $trans_E\llbracket\; expr \;\rrbracket(\sigma)$

$trans_E\llbracket\; expr_1$ if $expr_2$ else $expr_3 \;\rrbracket(\sigma) =$
$trans_E\llbracket\; expr_1 \;\rrbracket(\sigma)$ if $trans_E\llbracket\; expr_2 \;\rrbracket(\sigma)$ else $trans_E\llbracket\; expr_3 \;\rrbracket(\sigma)$

$trans_E\llbracket\; expr_1$ if $expr_2$ else $expr_3 \;\rrbracket(\sigma) =$
$trans_E\llbracket\; expr_1 \;\rrbracket(\sigma)$ if $trans_E\llbracket\; expr_2 \;\rrbracket(\sigma)$ else $trans_E\llbracket\; expr_3 \;\rrbracket(\sigma)$

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

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

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

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

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

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

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

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

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

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

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

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

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

 # Adjust Steps

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

IF $id_i \ =$ `count` **WHEN** $1 \le \mathrm{i} \le \mathrm{k}$ **THEN**

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

ELSE

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

ELSE

$trans_E[\![\ expr_1\ ]\!](\sigma)$ `(`$trans_E[\![\ expr_{11}\ ]\!](\sigma)$ ... $trans_E[\![\ expr_{1n}\ ]\!](\sigma)$

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

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

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

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

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

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

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

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

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

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

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

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

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

$trans_H[\![\ \texttt{except}\ \ expr?\ (\texttt{as}\ \ id)?\ \texttt{:}\ \ stmt^*\ ]\!](\sigma) = \texttt{except}\ \ expr?\ (\texttt{as}\ \ id)?\ \texttt{:}\ \ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)\mathbf{.\_1}$

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

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

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

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

$\quad \mathbf{LET}\ \ id\ = \texttt{tensorflow}\ \ \mathbf{THEN}\ \ \sigma[\text{\textbf{``tensor\_flow''}}\mapsto id]$

$\quad \mathbf{LET}\ \ id\ = \texttt{os}\ \ \mathbf{THEN}\ \ \sigma[\text{\textbf{``os''}}\mapsto id]$

$\quad \mathbf{LET}\ \ id\ = \texttt{config}\ \ \mathbf{THEN}\ \ \sigma[\text{\textbf{``config''}}\mapsto id]$

$\quad \mathbf{ELSE}\ \ \sigma$

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

$\quad \mathbf{LET}\ \ id_1\ = \texttt{tensorflow}\ \ \mathbf{THEN}\ \ \sigma[\text{\textbf{``tensor\_flow''}}\mapsto id_2]$

$\quad \mathbf{LET}\ \ id_1\ = \texttt{os}\ \ \mathbf{THEN}\ \ \sigma[\text{\textbf{``os''}}\mapsto id_2]$

$\quad \mathbf{LET}\ \ id\ = \texttt{config}\ \ \mathbf{THEN}\ \ \sigma[\text{\textbf{``config''}}\mapsto id_2]$

$\quad \mathbf{ELSE}\ \ \sigma$

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

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

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

$\quad \mathbf{LET}\ \ with\_item_1', \sigma_1\ = trans_W[\![\ with\_item_1\ ]\!](\sigma)\ \ \mathbf{IN}$

$\quad \mathbf{LET}\ \ with\_item_2', \sigma_2\ = trans_W[\![\ with\_item_2\ ]\!](\sigma_1)\ \ \mathbf{IN}$

$\quad \ldots$

$\quad \mathbf{LET}\ \ with\_item_n', \sigma_n\ = trans_W[\![\ with\_item_n\ ]\!](\sigma_{n-1})\ \ \mathbf{IN}$

$\quad (with\_item_1'\ \mathbf{::}\ with\_item_2'\ \mathbf{::}\ ...\ \mathbf{::}\ [with\_item_n'], \sigma_n)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.5 Rule: TensorFlow 1.x Mainscript API Pattern

2.5.1 Transformation Rules

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

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

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

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

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

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

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

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

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

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

A strict form of assignment statements

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

 # Config Proto

$\textbf{IF}\ \ id_t \ = \ \textbf{“tensor\_flow”}\ \ \textbf{AND}\ \ expr_1 \ = \ id_t\texttt{.ConfigProto} \ \ \textbf{THEN}$
$\quad (id_r\ \texttt{=}\ expr_1\ \texttt{(}expr_{11}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}\ (\texttt{\#type:}s)?,$
$\qquad \sigma[id_r \mapsto \textbf{“config\_proto”}])$

 # Minimize

 # Learning rate scheduler scaling

$\textbf{IF}\ \ expr_1\ \texttt{<:}\ \ \texttt{"tensorflow.keras.optimizers.schedules.PiecewiseConstantDecay"} \ \ \textbf{THEN}$
$\quad ([id_r\ \texttt{=}\ expr_1\ \texttt{(}expr_{11}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}\ (\texttt{\#type:}s)?],$
$\qquad \sigma[\textbf{“lr\_scheduler”} \mapsto id_r])$
$\textbf{ELIF}\ \ expr_1\ \texttt{<:}\ \ \texttt{tensorflow.keras.optimizers.schedules.LearningRateSchedule} \ \ \textbf{THEN}$
$\quad \textbf{IF}\ \ id_i \ = \ \texttt{initial\_learning\_rate} \ \ \textbf{WHEN}\ \ 1 \le \mathrm{i} \le \mathrm{k} \ \ \textbf{THEN}$
$\qquad ([id_r\ \texttt{=}\ expr_1\ \texttt{(}expr_{11}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ id_i\ \texttt{=}\ expr_{2i}\ \texttt{* hvd.size()}$
$\qquad\quad ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}\ (\texttt{\#type:}s)?], \sigma[\textbf{“lr\_scheduler”} \mapsto id_r])$
$\quad \textbf{ELSE}$
$\qquad ([id_r\ \texttt{=}\ expr_1\ \texttt{(}expr_{11}\ \texttt{* hvd.size()}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}\ (\texttt{\#type:}s)?],$
$\qquad\quad \sigma[\textbf{“lr\_scheduler”} \mapsto id_r])$
$\textbf{ELIF}\ \ expr_1\ \texttt{<:}\ \ \texttt{tensorflow.compat.v1.train.exponential\_decay} \ \ \textbf{THEN}$
$\quad \textbf{IF}\ \ id_i \ = \ \texttt{learning\_rate} \ \ \textbf{WHEN}\ \ 1 \le \mathrm{i} \le \mathrm{k} \ \ \textbf{THEN}$
$\qquad ([id_r\ \texttt{=}\ expr_1\ \texttt{(}expr_{11}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ id_i\ \texttt{=}\ expr_{2i}\ \texttt{* hvd.size()}$
$\qquad\quad ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}\ (\texttt{\#type:}s)?], \sigma[\textbf{“lr\_scheduler”} \mapsto id_r])$
$\quad \textbf{ELSE}$
$\qquad ([id_r\ \texttt{=}\ expr_1\ \texttt{(}expr_{11}\ \texttt{* hvd.size()}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}\ (\texttt{\#type:}s)?],$

σ["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** $1 \le i \le k$ **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)?

id_r = `hvd.DistributedOptimizer(`id_r`)`], σ["optimizer"$\mapsto id_r$])

ELSE

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

id_r = `hvd.DistributedOptimizer(`id_r`)`], σ["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 =)? $expr_{2k}$)], σ["model"$\mapsto id_r$])

ELIF id_m = "model" **AND** $expr_1$ = id_t`.evaluate` **THEN**

IF id_i = `verbose` **WHEN** $1 \le i \le k$ **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 σ("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)?],

σ["checkpoint"$\mapsto id_r$])

 # Default

ELSE ([id_r = $trans_E$⟦ $expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_2 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$) ⟧(σ) (`#type:`s)?],

σ)

$trans_S$⟦ $expr_0$ = $expr_1$ (`#type:`s)? ⟧(σ) =

IF σ("os") = id_{os} **AND** $expr_0$ = id_{os}`.environ ['CUDA_VISIBLE_DEVICES']`

THEN ([], σ)

A general form of assignment statements except for the strict form

$trans_S$⟦ $expr_1^*$ = $expr_2$ (`#type:`s)? ⟧(σ) = ([$expr_1^*$ = $trans_E$⟦ $expr_2$ ⟧(σ) (`#type:`s)?], σ)

$trans_S$⟦ $expr_1$ $binop$ = $expr_2$ ⟧(σ) = ([$expr_1$ $binop$ = $trans_E$⟦ $expr_2$ ⟧(σ)], σ)

$trans_S$⟦ $expr_1$: $expr_2$ (= $expr_3$)? ⟧(σ) = ([$expr_1$: $expr_2$ (= $trans_E$⟦ $expr_3$ ⟧(σ))?], σ)

$trans_S$⟦ (`#type:`s)? `for` $expr_1$ `in` $expr_2$: $stmt_1^*$ (`else` : $stmt_2^*$)? ⟧(σ) =

([(`#type:`s)? `for` $expr_1$ `in` $trans_E$⟦ $expr_2$ ⟧(σ) :

$trans_{\overline{S}}$⟦ $stmt_1^*$ ⟧(σ)._**1** (`else` : $trans_{\overline{S}}$⟦ $stmt_2^*$ ⟧(σ)._**1**)?], σ)

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

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

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

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

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

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

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

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

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

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

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

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

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

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`$], \sigma)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$trans_H[\![\ \texttt{except}\ \ expr?\ (\texttt{as}\ \ id)?\ :\ \ stmt^*\ ]\!](\sigma) = \texttt{except}\ \ expr?\ (\texttt{as}\ \ id)?\ :\ \ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)\textbf{.\_1}$$

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

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

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

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

LET id = `tensorflow.compat.v1` **THEN** $\sigma[$**"tensor_flow_compat"**$\mapsto id]$
LET id = `tensorflow` **THEN** $\sigma[$**"tensor_flow"**$\mapsto id]$
LET id = `os` **THEN** $\sigma[$**"os"**$\mapsto id]$
ELSE σ

$trans_A[\![\ id_1$ `as` $id_2\ ]\!](\sigma) =$

LET id_1 = `tensorflow.compat.v1` **THEN** $\sigma[$**"tensor_flow_compat"**$\mapsto id_2]$
LET id_1 = `tensorflow` **THEN** $\sigma[$**"tensor_flow"**$\mapsto id_2]$
LET id_1 = `os` **THEN** $\sigma[$**"os"**$\mapsto id_2]$
ELSE σ

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

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

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

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**
...
LET $with\_item_n', \sigma_n = trans_W[\![\ with\_item_n\ ]\!](\sigma_{n-1})$ **IN**
$(with\_item_1'\ \textbf{::}\ with\_item_2'\ \textbf{::}\ ...\ \textbf{::}\ [with\_item_n'], \sigma_n)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\sigma\ \in \Sigma\ = Id \xrightarrow{\text{fin}} \textit{Summary}$ Environment storing mappings from ids to summaries

3.2.2 Training Loop

$$\boxed{\textit{summary}_M\ :\ \textit{Module}\ \rightarrow\ \Sigma\ \rightarrow\ \textit{ModuleSummary}}$$

$$
\begin{array}{l}
\textit{summary}_M[\![\ id\ \ stmt^*\ \ type\_ignore\ ]\!] = \\
\quad \textbf{LET}\ \ \sigma, tl\ = \ \textit{summary}_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)\ \ \textbf{IN} \\
\quad \texttt{ModuleSummary}\ \ id\ \ \sigma\ \ tl
\end{array}
$$

$$\boxed{\textit{summary}_{\overline{S}}\ :\ \textit{Stmt}\ \textbf{list}\ \rightarrow\ \Sigma\ \rightarrow\ (\Sigma\ \times\ tl)}$$

$$
\begin{array}{lll}
\textit{summary}_{\overline{S}}[\![\ stmt_1\ \ stmt_2\ ...\ stmt_n\ ]\!](\sigma) & = & \textbf{LET}\ \ \sigma_1, tl_1\ = \textit{summary}_S[\![\ stmt_1\ ]\!](\sigma)\ \ \textbf{IN} \\
 & & \textbf{LET}\ \ \sigma_2, tl_2\ = \textit{summary}_{\overline{S}}[\![\ stmt_2\ ]\!](\sigma_1)\ \ \textbf{IN} \\
 & & \ldots \\
 & & \textbf{LET}\ \ \sigma_n, tl_n\ = \textit{summary}_{\overline{S}}[\![\ stmt_n\ ]\!](\sigma_{n-1})\ \ \textbf{IN} \\
 & & (\sigma_n,\ tl_1 \sqcup tl_2 \sqcup\ \ \ldots\ \ tl_n)
\end{array}
$$

$$\boxed{\textit{summary}_S\ :\ \textit{Stmt}\ \rightarrow\ \Sigma\ \rightarrow\ (\Sigma\ \times\ tl)}$$

$$
\begin{array}{l}
\textit{summary}_S[\![\ (@expr_1)^*\ \ \texttt{def}\ \ id\ \ (args)\ \ (\texttt{->}\ \ expr_2)?\ \ :\ \ (\texttt{\#type:}s)?\ \ stmt^*\ ]\!](\sigma) = \\
\quad \textbf{LET}\ \ \sigma', tl\ = \textit{summary}_{\overline{S}}[\![\ stmt^*\ ]\!]\ \ \textbf{IN} \\
\quad (\sigma[id \mapsto \texttt{FuncSummary}\ \ tl], \bot)
\end{array}
$$

$$
\begin{array}{l}
\textit{summary}_S[\![\ (@expr_1)^*\ \ \texttt{async def}\ \ id\ \ (args)\ \ (\texttt{->}\ \ expr_2)?\ \ :\ \ (\texttt{\#type:}s)?\ \ stmt^*\ ]\!](\sigma) = \\
\quad \textbf{LET}\ \ \sigma', tl\ = \textit{summary}_{\overline{S}}[\![\ stmt^*\ ]\!]\ \ \textbf{IN} \\
\quad (\sigma[id \mapsto \texttt{FuncSummary}\ \ tl], \bot)
\end{array}
$$

$$
\begin{array}{l}
\textit{summary}_S[\![\ (@expr)^*\ \ \texttt{class}\ \ id\ \ (expr_{11}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ \ :\ \ stmt^*\ ]\!](\sigma) = \\
\quad \textbf{IF}\ \ expr_{1i}\ = id_c\texttt{.keras.Model}\ \ \textbf{WHEN}\ \ 1 \le \text{i} \le \text{n}\ \ \textbf{AND} \\
\qquad \sigma(id_c)\ = \texttt{ModuleSummary}\ \text{Id("tensorflow")}\ \ []\ \ \bot\ \textbf{THEN}\ \ (\sigma[id \mapsto \texttt{ClassSummary}\ \ Model], \bot) \\
\quad \textbf{ELIF}\ \ expr_{2i}\ = id_c\texttt{.keras.Model}\ \ \textbf{WHEN}\ \ 1 \le \text{i} \le \text{k}\ \ \textbf{AND} \\
\qquad \sigma(id_c)\ = \texttt{ModuleSummary}\ \text{Id("tensorflow")}\ \ []\ \ \bot\ \textbf{THEN}\ \ (\sigma[id \mapsto \texttt{ClassSummary}\ \ Model], \bot) \\
\quad \textbf{ELSE}\ \ (\sigma[id \mapsto \texttt{ClassSummary}\ \ \bot], \bot)
\end{array}
$$

A strict form of assignment statements

$summary_S[\![$ id_r `=` id_c `()` (`#type:`s)? $]\!](\sigma) =$
  IF $\sigma(id_c)$ $=$ `ClassSummary` $Model$ **THEN** ($\sigma[id_r \mapsto$ `ValueSummary` **"model_instance"**$]$, $\perp$)
  ELSE $(\sigma, \perp)$

$summary_S[\![$ (`#type:`s)? `for` $expr_1$ `in` $expr_2$ `:` $stmt_1^*$ (`else` `:` $stmt_2^*$)? $]\!](\sigma) =$
  LET $\_, tl$ $=$ $summary_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma)$ **IN**
  (σ, tl)

$summary_S[\![$ (`#type:`s)? `async` `for` $expr_1$ `in` $expr_2$ `:` $stmt_1^*$ (`else` `:` $stmt_2^*$)? $]\!](\sigma) =$
  LET $\_, tl$ $=$ $summary_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma)$ **IN**
  (σ, tl)

$summary_S[\![$ `while` `(`$expr$`)` `:` $stmt_1^*$ (`else` `:` $stmt_2^*$)? $]\!](\sigma) =$
  LET $\_, tl$ $=$ $summary_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma)$ **IN**
  (σ, tl)

$summary_S[\![$ (`#type:`s)? `with` $with\_item^*$ `:` $stmt^*$ $]\!](\sigma) =$
  LET $\_, tl$ $=$ $summary_{\overline{S}}[\![$ $stmt^*$ $]\!](\sigma)$ **IN**
  $(\sigma, summary_{\overline{W}}[\![$ $with\_item^*$ $]\!](\sigma) \sqcup tl)$

$summary_S[\![$ (`#type:`s)? `async` `with` $with\_item^*$ `:` $stmt^*$ $]\!](\sigma) =$
  LET $\_, tl$ $=$ $summary_{\overline{S}}[\![$ $stmt^*$ $]\!](\sigma)$ **IN**
  $(\sigma, summary_{\overline{W}}[\![$ $with\_item^*$ $]\!](\sigma) \sqcup tl)$

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

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

$summary_S[\![$ `from` 0 id^* `import` $alias^*$ $]\!](\sigma) =$
  LET σ' $=$ $summary_{\overline{A}}[\![$ $alias^*$ $]\!](\sigma)$ **IN**
  LET $[id_2 \mapsto$ `ModuleSummary` id_2 σ_2 $\perp$, $id_2 \mapsto$ `ModuleSummary` id_2 σ_2 $\perp$,
    ... $id_n \mapsto$ `ModuleSummary` id_n σ_n $\perp]$ $=$ σ' $\setminus$ σ **IN**
  σ $++$ $[id_2 \mapsto$ `ModuleSummary` $(id^*{+}id_2)$ σ_2 $\perp$, $id_2 \mapsto$ `ModuleSummary` $(id^*{+}id_2)$ σ_2 $\perp$,
    ... $id_n \mapsto$ `ModuleSummary` $(id^*{+}id_n)$ σ_n $\perp]$

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

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

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

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

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

$trans_A[\![\ id_1\ id_2\ ...\ id_n\ ]\!](\sigma) =$

$\quad$ **LET** $\sigma_n = [id_n \mapsto \texttt{ModuleSummary}\ \ id_n \quad \top \ \ \bot]$

$\quad$ **LET** $\sigma_{n-1} = [id_{n-1} \mapsto \texttt{ModuleSummary}\ \ id_{n-1} \quad \sigma_n \quad \bot]$

$\quad$ **...**

$\quad$ **LET** $\sigma_2 = [id_2 \mapsto \texttt{ModuleSummary}\ \ id_2 \quad \sigma_3 \quad \bot]$

$\quad$ **LET** $\sigma_1 = [id_1 \mapsto \texttt{ModuleSummary}\ \ id_1 \quad \sigma_2 \quad \bot]$

$\quad \sigma \ \ ++ \ \ \sigma_1$

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

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

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

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

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

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

A strict form of call expression

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

$\quad$ **IF** $\sigma(id_r) \ \ = \ \texttt{ValueSummary}$ **"model_instance"** **AND**

$\qquad expr_1 \ \ = \ id_r\texttt{.fit}$ **THEN** $Optimizer$

$\quad$ **ELIF** $\sigma(id_r) \ \ = \ \texttt{ModuleSummary}\ \mathrm{Id}(\text{"tensorflow"}) \ \ \sigma' \ \ tl$ **AND**

$\qquad expr_1 \ \ = \ id_r\texttt{.GradientTape}$ **THEN** $GradTape$

$\quad$ **ELIF** $\sigma(id_r) \ \ = \ \texttt{FuncSummary}\ \ tl$ **AND**

$\qquad expr_1 \ \ = \ \texttt{idr}$ **THEN** tl

$\quad$ **ELSE** $\bot$

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