Latent Programmer:
Discrete Latent Codes for Program Synthesis

Joey Hong
David Dohan
Rishabh Singh
Charles Sutton
Manzil Zaheer

Google Research

Program Synthesis

Goal: Automatically generate programs given some specification that humans can easily provide, i.e. input-output (IO) examples or natural language descriptions

Program Synthesis

Goal: Automatically generate programs given some specification that humans can easily provide, i.e. input-output (IO) examples or natural language descriptions

Example 1: IO → String Transformation

1. "Mason Smith" → "Smith M"
2. "Henry Myers" → "Myers H"
3. "Barry Underwood" → "Underwood B"
4. "sandy Jones" → "Jones S"

GetToken_PROP_CASE_2 | " " | ToCase_UPPER(GetToken_CHAR_1)

Program Synthesis

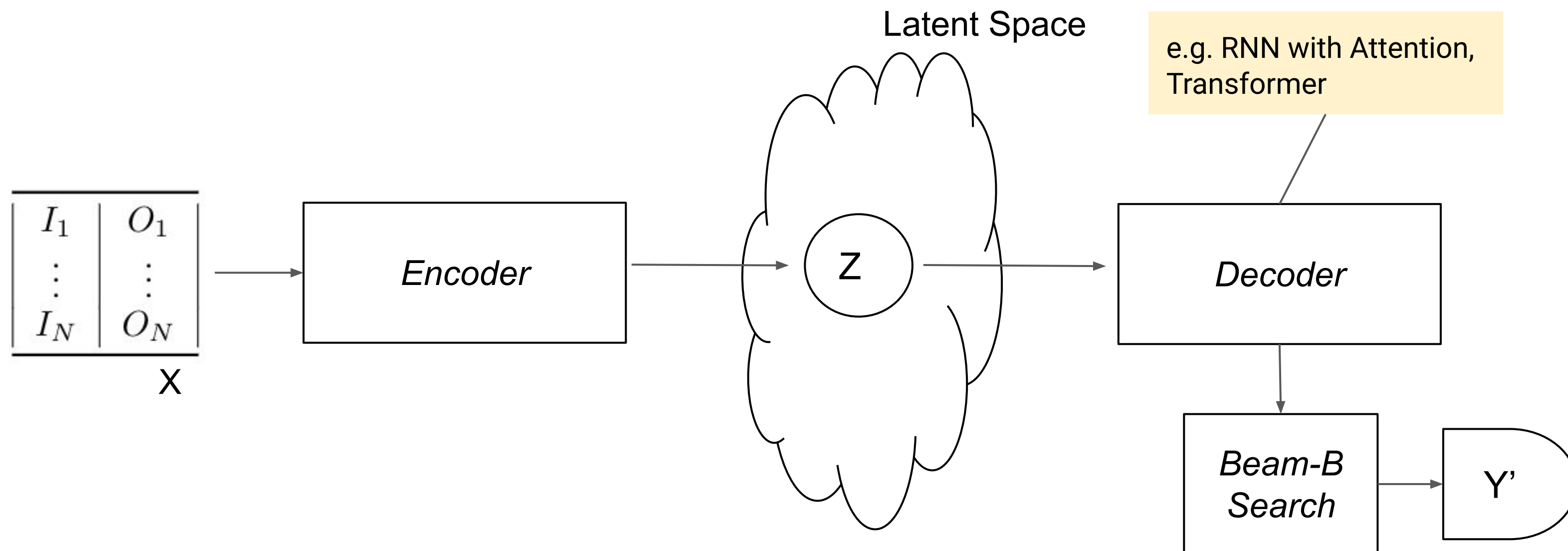
Goal: Automatically generate programs given some specification that humans can easily provide, i.e. input-output (IO) examples or natural language descriptions

Example 2: Natural Language → Python Function

“return a list of words in the string s”

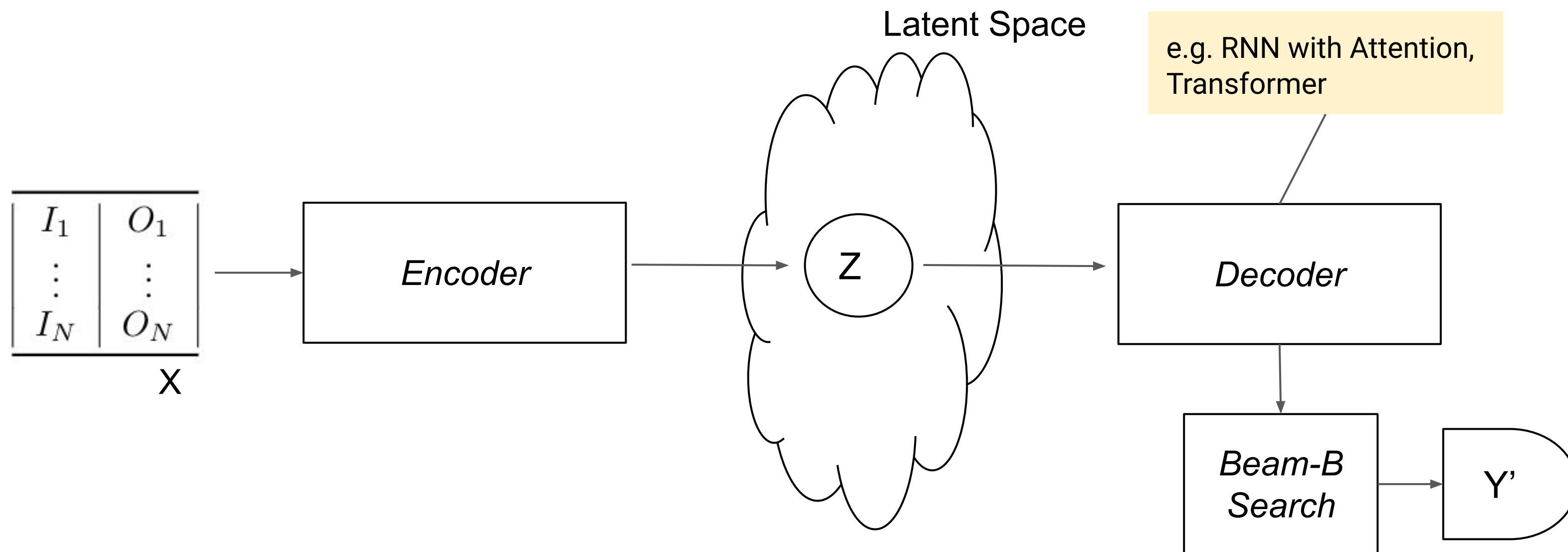
```
def split(s, sep=None, maxsplit=-1):
  return s.split(sep, maxsplit)
```

Neural Program Synthesis



Devlin et al., 2017

Neural Program Synthesis



Problem: Seq-to-seq networks do very well on simple tasks, but fail to infer more complicated programs.

Devlin et al., 2017

Two-Level Search: Motivation

Example problem: Find string transformation that maps inputs to outputs

1. “Jacob,Ethan,James 11” → “11:J.E.J.”
2. “Elijah,Daniel,Aiden 3162” → “3162:E.D.A”
3. “Rick,Oliver,Mia 26” → “26:R.O.M.”
4. “Mark,Ben,Sam 510” → “510:M.B.S.”

How might a person solve this problem?

Two-Level Search: Motivation

Example problem: Find string transformation that maps inputs to outputs

1. “Jacob,Ethan,James 11” → “11:J.E.J.”
2. “Elijah,Daniel,Aiden 3162” → “3162:E.D.A”
3. “Rick,Oliver,Mia 26” → “26:R.O.M.”
4. “Mark,Ben,Sam 510” → “510:M.B.S.”

Intuition: People would first construct a high-level plan for the program, then fills in details of the program based on the plan.

Two-Level Search: Motivation

Example problem: Find string transformation that maps inputs to outputs

1. “Jacob,Ethan,James 11” → “11:J.E.J.”
2. “Elijah,Daniel,Aiden 3162” → “3162:E.D.A”
3. “Rick,Oliver,Mia 26” → “26:R.O.M.”
4. “Mark,Ben,Sam 510” → “510:M.B.S.”

Number | 1st Initial | 2nd Initial | 3rd initial (High-level plan)

GetToken_NUMBER_1 | “:” | GetToken_ALL_CAPS_1 | “.” | GetToken_ALL_CAPS_2 | “.” | GetToken_ALL_CAPS_3 (Low-level program)

Two-Level Search: Plans

In two-level search, we consider generating high-level plan, then conditioned on the plan, perform low-level search over programs.

Two-Level Search: Plans

In two-level search, we consider generating high-level plan, then conditioned on the plan, perform low-level search over programs.

Question: How do we represent plans? Answer: A sequence of discrete tokens.

- Why discrete? Because we can apply standard combinatorial search (i.e. beam search) on the plan space.

Two-Level Search: Plans

In two-level search, we consider generating high-level plan, then conditioned on the plan, perform low-level search over programs.

Question: How do we represent plans? Answer: A sequence of discrete tokens.

- Why discrete? Because we can apply standard combinatorial search (i.e. beam search) on the plan space.

Example 1:

Last Initial | First Name → GetToken | GetToken (Program Sketch)

Murali et al., 2018

Two-Level Search: Plans

In two-level search, we consider generating high-level plan, then conditioned on the plan, perform low-level search over programs.

Question: How do we represent plans? Answer: A sequence of discrete tokens.

- Why discrete? Because we can apply standard combinatorial search (i.e. beam search) on the plan space.

Example 2:

Last Initial | First Name →
GetToken_<HOLE>_<HOLE> | GetToken_WORD_<HOLE> (Program Sketch)

Nye et al., 2019

Two-Level Search: Plans

In two-level search, we consider generating high-level plan, then conditioned on the plan, perform low-level search over programs.

Question: How do we represent plans? Answer: A sequence of discrete tokens.

- Why discrete? Because we can apply standard combinatorial search (i.e. beam search) on the plan space.

Example 3 (this work):

Last Initial | First Name ⟶ TOK_2 | TOK_8 (Latent Code)

Latent Codes

We consider plans that are latent codes, where each token is a discrete latent variable in some learned latent space.

Latent Codes

We consider plans that are latent codes, where each token is a discrete latent variable in some learned latent space.

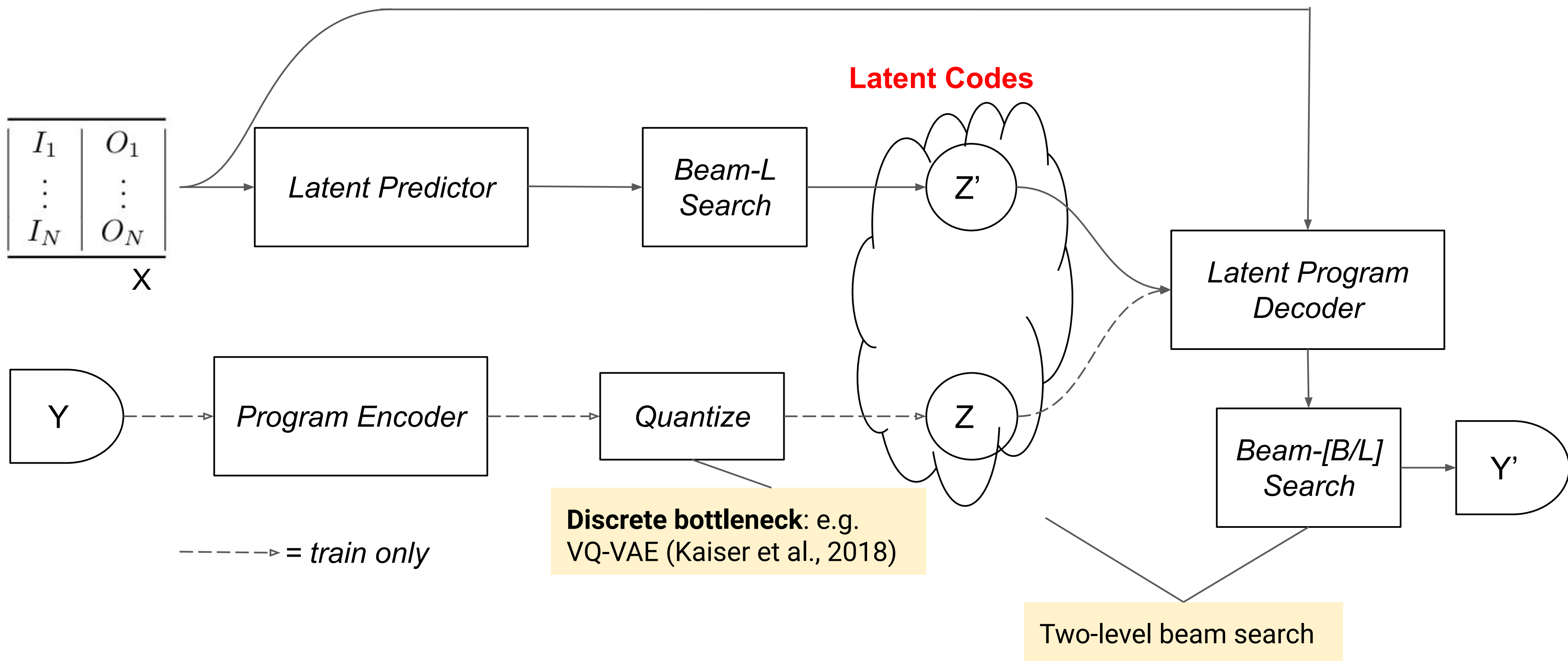
- Latent codes provide generality and flexibility. The model can assign arbitrary meanings to tokens in the latent space.

Latent Codes

We consider plans that are latent codes, where each token is a discrete latent variable in some learned latent space.

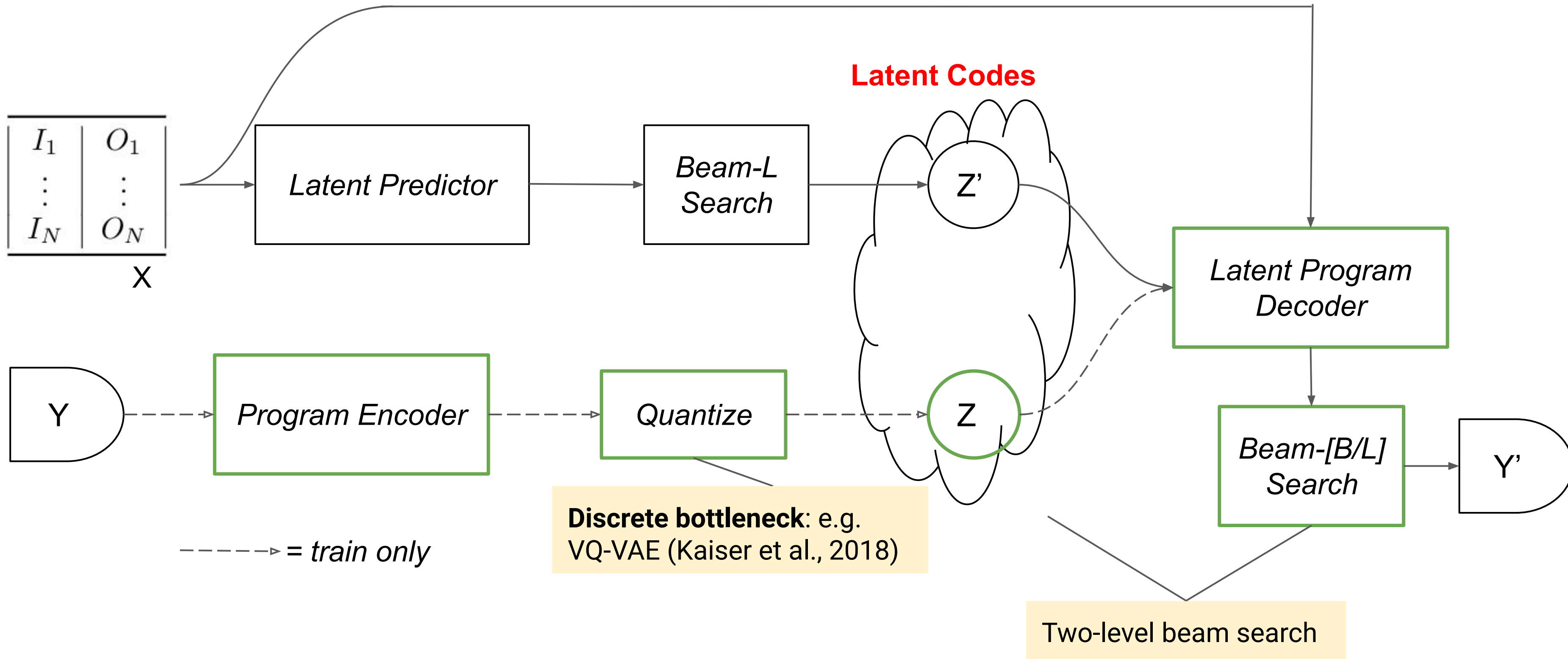
- Latent codes provide generality and flexibility. The model can assign arbitrary meanings to tokens in the latent space.
- How is it learned? Using a supervised technique where a discrete autoencoder generates intermediate latent code targets for the end-to-end prediction task. We consider using a VQ-VAE as the autoencoder (similarly done in Kaiser et al., 2018).

Latent Programmer



Training Loss 1: Fit Y’ to Y using Z (autoencoder loss)

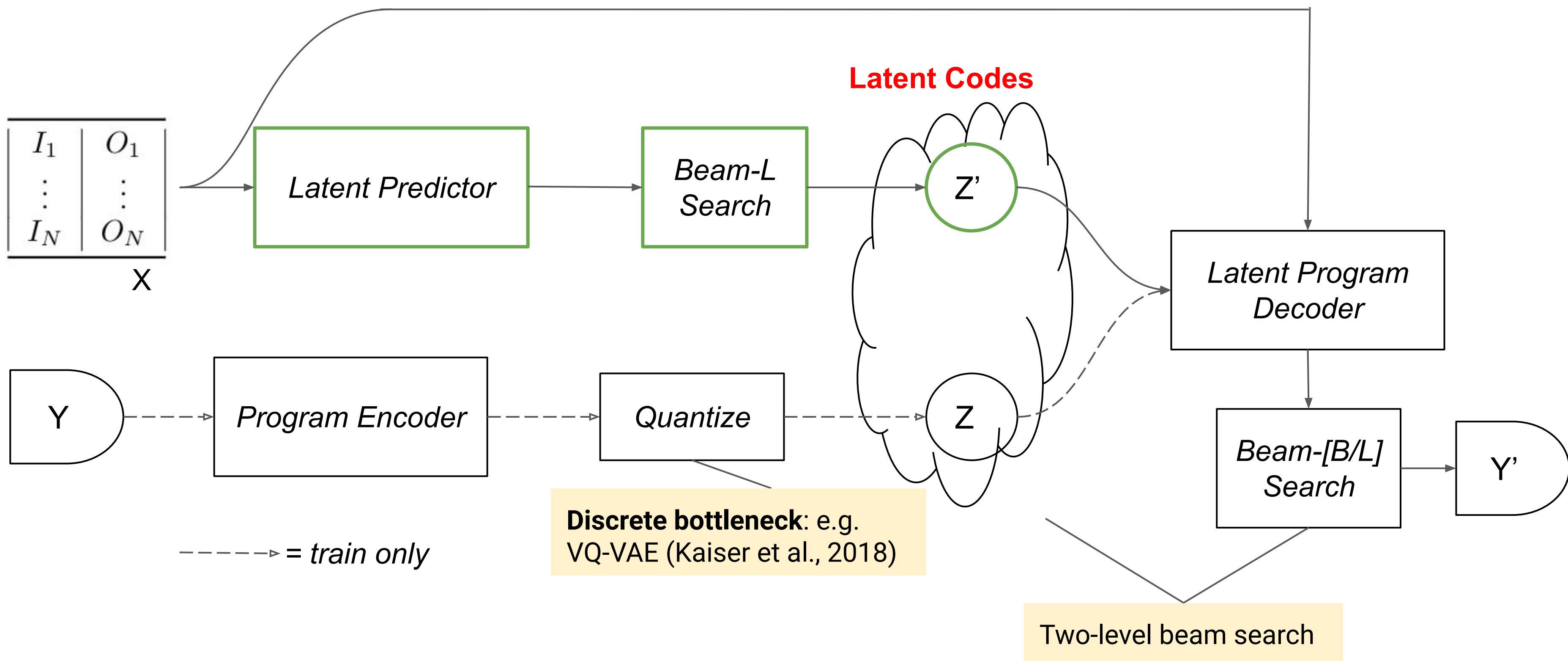
Latent Programmer



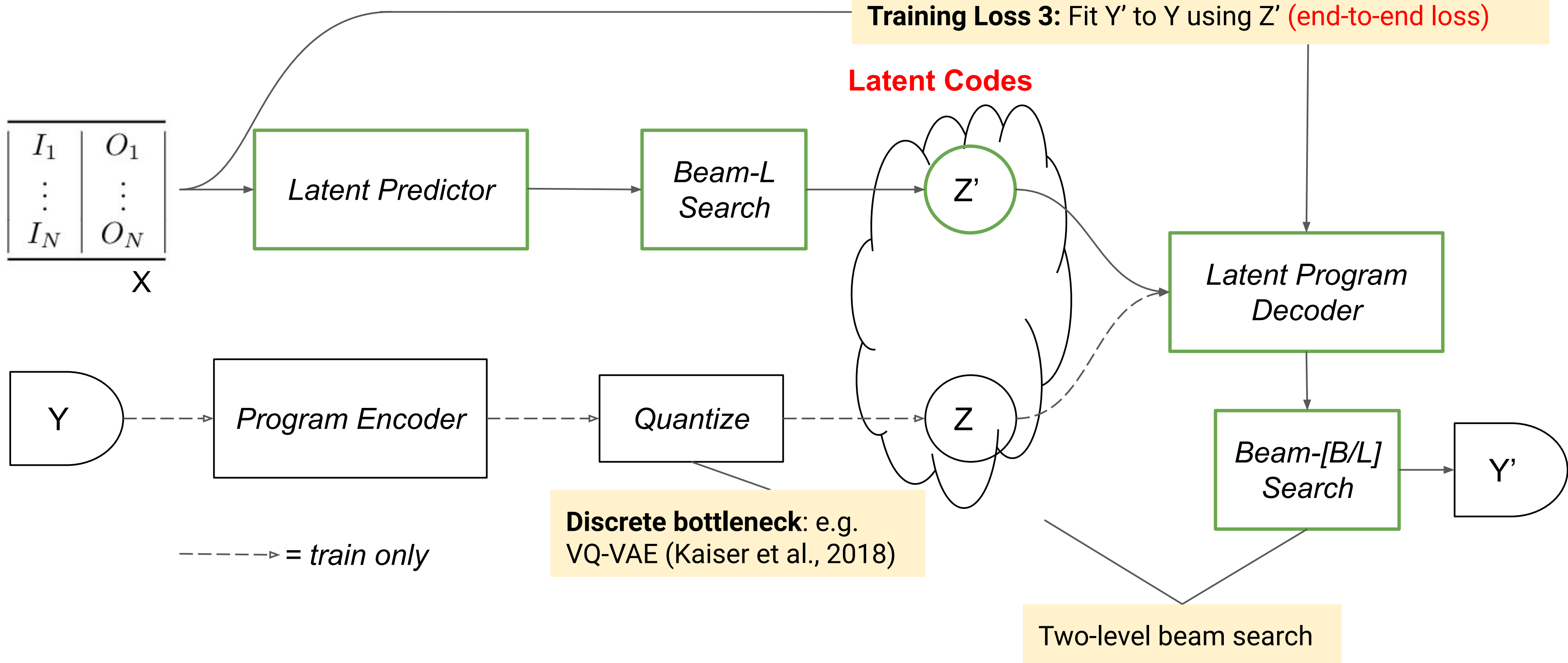
Latent Programmer

Training Loss 1: Fit Y' to Y using Z (autoencoder loss)

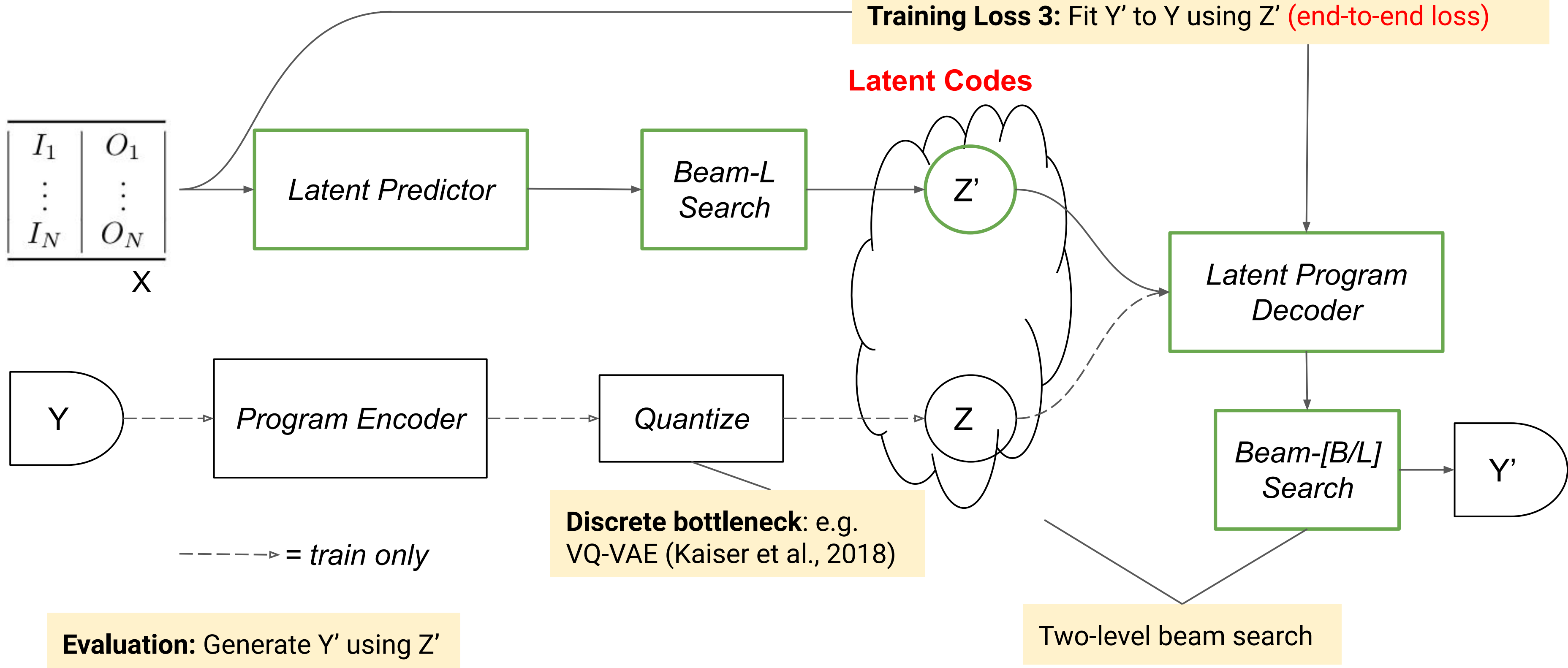
Training Loss 2: Fit Z' to Z (latent prediction loss)



Latent Programmer



Latent Programmer



String Transformation: Setup

String transformation DSL:

$$
\begin{array}{rcl}
\text{Program } Y & := & \texttt{Concat}(e_1, e_2, \ldots) \\
\text{Expression } e & := & f \mid n \mid n_1(n_2) \mid n(f) \mid \texttt{ConstStr}(c) \\
\text{Substring } f & := & \texttt{SubStr}(k_1, k_2) \mid \texttt{GetSpan}(r_1, i_1, b_1, r_2, i_2, b_2) \\
\text{Nesting } n & := & \texttt{GetToken}(t, i) \mid \texttt{ToCase}(s) \mid \texttt{Replace}(\delta_1, \delta_2) \mid \texttt{Trim}() \mid \texttt{GetUpto}(r) \mid \texttt{GetFrom}(r) \\
 & & \mid \texttt{GetFirst}(t, i) \mid \texttt{GetAll}(t) \\
\text{Regex } r & := & t_1 \mid \ldots \mid t_n \mid \delta_1 \mid \ldots \mid \delta_m \\
\text{Type } t & := & \texttt{NUMBER} \mid \texttt{WORD} \mid \texttt{ALPHANUM} \mid \texttt{ALL_CAPS} \mid \texttt{PROP_CASE} \mid \texttt{LOWER} \mid \texttt{DIGIT} \mid \texttt{CHAR} \\
\text{Case } s & := & \texttt{PROPER} \mid \texttt{ALL_CAPS} \mid \texttt{LOWER} \\
\text{Position } k & := & -100 \mid -99 \mid \ldots \mid 1 \mid 2 \mid \ldots \mid 100 \\
\text{Index } i & := & -5 \mid -4 \mid \ldots \mid -1 \mid 1 \mid 2 \mid \ldots \mid 5 \\
\text{Boundary } b & := & \texttt{START} \mid \texttt{END} \\
\text{Delimiter } \delta & := & \&,.?@()[]\%\{\}/ :;\$\#"' \\
\text{Character } c & := & A - Z \mid a - z \mid 0 - 9 \mid \&,.?@\ldots
\end{array}
$$

String Transformation: Setup

Dataset:

- Randomly sampled 2M programs from the DSL of 1-10 expressions
- Each program has 4 randomly generated IO examples: used heuristics to ensure each input mapped to non-empty output

Example problem:

Inputs	Outputs	Program
"Jacob,Ethan,James 11"	"11:J.E.J."	GetToken_NUMBER_1 | Const(:) |
"Elijah,Daniel,Aiden 3162"	"3162:E.D.A"	GetToken_ALL_CAPS_1 | Const(.) |
"Rick,Oliver,Mia 26"	"26:R.O.M."	GetToken_ALL_CAPS_2 | Const(.) |
"Mark,Ben,Sam 510"	"510:M.B.S."	GetToken_ALL_CAPS_3 | Const(.)

String Transformation: Results

Latent Programmer outperforms strong state-of-the-art-baselines

Results from ablation study:

- LSTM vs transformer
- Continuous autoencoder vs. discrete

Method	Accuracy		
	B = 1	10	100
RobustFill [LSTM]	45%	49%	61%
RobustFill [Transformer]	47%	51%	61%
Latent RobustFill [AE]	47%	50%	60%
Latent RobustFill [VAE]	46%	51%	62%
Latent Programmer	**51%**	**57%**	**68%**

Devlin et al., 2017

Comparison to other prior work:

Method	Accuracy
DeepCoder (Balog et al., 2017)	40%
SketchAdapt (Nye et al., 2019)	62%
Latent Programmer	**67%**

Another form of two-level search

String Transformation: Results

Example with long repetitive structure where baseline fails but Latent Programmer recovers the correct program

Inputs	Outputs	Program
"Jacob,Ethan,James 11"	"11:J.E.J."	GetToken_NUMBER_1 | Const(:) |
"Elijah,Daniel,Aiden 3162"	"3162:E.D.A"	GetToken_ALL_CAPS_1 | Const(.) |
"Rick,Oliver,Mia 26"	"26:R.O.M."	GetToken_ALL_CAPS_2 | Const(.) |
"Mark,Ben,Sam 510"	"510:M.B.S."	GetToken_ALL_CAPS_3 | Const(.)

RobustFill	GetAll_NUMBER | Const(:)| GetToken_ALL_CAPS_2 | Const(.)
LP	GetAll_NUMBER | Const(:) | GetToken_ALL_CAPS_1 | Const(.) | GetToken_ALL_CAPS_2 | Const(.) | GetToken_ALL_CAPS_-1 | Const(.)
LP Latent	TOK_14 | TOK_36 | TOK_36 | TOK_36

String Transformation: Analysis

Exploration-exploitation trade-off:
Higher latent beam size leads to more diverse programs

Latent Beam Size	Accuracy	Distinct n-Grams			
		n = 1	2	3	4
L = 1	50%	0.13	0.23	0.26	0.28
2	51%	0.13	0.24	0.26	0.28
3	**55%**	0.14	0.25	0.28	0.31
5	54%	0.14	0.26	0.29	0.32
10	54%	**0.14**	**0.26**	**0.30**	**0.33**

String Transformation: Analysis

Performs much better on longer (more complex) programs.

Exploration-exploitation trade-off: Higher latent beam size leads to more diverse programs

Latent Beam Size	Accuracy	Distinct n-Grams			
		n = 1	2	3	4
L = 1	50%	0.13	0.23	0.26	0.28
2	51%	0.13	0.24	0.26	0.28
3	**55%**	0.14	0.25	0.28	0.31
5	54%	0.14	0.26	0.29	0.32
10	54%	**0.14**	**0.26**	**0.30**	**0.33**

Length	RobustFill Acc.	LP Acc.
1	**94.5%**	94.0%
2	83.9%	**84.6%**
3	**72.8%**	72.2%
4	63.1%	**66.1%**
5	47.1%	**49.8%**
6	40.6%	**43.0%**
7	30.2%	**34.6%**
8	22.7%	**28.4%**
9	18.6%	**27.0%**
10	14.4%	**25.6%**

String Transformation: Analysis

Exploration-exploitation trade-off: Higher latent beam size leads to more diverse programs

Latent Beam Size	Accuracy	Distinct n-Grams			
		n = 1	2	3	4
L = 1	50%	0.13	0.23	0.26	0.28
2	51%	0.13	0.24	0.26	0.28
3	**55%**	0.14	0.25	0.28	0.31
5	54%	0.14	0.26	0.29	0.32
10	54%	**0.14**	**0.26**	**0.30**	**0.33**

Performs much better on longer (more complex) programs.

Length	RobustFill Acc.	LP Acc.
1	**94.5%**	94.0%
2	83.9%	**84.6%**
3	**72.8%**	72.2%
4	63.1%	**66.1%**
5	47.1%	**49.8%**
6	40.6%	**43.0%**
7	30.2%	**34.6%**
8	22.7%	**28.4%**
9	18.6%	**27.0%**
10	14.4%	**25.6%**

Tokens often have high-level semantic meaning

	TOK_3	TOK_4	TOK_5	TOK_6	TOK_7	TOK_8	TOK_9
Get First Number	12%	5%	0%	9%	70%	6%	0%
Get Last Number	22%	49%	0%	11%	8%	8%	0%
Get First Word	10%	20%	0%	56%	7%	9%	0%
Get Last Word	75%	4%	0%	6%	9%	6%	0%
Get First Alphanum	11%	3%	0%	35%	42%	9%	0%
Get Last Alphanum	45%	29%	0%	22%	0%	4%	0%

Python Code: Results

Latent Programmer also performs well in generating Python code from docstrings:

Docstring	Program
get an environment variable	`def set_key(key, val, key_prefix=None):` `  return return environ.get(key, key_prefix)`
return a list of the words in the string s	`def split(s, sep=None, maxsplit=-1):` `  return s.split(sep, maxsplit)`
mean squared error function	`def mean_squared_error(y_true, y_pred):` `  return tf.reduce_mean(tf.square((y_true - y_pred)))`

read a python file	`def read_file(fname):`
	`  f = open(fname)`
	`  with open(fname, 'r') as f:`
	`    f.seek(0)`
	`    return f.read()`

Method	BLEU		
	B = 1	10	100
Base (Wei et al., 2019)	10.4	-	-
Dual (Wei et al., 2019)	12.1	-	-
RobustFill [LSTM]	11.4	14.8	16.0
RobustFill [Transformer]	12.1	15.5	17.2
Latent Programmer	**14.0**	**18.6**	**21.3**

Wei et al., 2019

Python Code: Results

Latent Programmer also performs well in generating Python code from docstrings:

Docstring	Program
get an environment variable	`def set_key(key, val, key_prefix=None):` `  return return environ.get(key, key_prefix)`
return a list of the words in the string s	`def split(s, sep=None, maxsplit=-1):` `  return s.split(sep, maxsplit)`
mean squared error function	`def mean_squared_error(y_true, y_pred):` `  return tf.reduce_mean(tf.square((y_true - y_pred)))`

read a python file	`def read_file(fname):`
	`  f = open(fname)`
	`  with open(fname, 'r') as f:`
	`    f.seek(0)`
	`    return f.read()`

Method	BLEU		
	B = 1	10	100
Base (Wei et al., 2019)	10.4	-	-
Dual (Wei et al., 2019)	12.1	-	-
RobustFill [LSTM]	11.4	14.8	16.0
RobustFill [Transformer]	12.1	15.5	17.2
Latent Programmer	**14.0**	**18.6**	**21.3**

Top program tokens (TF-IDF score) for select latent tokens.

0	_files	dirname	glob	isdir	makedir
1	server	_port	_socket	_password	host
2	pip	package	wheel	install	sudo
3	dt	interval	seconds	time	timestamp
4	timeout	_timeout	handle	future	notifier

Conclusion

Propose general two-level search where a high-level plan is generated, then program conditioned on the plan

Latent Programmer:

- Plans are latent codes, or sequences of discrete latent variables
- Latent space is learned in a supervised algorithm using a discrete autoencoder
- Two-level beam search on latent codes, then on program

Thank You!