Code Transformation for Distributed Python ML Code

1 Python Abstract Syntax

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ELSE

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

ELSE

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

ELSE

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

 # Checkpoint

ELIF $\sigma($**"tensor_flow"**$)$ `=` id_t **AND** $expr_1$ `=` id_t`.train.Checkpoint` **THEN**

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

 # Default

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$trans_S[\![$ `import` $alias^*\ ]\!](\sigma) =$
LET $\sigma_1 = trans_{\overline{A}}[\![\ alias^*\ ]\!](\sigma)$ **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')`$], \sigma_1)$
ELSE $([$`import` $alias^*], \sigma_1)$

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

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

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

A strict form of expr statements

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

 # Config

IF $id_t\ =\ \sigma(\text{“}\textbf{tensor\_flow}\text{”})$ **AND**

$expr_1\ =\ id_t\texttt{.config.experimental.set\_visible\_devices}$ **THEN** $([], \sigma)$

 # model.compile

ELIF $id_m\ =\ \sigma(\text{“}\textbf{model}\text{”})$ **AND** $expr_1\ =\ id_m\texttt{.compile}$ **THEN**

 # string "adam" case

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

$([\texttt{optim = tf.optimizers.Adam(learning\_rate=0.001 * hvd.size),}$
$\texttt{optim = hvd.DistributedOptimizer(optim),}$
$expr_1\ \texttt{(}expr_{11}\ \texttt{...}\quad expr_{1n}\ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ id_i\ \texttt{=}\ \texttt{optim}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}], \sigma)$

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

$([\texttt{optim = tf.optimizers.Adam(learning\_rate=0.001 * hvd.size),}$
$\texttt{optim = hvd.DistributedOptimizer(optim),}$
$expr_1\ \texttt{(optim ...}\quad expr_{1n}\ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}], \sigma)$

ELSE

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

 # Root Rank Blocking

ELIF $id_m\ =\ \sigma(\text{“}\textbf{model}\text{”})$ **AND** $(expr_1\ =\ id_m\texttt{.write}$ **OR**

$expr_1\ =\ id_m\texttt{.summary}$ **OR** $expr_1\ =\ id_m\texttt{.save\_weights}$ **OR**

$expr_1\ =\ expr\texttt{.load\_weights}$ **OR** $expr_1\ =\ id_m\texttt{.save})$ **OR**

$id_c\ =\ \sigma(\text{“}\textbf{checkpoint}\text{”})$ **AND** $expr_1\ =\ id_c\texttt{.save}$ **OR**

$id_t\ =\ \sigma(\text{“}\textbf{tensor\_flow}\text{”})$ **AND** $expr_1\ =\ id_t\texttt{.print}$ **OR**

$expr_1\ =\ \texttt{print}$ **THEN**

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

ELSE

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

A general form of expr statements except for the strict form

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.4 Rule: TensorFlow 2.x GradientTape API Pattern

2.4.1 Transformation Rules

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

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

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

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

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

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

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

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

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

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

A strict form of assignment statements

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

 # Dataset

IF $\sigma(\textbf{“tensor\_flow”}) \ = \ id_t$ **AND** $expr_1 \ = \ id_t\texttt{.data.Dataset.}expr_3$ **THEN**

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

 # Variable Broadcasting

ELIF $\sigma(\textbf{“optimizer”}) \ = \ id_t$ **AND** $expr_1 \ = \ id_t\texttt{.apply\_gradients}$ **THEN**

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

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

$([id_z \ \texttt{=}\ \ expr_{2i},$
$id_r\ \texttt{=}\ expr_1\ \texttt{(}expr_{11}\ ...\ expr_{1n}\ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ id_i\ \texttt{=}\ id_z\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}\ (\texttt{\#type:}s)?,$
`global hvd_broadcast_done,`
`if not hvd_broadcast_done: [ hvd.broadcast_variables([x[1] for x in` id_z`], root_rank=0),`
$\quad$`hvd.broadcast_variables(`id_t`.variables(), root_rank=0),`
$\quad$`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 ≤ i ≤ 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 ≤ i ≤ 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 ≤ i ≤ 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 $\sigma($**"os"**$)$ $=$ id_{os} **AND** $expr_0$ $=$ id_{os}`.environ ['CUDA_VISIBLE_DEVICES']` **THEN** $([], \sigma)$

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

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

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

 # Dataset

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

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

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

 # Adjust Steps

IF $\sigma($**"config"**$)$ $=$ id_c **AND** $(expr_2$ $=$ `range(`id_c`.iterations_per_epoch)` **OR**
$expr_2$ $=$ `tqdm(range(`id_c`.iterations_per_epoch)))` **THEN**
$([$`(#type:`s`)`$?$ `for` $expr_1$ `in range(`id_c`.iterations_per_epoch // hvd.size())` `:`
$trans_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma)$**._1** `(else :` $trans_{\overline{S}}[\![\ stmt_2^*\ ]\!](\sigma)$**._1**`)`$?], \sigma)$

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

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

 # Adjust Steps

IF $\sigma($**"config"**$)$ $=$ id_c **AND** $(expr_2$ $=$ `range(`id_c`.iterations_per_epoch)` **OR**
$expr_2$ $=$ `tqdm(range(`id_c`.iterations_per_epoch)))` **THEN**
$([$`(#type:`s`)`$?$ `async for` $expr_1$ `in range(`id_c`.iterations_per_epoch // hvd.size())` `:`
$trans_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma)$**._1** `(else :` $trans_{\overline{S}}[\![\ stmt_2^*\ ]\!](\sigma)$**._1**`)`$?], \sigma)$

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**`)`$?], \sigma)$

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

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

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

A strict form of expr statements

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

 # Variable Broadcasting

IF $\sigma($**"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**

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

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

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 ]]`, $\sigma)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$trans_O$: $Comprehension \rightarrow \Sigma \rightarrow Comprehension$

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

$trans_H$: $ExcHandler \rightarrow \Sigma \rightarrow ExcHandler$

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

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

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

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

$trans_A[\![\ id\ ]\!](\sigma) =$
  LET id = `tensorflow` **THEN** $\sigma[$**"tensor_flow"**$\mapsto id]$
  LET id = `os` **THEN** $\sigma[$**"os"**$\mapsto id]$
  LET id = `config` **THEN** $\sigma[$**"config"**$\mapsto id]$
  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"**$\mapsto id_2]$
  LET id = `config` **THEN** $\sigma[$**"config"**$\mapsto id_2]$
  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**
  $(with\_item_1'$ **::** $with\_item_2'$ **::** ... **::** $[with\_item_n'], \sigma_n)$

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

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

$trans_W[\![\ expr_1$ `as` $expr_2\ ]\!](\text{smodenv}) =$
  IF $\sigma($**"tensor_flow"**$) = id_1$ **AND** $expr_1 = id_1$`.GradientTape()` **AND** $expr_2 = id_2$ **THEN**
    $(expr_1$ `as` $expr_2, \sigma[$**"gradient_tape"**$\mapsto id_2])$
  ELSE $(trans_E[\![\ expr_1\ ]\!](\sigma)$ `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` $trans_E[\![\ expr\ ]\!](\sigma))?$ `:` $trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)$**._1**

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

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

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

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

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

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

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

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

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

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

2.5 Rule: TensorFlow 1.x Mainscript API Pattern

2.5.1 Transformation Rules

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

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

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

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

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

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

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

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

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

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

A strict form of assignment statements

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

 # Config Proto

IF id_t = **"tensor_flow"** **AND** $expr_1$ = id_t`.ConfigProto` **THEN**

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

 # Minimize

 # 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

ELIF $expr_1$ `<: tensorflow.keras.optimizers.Optimizer` **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:`$s)?$
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** $1 \leq$ i $\leq$ k **THEN**

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

ELSE

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

 # Checkpoint
ELIF $\sigma($**"tensor_flow"**$)$ $=$ id_t **AND** $expr_1$ $=$ id_t`.train.Checkpoint` **THEN**
$([id_r$ `=` $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $(\texttt{\#type:}s)?]$,
$\sigma[$**"checkpoint"**$\mapsto id_r])$
 # Default
ELSE $([id_r$ `=` $trans_E[\![$ $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_2$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $]\!](\sigma)$ $(\texttt{\#type:}s)?]$,
$\sigma)$

$trans_S[\![$ $expr_0$ `=` $expr_1$ $(\texttt{\#type:}s)?$ $]\!](\sigma) =$
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
$trans_S[\![$ $expr_1^*$ `=` $expr_2$ $(\texttt{\#type:}s)?$ $]\!](\sigma) = ([expr_1^*$ `=` $trans_E[\![$ $expr_2$ $]\!](\sigma)$ $(\texttt{\#type:}s)?], \sigma)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A strict form of expr statements
$trans_S[\![\ expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $]\!](\sigma) =$
 # 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:` $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)`$], \sigma)$
ELSE
$[trans_E[\![\ expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $]\!](\sigma)], \sigma)$

A general form of expr statements except for the strict form
$trans_S[\![\ expr\ ]\!](\sigma) = ([trans_E[\![\ expr\ ]\!](\sigma)], \sigma)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\boxed{trans_A\ \ \texttt{:}\ \ Alias\ \rightarrow\ \Sigma\ \rightarrow\ \Sigma}$
$trans_A[\![\ id\ ]\!](\sigma) =$
$\quad\textbf{LET}\ \ id\ =\ \texttt{tensorflow.compat.v1}\ \ \textbf{THEN}\ \ \sigma[\textbf{“tensor\_flow\_compat”}\mapsto id]$
$\quad\textbf{LET}\ \ id\ =\ \texttt{tensorflow}\ \ \textbf{THEN}\ \ \sigma[\textbf{“tensor\_flow”}\mapsto id]$
$\quad\textbf{LET}\ \ id\ =\ \texttt{os}\ \ \textbf{THEN}\ \ \sigma[\textbf{“os”}\mapsto id]$
$\quad\textbf{ELSE}\ \ \sigma$

$trans_A[\![\ id_1\ \texttt{as}\ \ id_2\ ]\!](\sigma) =$
  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**
  $(with\_item_1'$ **::** $with\_item_2'$ **::** ... **::** $[with\_item_n'], \sigma_n)$

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

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

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

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

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

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

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

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

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

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

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

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

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

$Summary$	::=	`ModuleSummary` $id\ \sigma_\top\ tl$	(ModuleSummary)
		`FuncSummary` tl	(FuncSummary)
		`ClassSummary` arg	(ClassSummary)
		`ValueSummary` s	(ValueSummary)
arg	::=	$Model \mid \bot$	(Arg)
tl	::=	$GradTape \mid Optimizer \mid \bot$	(TrainingLoop)
$\sigma_\top$	::=	$\sigma\ \cup \top$	

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

3.2.2 Training Loop

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

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

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

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

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

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

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

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

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"**$], \bot)$
ELSE $(\sigma, \bot)$

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

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

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

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

$summary_{\overline{A}}$ **:** $Alias$ **list** $\rightarrow$ Σ $\rightarrow$ Σ

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

$trans_A$ **:** $Alias$ $\rightarrow$ Σ $\rightarrow$ Σ

$trans_A[\![$ id_1 id_2 ... id_n $]\!](\sigma) =$
LET $\sigma_n = [id_n \mapsto$ `ModuleSummary` id_n $\top$ $\bot]$
LET $\sigma_{n-1} = [id_{n-1} \mapsto$ `ModuleSummary` id_{n-1} σ_n $\bot]$
...
LET $\sigma_2 = [id_2 \mapsto$ `ModuleSummary` id_2 σ_3 $\bot]$
LET $\sigma_1 = [id_1 \mapsto$ `ModuleSummary` id_1 σ_2 $\bot]$
σ $++$ σ_1

$summary_{\overline{W}}$ **:** $WithItem$ **list** $\rightarrow$ Σ $\rightarrow$ tl

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

$summary_W$ **:** $WithItem$ $\rightarrow$ Σ $\rightarrow$ tl

$summary_W[\![$ $expr_1$ `as` $expr_2?$ $]\!](\sigma) = summary_E[\![$ $expr_1$ $]\!](\sigma)$

$summary_E$ **:** $Expr$ $\rightarrow$ Σ $\rightarrow$ 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 $\bot$

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