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)

2

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 for TF2 Python 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

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

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

$$
\begin{array}{llll}
\textbf{.\_1} & : & \tau \times \tau \to \tau & \text{GET THE FIRST ELEMENT OF THE GIVEN PAIR} \\
\textbf{@} & : & \tau \text{ list} \to \tau \text{ list} \to \tau \text{ list} & \text{CONCATENATE TWO LISTS} \\
\textbf{::} & : & \tau \to \tau \text{ list} \to \tau \text{ list} & \text{APPEND AN ELEMENT TO A LIST (RIGHT-ASSOCIATIVE)}
\end{array}
$$

2.2.2 Transformation Rules

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

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

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

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

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

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

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

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

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

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

A strict form of assignment statements

$trans_S[\![\ id_r\ \texttt{=}\ expr_1\ (expr_{11}\ ...\ expr_{1n}\ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ (\texttt{\#type:}s)?\ ]\!](\sigma) =$
$\quad \textbf{IF}\ \sigma(\text{``tensor\_flow''})\ =\ id_t\ \ \textbf{AND}\ \ expr_1\ =\ id_t.\texttt{estimator.Estimator}\ \ \textbf{THEN}$
$\quad\quad ([id_r\ \texttt{=}\ expr_1\ (expr_{11}\ ...\ expr_{1n}\ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ (\texttt{\#type:}s)?], \sigma[\text{``classifier''} \mapsto id_r])$
$\quad \textbf{ELIF}\ \sigma(\text{``tensor\_flow''})\ =\ id_t\ \ \textbf{AND}\ \ expr_1\ =\ id_t.\texttt{train.MomentumOptimizer}\ \ \textbf{THEN}$
$\quad\quad \textbf{IF}\ id_i\ =\ \texttt{learning\_rate}\ \ \textbf{WHEN}\ 1 \leq \text{i} \leq \text{k}\ \ \textbf{THEN}$
$\quad\quad\quad ([id_r\ \texttt{=}\ expr_1\ (expr_{11}\ ...\ expr_{1n}\ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ id_i\ \texttt{=}\ expr_{2i}\ \texttt{* hvd.size()}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ (\texttt{\#type:}$
$\quad\quad\quad id_r\ \texttt{= hvd.DistributedOptimizer}(id_r, \texttt{backward\_passes\_per\_step=1})],$
$\quad\quad\quad \sigma[\text{``optimizer''} \mapsto id_r]\ )$
$\quad\quad \textbf{ELSE}$
$\quad\quad\quad ([id_r\ \texttt{=}\ expr_1\ (expr_{11}\ \texttt{* hvd.size()}\ ...\ expr_{1n}\ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ (\texttt{\#type:}s)?,$
$\quad\quad\quad id_r\ \texttt{= hvd.DistributedOptimizer}(id_r, \texttt{backward\_passes\_per\_step=1})],$
$\quad\quad\quad \sigma[\text{``optimizer''} \mapsto id_r]\ )$
$\quad \textbf{ELIF}\ \sigma(\text{``checkpoint''})\ =\ id_t\ \ \textbf{AND}\ \ expr_1\ =\ id_t.\texttt{save}\ \ \textbf{THEN}$
$\quad\quad ([\texttt{if hvd.rank() == 0:}\ [id_r\ \texttt{=}\ expr_1\ (expr_{11}\ ...\ expr_{1n}\ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ (\texttt{\#type:}s)?]],$
$\quad\quad\quad \sigma)$
$\quad \textbf{ELSE}\ ([id_r\ \texttt{=}\ trans_E[\![\ expr_1\ (expr_{11}\ ...\ expr_{1n}\ (id_2\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ ]\!](\sigma)\ (\texttt{\#type:}s)?],$
$\quad\quad\quad \sigma)$

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

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

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

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

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

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

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

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

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

$trans_S[\![\ (\texttt{\#type:}s)?\ \texttt{with}\ with\_item^*\ :\ stmt^*\ ]\!](\sigma) =$
    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**
    IF $\sigma_1 \setminus \sigma = [\text{``\textbf{gradient\_tape}''} \mapsto id]$ **THEN**
        $([(\texttt{\#type:}s)?\ \texttt{with}\ with\_item^{*\prime}\ :\ stmt^{*\prime},$
        $id = \texttt{hvd.DistributedGradientTape}(id)],\sigma_2)$
    ELSE $([(\texttt{\#type:}s)?\ \texttt{with}\ with\_item^{*\prime}\ :\ stmt^{*\prime}],\sigma_2)$

$trans_S[\![\ (\texttt{\#type:}s)?\ \texttt{async with}\ with\_item^*\ :\ stmt^*\ ]\!](\sigma) =$
    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**
    IF $\sigma_1 \setminus \sigma = [\text{``\textbf{gradient\_tape}''} \mapsto id]$ **THEN**
        $([(\texttt{\#type:}s)?\ \texttt{async with}\ with\_item^{*\prime}\ :\ stmt^{*\prime},$
        $id = \texttt{hvd.DistributedGradientTape}(id)],\sigma_2)$
    ELSE $([(\texttt{\#type:}s)?\ \texttt{async with}\ with\_item^{*\prime}\ :\ stmt^{*\prime}],\sigma_2)$

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

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

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

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

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

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

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

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

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

$\quad \textbf{IF}\ \sigma_1\ \setminus\ \sigma\ =\ [\textbf{``tensor\_flow''} \mapsto id]\ \ \textbf{THEN}$

```
      ([import alias*,
      import horovod.tensorflow as hvd,
      hvd_broadcast_done = False,
      hvd.init(),
```
$\quad\quad\quad\texttt{gpus = } id\texttt{.config.experimental.list\_physical\_devices(`GPU'),}$

$\quad\quad\quad\texttt{for gpu in gpus: } id\texttt{.config.experimental.set\_memory\_growth(gpu, True),}$

$\quad\quad\quad\texttt{if gpus: } id\texttt{.config.experimental.set\_visible\_devices(gpus[hvd.local\_rank()], `GPU')]},\ \sigma_1)$

$\quad \textbf{ELSE}\ ([\texttt{import}\ alias^*],\ \sigma_1)$

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

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

$\quad \textbf{IF}\ id\ =\ \texttt{tensorflow}\ \ \textbf{AND}\ \sigma_1\ \setminus\ \sigma\ =\ [\textbf{``keras''} \mapsto id]\ \ \textbf{THEN}$

```
      ([from i id? import alias*,
      import horovod.tensorflow.keras as hvd,
      hvd.init(),
```
$\quad\quad\quad\texttt{gpus = } id\texttt{.config.experimental.list\_physical\_devices(`GPU'),}$

$\quad\quad\quad\texttt{for gpu in gpus: } id\texttt{.config.experimental.set\_memory\_growth(gpu, True),}$

$\quad\quad\quad\texttt{if gpus: } id\texttt{.config.experimental.set\_visible\_devices(gpus[hvd.local\_rank()], `GPU')]},\ \sigma_1)$

$\quad \textbf{ELSE}\ ([\texttt{from}\ i\ id?\ \texttt{import}\ alias^*],\ \sigma)$

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

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

A strict form of expr statements

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

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

        IF $id_i =$ grads_and_vars **WHEN** $1 \leq i \leq k$ **THEN**

            LET $id_z =$ **NewID() IN**

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

            $expr_1 \; (expr_{11} \; ... \; expr_{1n} \quad (id_1 = \,)? \; expr_{21} \; ... \; id_i = id_z \; ... \; (id_k = \,)? \; expr_{2k}),$

            `global hvd_broadcast_done,`

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

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

                `hvd_broadcast_done = True` $\rceil], \sigma)$

        ELSE

            LET $id_z =$ **NewID() IN**

            $([id_z = \; expr_{11},$

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

            `global hvd_broadcast_done,`

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

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

                `hvd_broadcast_done = True` $\rceil], \sigma)$

    ELIF $expr_1 =$ print **THEN**

        $([$`if hvd.rank() == 0:`$,$

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

    ELIF $\sigma($**"checkpoint"**$) = id_t$ **AND** $expr_1 = id_t$.save **THEN**

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

    ELIF $\sigma($**"classifier"**$) = id_t$ **AND** $expr_1 = id_t$.train **THEN**

        IF $id_i =$ hooks **WHEN** $1 \leq i \leq k$ **THEN**

            $([expr_1 \; (expr_{11} \; ... \; expr_{1n} \quad (id_1 = \,)? \; expr_{21} \; ...$

              $id_i = expr_{2i} +$ `[hvd.BroadcastGlobalVariablesHook(0)]`

            $... \; (id_k = \,)? \; expr_{2k})], \sigma)$

        ELSE

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

              `hooks = [hvd.BroadcastGlobalVariablesHook(0)])], ` $\sigma)$

    ELSE

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

A general form of expr statements except for the strict form

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

 LET id = keras   **THEN** $\sigma[$**"keras"**$\mapsto id]$

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

 ELSE σ

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

 LET id_1 = tensorflow.compat.v1   **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"**$\mapsto id_2]$

 ELSE σ

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

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

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

 LET $with\_item_1'$, σ_1 = $trans_W[\![\ with\_item_1\ ]\!](\sigma)$ **IN**

 LET $with\_item_2'$, σ_2 = $trans_W[\![\ with\_item_2\ ]\!](\sigma_1)$ **IN**

 . . .

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

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

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

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

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

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

  $(expr_1\ \text{as}\ expr_2,\ \sigma[$**"gradient_tape"**$\mapsto id_2])$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3 Identifying trainig loop

3.1 Restrictions

1. Training loop must be defined in only one file.

2. Training loop type is either `distributed gradient tape` or `distributed optimizer`.

3. Each model must have only one type of training loop.

4. Function must not be assigned to the variables or passed as an argument.

5. Training loop must not be defined conditionally.

3.2 Rules

3.2.1 Summary

$Summary$	$::=$	`ModuleSummary` $id \ \sigma_\top \ tl$	(MODULESUMMARY)
		`FuncSummary` tl	(FUNCSUMMARY)
		`ClassSummary` arg	(CLASSSUMMARY)
		`ValueSummary` s	(VALUESUMMARY)
arg	$::=$	$Model \ | \ \bot$	(ARG)
tl	$::=$	$GradTape \ | \ Optimizer \ | \ \bot$	(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

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

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

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

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

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

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

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

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

A strict form of assignment statements
$summary_S[\![\ id_r \ \texttt{=} \ id_c \ \texttt{()} \ (\texttt{\#type:}s)?\ ]\!](\sigma) =$
    $\textbf{IF} \ \sigma(id_c) \ = \ \texttt{ClassSummary} \ Model \ \textbf{THEN} \ (\sigma[id_r \mapsto \texttt{ValueSummary} \ \textbf{"model\_instance"}], \perp)$
    $\textbf{ELSE} \ (\sigma, \perp)$

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

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

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

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

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

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

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

$summary_S[\![\ \texttt{from}\ \ 0\ \ id^*\ \ \texttt{import}\ \ alias^*\ ]\!](\sigma) =$
$\quad \textbf{LET}\ \ \sigma'\ =\ summary_{\overline{A}}[\![\ alias^*\ ]\!](\sigma)\ \ \textbf{IN}$
$\quad \textbf{LET}\ \ [id_2 \mapsto \texttt{ModuleSummary}\ id_2\ \sigma_2\ \bot,\ \ id_2 \mapsto \texttt{ModuleSummary}\ id_2\ \sigma_2\ \bot,$
$\qquad ...\ id_n \mapsto \texttt{ModuleSummary}\ id_n\ \sigma_n\ \bot]\ =\ \sigma'\ \setminus\ \sigma\ \ \textbf{IN}$
$\quad \sigma\ \texttt{++}\ \ [id_2 \mapsto \texttt{ModuleSummary}\ (id^*\texttt{+}id_2)\ \sigma_2\ \bot,\ \ id_2 \mapsto \texttt{ModuleSummary}\ (id^*\texttt{+}id_2)\ \sigma_2\ \bot,$
$\qquad ...\ id_n \mapsto \texttt{ModuleSummary}\ (id^*\texttt{+}id_n)\ \sigma_n\ \bot]$

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

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

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

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

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

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

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

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

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

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

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

A strict form of call expression

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

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

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

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

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

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

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

 ELSE $\perp$

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