Code Transformation for Distributed Python ML Code

1 Python Abstract Syntax

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

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

match_case	$::=$	`case` *pattern* (`if` *expr*)? `:` *stmt**	(MatchCase)
pattern	$::=$	*expr*	(MatchValue)
	$\mid$	*constant*	(MatchSingleton)
	$\mid$	`[`*pattern**`]`	(MatchSequence)
	$\mid$	`*`(*id*)?	(MatchStar)
	$\mid$	$\{$(*expr* `:` *pattern*)* *id*?$\}$	(MatchMapping)
	$\mid$	*expr* `(`*pattern** (*id* = *pattern*)*`)`	(MatchClass)
	$\mid$	(*pattern* `as`)? *id*	(MatchAs)
	$\mid$	*pattern* `|` *pattern*	(MatchOr)
	$\mid$	_	(MatchWildcard)
constant	$::=$	`None`	(NoneLiteral)
	$\mid$	i	(IntLiteral)
	$\mid$	f	(FloatLiteral)
	$\mid$	c	(ComplexLiteral)
	$\mid$	s	(StringLiteral)
	$\mid$	b	(BooleanLiteral)
	$\mid$	`(`*constant**`)`	(TupleLiteral)
	$\mid$	...	(Ellipsis)
type_ignore	$::=$	i^*	(TypeIgnore)
id	$\in$	Id	
s	$\in$	Str	
b	$\in$	$\{$`True`, `False`$\}$	
i	$\in$	$\mathbb{Z}$	
f	$\in$	$\mathbb{R}$	
c	$\in$	$\mathbb{C}$	

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

2 Transformation Rule for TensorFlow ML Training Code

2.1 Restrictions

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

2.2 Rules

2.2.1 Types and Auxiliary Functions

$$
\begin{array}{lll}
\tau & ::= & \mathit{Module} \quad & (\textsc{Modules}) \\
& & \mathit{Stmt} & (\textsc{Statements}) \\
& & \mathit{Expr} & (\textsc{Expressions}) \\
& & \mathit{Comprehension} & (\textsc{Comprehensions}) \\
& & \mathit{ExcHandler} & (\textsc{Exception handlers}) \\
& & \mathit{Alias} & (\textsc{Aliases}) \\
& & \mathit{WithItem} & (\textsc{WithItems}) \\
& & \mathit{Pattern} & (\textsc{Patterns}) \\
& & \mathit{Id} & (\textsc{Identifiers}) \\
& & \mathit{Str} & (\textsc{Strings}) \\
& & \tau \text{ list} & (\textsc{List of elements typed } \tau)
\end{array}
$$

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

._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)
<:	:	$\mathit{Expr} \times \mathit{Expr} \to \mathbb{B}$	Whether classes specified by the expressions are subclasses

2.3 Rule: TensorFlow 2.x Keras API Pattern

2.3.1 Transformation Rules

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

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

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

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

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

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

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

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

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

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

A strict form of assignment statements

$$\begin{aligned} & trans_S[\![\; id_r \texttt{ = } expr_1 \texttt{ (}expr_{11} \ldots expr_{1n} \quad (id_1 \texttt{ = })?\ expr_{21} \ldots (id_k \texttt{ = })?\ expr_{2k}\texttt{)} \;(\texttt{\#type:}s)? \;]\!](\sigma) = \\ & \quad \textit{\# Learning rate scheduler scaling} \\ & \quad \mathbf{IF} \;\; expr_1 <: \texttt{tensorflow.keras.optimizers.schedules.PiecewiseConstantDecay} \;\; \mathbf{THEN} \\ & \qquad ([id_r \texttt{ = } expr_1 \texttt{ (}expr_{11} \ldots expr_{1n} \quad (id_1 \texttt{ = })?\ expr_{21} \ldots (id_k \texttt{ = })?\ expr_{2k}\texttt{)} \;(\texttt{\#type:}s)?], \\ & \qquad\quad \sigma[\text{“}\mathbf{lr\_scheduler}\text{”} \mapsto id_r]) \\ & \quad \mathbf{ELIF} \;\; expr_1 <: \texttt{tensorflow.keras.experimental.PiecewiseConstantDecay} \;\; \mathbf{THEN} \\ & \qquad ([id_r \texttt{ = } expr_1 \texttt{ (}expr_{11} \ldots expr_{1n} \quad (id_1 \texttt{ = })?\ expr_{21} \ldots (id_k \texttt{ = })?\ expr_{2k}\texttt{)} \;(\texttt{\#type:}s)?], \\ & \qquad\quad \sigma[\text{“}\mathbf{lr\_scheduler}\text{”} \mapsto id_r]) \\ & \quad \mathbf{ELIF} \;\; expr_1 <: \texttt{tensorflow.keras.optimizers.schedules.LearningRateSchedule} \;\; \mathbf{THEN} \\ & \qquad \mathbf{IF} \;\; id_i \;=\; \texttt{initial\_learning\_rate} \;\; \mathbf{WHEN} \;\; 1 \leq \mathrm{i} \leq \mathrm{k} \;\; \mathbf{THEN} \\ & \qquad\quad ([id_r \texttt{ = } expr_1 \texttt{ (}expr_{11} \ldots expr_{1n} \quad (id_1 \texttt{ = })?\ expr_{21} \ldots id_i \texttt{ = } expr_{2i} \texttt{ * hvd.size()} \\ & \qquad\qquad \ldots (id_k \texttt{ = })?\ expr_{2k}\texttt{)} \;(\texttt{\#type:}s)?], \sigma[\text{“}\mathbf{lr\_scheduler}\text{”} \mapsto id_r]) \\ & \qquad \mathbf{ELSE} \\ & \qquad\quad ([id_r \texttt{ = } expr_1 \texttt{ (}expr_{11} \texttt{ * hvd.size()} \ldots expr_{1n} \quad (id_1 \texttt{ = })?\ expr_{21} \ldots (id_k \texttt{ = })?\ expr_{2k}\texttt{)} \;(\texttt{\#type:}s)?], \\ & \qquad\qquad \sigma[\text{“}\mathbf{lr\_scheduler}\text{”} \mapsto id_r]) \\ & \quad \mathbf{ELIF} \;\; expr_1 <: \texttt{tensorflow.compat.v1.train.exponential\_decay} \;\; \mathbf{THEN} \end{aligned}$$

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

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

IF id_i $=$ `learning_rate` **WHEN** $1 \le \mathrm{i} \le \mathrm{k}$ **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`)`] 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.models.Model` **THEN**

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

σ[**"model"**$\mapsto$ id_r])

ELIF σ(**"model"**) $=$ id_t **AND** $expr_1$ $=$ id_t`.evaluate` **THEN**

IF id_i $=$ `verbose` **WHEN** $1 \le \mathrm{i} \le \mathrm{k}$ **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)?], σ)

ELSE

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

`verbose = 1 if hvd.rank() == 0 else 0`) (`#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)

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

$\quad$ *# Dataset in Assign*

$\quad$ **IF** $expr_1\ =\ id_1$ **AND** $\sigma(\textbf{“tensor\_flow”})\ =\ id_2$

$\quad$ **AND** $expr_3\ =\ id_2\texttt{.data.Dataset.}expr_4\ (expr_5^*\ keyword^*)$ **THEN**

$\qquad ([expr_1\ \texttt{:}\ expr_2\ \texttt{=}\ expr_3\ (\texttt{\#type:}s)?],\ \sigma[\textbf{“dataset”}\mapsto id_1])$

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

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

$\quad ([(\texttt{\#type:}s)?\ \texttt{for}\ expr_1\ \texttt{in}\ trans_E[\![\ expr_2\ ]\!](\sigma)\ \texttt{:}$

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

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

$\quad ([(\texttt{\#type:}s)?\ \texttt{async for}\ expr_1\ \texttt{in}\ trans_E[\![\ expr_2\ ]\!](\sigma)\ \texttt{:}$

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

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

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

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

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

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

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

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

$\quad$ **IF** $\sigma_1 \setminus \sigma\ =\ [\textbf{“gradient\_tape”}\mapsto id]$ **THEN**

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

$\qquad id\ \texttt{= hvd.DistributedGradientTape(}id\texttt{)}],\ \sigma_2)$

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

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

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

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

$\quad$ **IF** $\sigma_1 \setminus \sigma\ =\ [\textbf{“gradient\_tape”}\mapsto id]$ **THEN**

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

$\qquad id\ \texttt{= hvd.DistributedGradientTape(}id\texttt{)}],\ \sigma_2)$

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

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

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

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

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

$\quad ([\texttt{try}\ \texttt{:}\ trans_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma)\textbf{.\_1}\ (trans_H[\![exc\_handler]\!](\sigma))^*$

$\quad (\texttt{else}\ \texttt{:}\ trans_{\overline{S}}[\![\ stmt_2^*\ ]\!](\sigma)\textbf{.\_1})?\ (\texttt{finally}\ \texttt{:}\ trans_{\overline{S}}[\![\ stmt_3^*\ ]\!](\sigma)\textbf{.\_1})?],\ \sigma)$

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

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

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

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)

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

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

IF $\sigma_1 \ \setminus \ \sigma \ = \ [$**"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')`], σ_1)

ELSE ([`import` $alias^*$], σ_1)

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

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

IF id = `tensorflow` **AND** $\sigma_1 \ \setminus \ \sigma \ = \ [$**"keras"**$\mapsto id]$ **THEN**

([`from` $i \quad id?$ `import` $alias^*$,

`import horovod.tensorflow.keras as hvd`,

`hvd.init()`,

`gpus =` id`.config.experimental.list_physical_devices('GPU')`,

`for gpu in gpus:` id`.config.experimental.set_memory_growth(gpu, True)`,

`if gpus:` id`.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')`], σ_1)

ELSE ([`from` $i \quad id?$ `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

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

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

```
  IF id_i = grads_and_vars WHEN 1 ≤ i ≤ 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)
# MainScript rule
ELIF expr_1 = print OR expr_1 = expr.write OR expr_1 = expr.summary OR
  expr_1 = expr.save_weights OR expr_1 = expr.load_weights THEN
     ([if hvd.rank() == 0: expr_1 (expr_11 ... expr_1n (id_1 = )? expr_21 ... (id_k = )? expr_2k)], σ)
ELIF σ("checkpoint") = id_t AND expr_1 = id_t.save THEN
  ([if hvd.rank() == 0: [expr_1 (expr_11 ... expr_1n (id_1 = )? expr_21 ... (id_k = )? expr_2k)]], σ)
ELIF σ("model") = id_t AND expr_1 = id_t.fit THEN
  IF id_i = verbose WHEN 1 ≤ i ≤ k THEN
     IF id_j = callbacks WHEN 1 ≤ j ≤ k THEN
        ([callback = [hvd.callbacks.BroadcastGlobalVariablesCallback(root_rank=0)]
        if hvd.rank() == 0:  callbacks.append(expr_2j)
        expr_1 (expr_11 ... expr_1n (id_1 = )? expr_21 ... id_i = 1 if hvd.rank() == 0 else 0
        ... id_j = callbacks ... (id_k = )? expr_2k)], σ)
     ELSE
        ([expr_1 (expr_11 ... expr_1n (id_1 = )? expr_21 ...
           id_i = 1 if hvd.rank() == 0 else 0 ... (id_k = )? expr_2k
           callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(root_rank=0)])], σ)
  ELSE
     IF id_j = callbacks WHEN 1 ≤ j ≤ k THEN
        ([callback = [hvd.callbacks.BroadcastGlobalVariablesCallback(root_rank=0)]
        if hvd.rank() == 0:  callbacks.append(expr_2j)
        expr_1 (expr_11 ... expr_1n (id_1 = )? expr_21 ... id_j = callbacks ... (id_k = )? expr_2k
           verbose = 1 if hvd.rank() == 0 else 0)], σ)
     ELSE
        ([expr_1 (expr_11 ... expr_1n (id_1 = )? expr_21 ... (id_k = )? expr_2k
           verbose = 1 if hvd.rank() == 0 else 0
           callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(0)])], σ)
ELIF σ("model") = id_t AND expr_1 = id_t.compile THEN
```

IF id_i = `optimizer` **AND** $expr_{2i}$ = `"adam"` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**

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

id_z `= hvd.DistributedOptimizer(`id_z`)`

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

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

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

id_z `= hvd.DistributedOptimizer(`id_z`)`

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

ELSE

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

ELIF $expr_1$ = `model.summary` **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 `=`)? $expr_{2k}$`)` ⟧(σ)], 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` $⟧(\sigma) = ([$`continue`$],\ \sigma)$

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

$trans_E⟦\ expr_1 \ \ boolop \ \ expr_2\ ⟧(\sigma) = trans_E⟦\ expr_1\ ⟧(\sigma) \ \ boolop \ \ trans_E⟦\ expr_2\ ⟧(\sigma)$

$trans_E⟦\ expr_1 \ \ \mathbf{:=} \ \ expr_2\ ⟧(\sigma) = expr_1 \ \ \mathbf{:=} \ \ trans_E⟦\ expr_2\ ⟧(\sigma)$

$trans_E⟦\ expr_1 \ \ binop \ \ expr_2\ ⟧(\sigma) = trans_E⟦\ expr_1\ ⟧(\sigma) \ \ binop \ \ trans_E⟦\ expr_2\ ⟧(\sigma)$

$trans_E⟦\ unop \ \ expr\ ⟧(\sigma) = unop \ \ trans_E⟦\ expr\ ⟧(\sigma)$

$trans_E⟦$ `lambda` $args$ `:` $expr\ ⟧(\sigma) =$ `lambda` $args$ `:` $trans_E⟦\ expr\ ⟧(\sigma)$

$trans_E⟦\ expr_1$ `if` $expr_2$ `else` $expr_3\ ⟧(\sigma) =$
$trans_E⟦\ expr_1\ ⟧(\sigma)$ `if` $trans_E⟦\ expr_2\ ⟧(\sigma)$ `else` $trans_E⟦\ expr_3\ ⟧(\sigma)$

$trans_E⟦\ expr_1$ `if` $expr_2$ `else` $expr_3\ ⟧(\sigma) =$
$trans_E⟦\ expr_1\ ⟧(\sigma)$ `if` $trans_E⟦\ expr_2\ ⟧(\sigma)$ `else` $trans_E⟦\ expr_3\ ⟧(\sigma)$

$trans_E⟦\ \{(expr_1 \ : \ expr_2)^* \ \ (**expr_3)^*\}\ ⟧(\sigma) = \{(expr_1 \ : \ trans_E⟦expr_2⟧(\sigma))^* \ \ (**expr_3)^*\}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$trans_A[\![\ id_1\ \texttt{as}\ \ id_2\ ]\!](\sigma) =$
LET id_1 = `tensorflow` **THEN** $\sigma[$**"tensor_flow"**$\mapsto id_2]$
LET id_1 = `tensorflow.keras` **THEN** $\sigma[$**"keras"**$\mapsto id_2]$
LET id_1 = `os` **THEN** $\sigma[$**"os"**$\mapsto id_2]$
LET id = `optimizers` **THEN** $\sigma[$**"optimizers"**$\mapsto id]$
ELSE σ

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

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

$$
\begin{array}{l}
trans_{\overline{W}}[\![\ with\_item_1\ with\_item_2\ ...\ with\_item_n\ ]\!](\sigma) = \\
\quad \textbf{LET}\ \ with\_item_1', \sigma_1 \ \ = \ trans_W[\![\ with\_item_1\ ]\!](\sigma) \quad \textbf{IN} \\
\quad \textbf{LET}\ \ with\_item_2', \sigma_2 \ \ = \ trans_W[\![\ with\_item_2\ ]\!](\sigma_1) \quad \textbf{IN} \\
\quad \ldots \\
\quad \textbf{LET}\ \ with\_item_n', \sigma_n \ \ = \ trans_W[\![\ with\_item_n\ ]\!](\sigma_{n-1}) \quad \textbf{IN} \\
\quad (with\_item_1' \mathrel{\textbf{::}} with\_item_2' \mathrel{\textbf{::}} ... \mathrel{\textbf{::}} [with\_item_n'], \sigma_n)
\end{array}
$$

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

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

$$
\begin{array}{l}
trans_W[\![\ expr_1\ \texttt{as}\ \ expr_2\ ]\!](\text{smodenv}) = \\
\quad \textbf{IF}\ \ \sigma(\textbf{“tensor\_flow”}) \ \ = id_1 \quad \textbf{AND}\ \ expr_1 \ \ = id_1\texttt{.GradientTape()} \quad \textbf{AND}\ \ expr_2 \ \ = id_2 \quad \textbf{THEN} \\
\qquad (expr_1\ \texttt{as}\ \ expr_2, \sigma[\textbf{“gradient\_tape”} \mapsto id_2]) \\
\quad \textbf{ELSE}\ \ (trans_E[\![\ expr_1\ ]\!](\sigma)\ \texttt{as}\ \ expr_2, \sigma)
\end{array}
$$

$$\boxed{trans_C \ \ \textbf{:} \ \ MatchCase \ \to \ \Sigma \ \to \ MatchCase}$$

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

$$\boxed{trans_P \ \ \textbf{:} \ \ Pattern \ \to \ \Sigma \ \to \ Pattern}$$

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

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

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

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

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

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

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

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

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

2.4 Rule: TensorFlow 2.x GradientTape API Pattern

2.4.1 Transformation Rules

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

$$trans_M⟦\ stmt^* \quad type\_ignore\ ⟧ \quad = \quad trans_{\overline{S}}⟦\ stmt^*\ ⟧(\sigma).\_\mathbf{1} \ \ type\_ignore$$

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

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

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

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

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

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

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

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

A strict form of assignment statements

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

LET id_z = **NewID() IN**

([id_z = $expr_{11}$,

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

`global hvd_broadcast_done`,

`if not hvd_broadcast_done:` [`hvd.broadcast_variables([x[1] for x in` id_z`], root_rank=0)`,

`hvd.broadcast_variables(`id_t`.variables(), root_rank=0)`,

`hvd_broadcast_done = True`]], 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** $1 \leq i \leq k$ **THEN**

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

... (id_k =)? $expr_{2k}$) (`#type:`s)?], 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** $1 \leq i \leq k$ **THEN**

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

... (id_k =)? $expr_{2k}$) (`#type:`s)?], 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** $1 \leq i \leq k$ **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 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**

([], σ)

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**)?], σ)

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**)?], σ)

$trans_S[\![$ (`#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())` `:`

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

ELSE ([(`#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**)?], σ)

$trans_S[\![$ `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**)?], σ)

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

([`if` `(`$trans_E[\![$ $expr$ $]\!](\sigma)$`)` `:` $trans_{\overline{S}}[\![$ $stmt^*$ $]\!](\sigma)$.**_1** (`else` `:` $trans_{\overline{S}}[\![$ $stmt^*$ $]\!](\sigma)$.**_1**)?], 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 : $trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma)$.**_1** $(trans_H[\![exc\_handler]\!](\sigma))^*$
(else : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma)$.**_1**)? (finally : $trans_{\overline{S}}[\![$ $stmt_3^*$ $]\!](\sigma)$.**_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` $id^*], \sigma)$

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

IF id_i $=$ `grads_and_vars` **WHEN** $1 \le \mathrm{i} \le \mathrm{k}$ **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 $=$ σ(**"tensor_flow"**) **AND**

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

 # Root Rank Blocking

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

id_t $=$ σ(**"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

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

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`$], \sigma)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  # Adjust Steps

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

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

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

    ELSE

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

  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`)?` `(:`$expr_2$`)?`$\}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\quad$ **ELSE** σ

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

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

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

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

$\quad$ **ELSE** σ

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

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

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

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

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

$\quad \ldots$

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

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

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

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

$trans_W[\![\ expr_1$ `as` $expr_2\ ]\!](\mathrm{smodenv}) =$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2.2 Training Loop

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

$summary_M[\![\ id\ \ stmt^*\ \ \ type\_ignore\ ]\!] =$

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

$\quad \texttt{ModuleSummary}\ \ id\ \ \sigma\ \ tl$

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

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

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

$summary_S[\![\ (@expr_1)^*\ \ \ \texttt{def}\ \ id\ \ \texttt{(}args\texttt{)}\ \ \ (\texttt{->}\ expr_2)?\ \ \ \texttt{:}\ \ (\texttt{\#type:}s)?\ \ stmt^*\ ]\!](\sigma) =$

$\quad \textbf{LET}\ \ \sigma', tl\ \ = summary_{\overline{S}}[\![\ stmt^*\ ]\!]\ \ \ \textbf{IN}$

$\quad (\sigma[id \mapsto \texttt{FuncSummary}\ \ tl], \bot)$

$summary_S[\![\ (@expr_1)^*\ \ \ \texttt{async def}\ \ id\ \ \texttt{(}args\texttt{)}\ \ \ (\texttt{->}\ expr_2)?\ \ \ \texttt{:}\ \ (\texttt{\#type:}s)?\ \ stmt^*\ ]\!](\sigma) =$

$\quad \textbf{LET}\ \ \sigma', tl\ \ = summary_{\overline{S}}[\![\ stmt^*\ ]\!]\ \ \ \textbf{IN}$

$\quad (\sigma[id \mapsto \texttt{FuncSummary}\ \ tl], \bot)$

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

$\quad \textbf{IF}\ \ expr_{1i}\ \ \ =\ id_c\texttt{.keras.Model}\ \ \ \textbf{WHEN}\ \ 1 \leq \text{i} \leq \text{n}\ \ \ \textbf{AND}$

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

$\quad \textbf{ELIF}\ \ expr_{2i}\ \ \ =\ id_c\texttt{.keras.Model}\ \ \ \textbf{WHEN}\ \ 1 \leq \text{i} \leq \text{k}\ \ \ \textbf{AND}$

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

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

A strict form of assignment statements

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

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

$\quad \textbf{ELSE}\ \ (\sigma, \bot)$

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

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

$\quad (\sigma, tl)$

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

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

$\quad (\sigma, tl)$

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

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

$\quad (\sigma, tl)$

$$
\begin{array}{l}
\mathit{summary}_S[\![\ (\texttt{\#type:}s)?\ \texttt{with}\ \ \mathit{with\_item}^*\ \ \texttt{:}\ \ \mathit{stmt}^*\ ]\!](\sigma) = \\
\quad \textbf{LET}\ \ \_,\mathit{tl}\ \ =\ \ \mathit{summary}_{\overline{S}}[\![\ \mathit{stmt}^*\ ]\!](\sigma)\quad \textbf{IN} \\
\quad (\sigma,\ \mathit{summary}_{\overline{W}}[\![\ \mathit{with\_item}^*\ ]\!](\sigma) \sqcup\ \ \mathit{tl})
\end{array}
$$

$$
\begin{array}{l}
\mathit{summary}_S[\![\ (\texttt{\#type:}s)?\ \texttt{async with}\ \ \mathit{with\_item}^*\ \ \texttt{:}\ \ \mathit{stmt}^*\ ]\!](\sigma) = \\
\quad \textbf{LET}\ \ \_,\mathit{tl}\ \ =\ \ \mathit{summary}_{\overline{S}}[\![\ \mathit{stmt}^*\ ]\!](\sigma)\quad \textbf{IN} \\
\quad (\sigma,\ \mathit{summary}_{\overline{W}}[\![\ \mathit{with\_item}^*\ ]\!](\sigma) \sqcup\ \ \mathit{tl})
\end{array}
$$

$$
\begin{array}{l}
\mathit{summary}_S[\![\ \texttt{try}\ \ \texttt{:}\ \ \mathit{stmt}_1^*\ \ \mathit{exc\_handler}^*\ \ (\texttt{else}\ \ \texttt{:}\ \ \mathit{stmt}_2^*)?\ \ (\texttt{finally}\ \ \texttt{:}\ \ \mathit{stmt}_3^*)?\ ]\!](\sigma) = \\
\quad (\sigma,\ \mathit{summary}_{\overline{S}}[\![\ \mathit{stmt}_1^*\ ]\!](\sigma))
\end{array}
$$

$$
\mathit{summary}_S[\![\ \texttt{import}\ \ \mathit{alias}^*\ ]\!](\sigma) = (\mathit{summary}_{\overline{A}}[\![\ \mathit{alias}^*\ ]\!](\sigma),\ \bot)
$$

$$
\begin{array}{l}
\mathit{summary}_S[\![\ \texttt{from}\ \ 0\ \ \mathit{id}^*\ \ \texttt{import}\ \ \mathit{alias}^*\ ]\!](\sigma) = \\
\quad \textbf{LET}\ \ \sigma'\ \ =\ \ \mathit{summary}_{\overline{A}}[\![\ \mathit{alias}^*\ ]\!](\sigma)\quad \textbf{IN} \\
\quad \textbf{LET}\ \ [\mathit{id}_2 \mapsto \texttt{ModuleSummary}\ \ \mathit{id}_2\ \sigma_2\ \bot,\ \ \mathit{id}_2 \mapsto \texttt{ModuleSummary}\ \mathit{id}_2\ \sigma_2\ \bot, \\
\qquad \textbf{...}\ \mathit{id}_n \mapsto \texttt{ModuleSummary}\ \ \mathit{id}_n\ \sigma_n\ \bot]\ \ =\ \ \sigma'\ \ \setminus\ \ \sigma\ \ \textbf{IN} \\
\quad \sigma\ \ {+}{+}\ \ [\mathit{id}_2 \mapsto \texttt{ModuleSummary}\ \ (\mathit{id}^*{+}\mathit{id}_2)\ \sigma_2\ \bot,\ \ \mathit{id}_2 \mapsto \texttt{ModuleSummary}\ \ (\mathit{id}^*{+}\mathit{id}_2)\ \sigma_2\ \bot, \\
\qquad \textbf{...}\ \mathit{id}_n \mapsto \texttt{ModuleSummary}\ \ (\mathit{id}^*{+}\mathit{id}_n)\ \sigma_n\ \bot]
\end{array}
$$

$$
\mathit{summary}_S[\![\ \mathit{expr}\ ]\!](\sigma) = (\sigma,\ \mathit{summary}_E[\![\ \mathit{expr}\ ]\!](\sigma))
$$

$$
\mathit{summary}_S[\![\ \mathit{stmt}\ ]\!](\sigma) = (\sigma,\ \bot)
$$

$$
\boxed{\mathit{summary}_{\overline{A}}\ \ \textbf{:}\ \ \mathit{Alias}\ \ \textbf{list}\ \ \to\ \ \Sigma\ \ \to\ \ \Sigma}
$$

$$
\begin{array}{l}
\mathit{summary}_{\overline{A}}[\![\ \mathit{alias}_1\ ...\ \mathit{alias}_{n-1}\ \mathit{alias}_n\ ]\!](\sigma) = \\
\quad \mathit{summary}_A[\![\ \mathit{alias}_n\ ]\!](\mathit{summary}_A[\![\ \mathit{alias}_{n-1}\ ]\!](...\ \mathit{summary}_A[\![\ \mathit{alias}_1\ ]\!](\sigma)\ ...))
\end{array}
$$

$$
\boxed{\mathit{trans}_A\ \ \textbf{:}\ \ \mathit{Alias}\ \ \to\ \ \Sigma\ \ \to\ \ \Sigma}
$$

$$
\begin{array}{l}
\mathit{trans}_A[\![\ \mathit{id}_1\ \mathit{id}_2\ ...\ \mathit{id}_n\ ]\!](\sigma) = \\
\quad \textbf{LET}\ \ \sigma_n = [\mathit{id}_n \mapsto \texttt{ModuleSummary}\ \ \mathit{id}_n\ \ \ \top\ \ \bot] \\
\quad \textbf{LET}\ \ \sigma_{n-1} = [\mathit{id}_{n-1} \mapsto \texttt{ModuleSummary}\ \ \mathit{id}_{n-1}\ \ \ \sigma_n\ \ \ \bot] \\
\quad \textbf{...} \\
\quad \textbf{LET}\ \ \sigma_2 = [\mathit{id}_2 \mapsto \texttt{ModuleSummary}\ \ \mathit{id}_2\ \ \ \sigma_3\ \ \ \bot] \\
\quad \textbf{LET}\ \ \sigma_1 = [\mathit{id}_1 \mapsto \texttt{ModuleSummary}\ \ \mathit{id}_1\ \ \ \sigma_2\ \ \ \bot] \\
\quad \sigma\ \ {+}{+}\ \ \sigma_1
\end{array}
$$

$$
\boxed{\mathit{summary}_{\overline{W}}\ \ \textbf{:}\ \ \mathit{WithItem}\ \ \textbf{list}\ \ \to\ \ \Sigma\ \ \to\ \ \mathit{tl}}
$$

$$
\begin{array}{l}
\mathit{summary}_{\overline{W}}[\![\ \mathit{with\_item}_1\ \mathit{with\_item}_2\ ...\ \mathit{with\_item}_n\ ]\!](\sigma) = \\
\quad \mathit{summary}_W[\![\ \mathit{with\_item}_1\ ]\!](\sigma) \sqcup\ \ \mathit{summary}_W[\![\ \mathit{with\_item}_2\ ]\!](\sigma) \sqcup\ \ \textbf{...}\ \ \ \mathit{summary}_W[\![\ \mathit{with\_item}_n\ ]\!](\sigma)
\end{array}
$$

$$
\boxed{\mathit{summary}_W\ \ \textbf{:}\ \ \mathit{WithItem}\ \ \to\ \ \Sigma\ \ \to\ \ \mathit{tl}}
$$

$$
\mathit{summary}_W[\![\ \mathit{expr}_1\ \texttt{as}\ \ \mathit{expr}_2?\ ]\!](\sigma) = \mathit{summary}_E[\![\ \mathit{expr}_1\ ]\!](\sigma)
$$

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

A strict form of call expression

$summary_E⟦\ expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $⟧(\sigma) =$

IF $\sigma(id_r)$ $=$ `ValueSummary` **"model_instance"** **AND**

$expr_1$ $=$ id_r`.fit` **THEN** *Optimizer*

ELIF $\sigma(id_r)$ $=$ `ModuleSummary` Id("tensorflow") σ' tl **AND**

$expr_1$ $=$ id_r`.GradientTape` **THEN** *GradTape*

ELIF $\sigma(id_r)$ $=$ `FuncSummary` tl **AND**

$expr_1$ $=$ `idr` **THEN** tl

ELSE $\perp$

$summary_E⟦\ expr\ ⟧(\sigma) = \perp$