

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

Figure 7: The DSL for string transformation tasks (Devlin et al., 2017)

A EXTENDED DESCRIPTION OF DSL AND ROBUSTFILL MODEL

The DSL for string transformations we use is the same as used in RobustFill (Devlin et al., 2017), and is shown in Figure 7. The top-level operator for programs in the DSL is a Concat operator that concatenates a random number (up to 10) of expressions e_i . Each expression e can either be a substring expression f , a nesting expression n , or a constant string c . A substring expression can either return the substring between left k_1 and right k_2 indices, or between the i_1 -th occurrence of regex r_1 and i_2 -th occurrence of regex r_2 . The nesting expressions also return substrings of the input, such as extracting the i -th occurrence of a regex, but can also be composed with existing substring or nesting expressions for more complex string transformations.

RobustFill Model RobustFill (Devlin et al., 2017) is a seq-to-seq neural network that uses an encoder-decoder architecture where the encoder computes a representation of the input $e(X)$, and the decoder autoregressively generates the output given the source representation, i.e. conditional likelihood of $Y = [y_1, \dots, y_T]$ decomposes as $p(Y|X) = \prod_{t=1}^T p(y_t|y_{<t}, X)$.

In RobustFill, the probability of decoding each token y_t is given by $p(y_t|y_{<t}, X) = \text{Softmax}(W(h_t))$ with W being the projection onto logits, or unnormalized log probabilities. The hidden representation h_t is an LSTM hidden unit given by,

$$\begin{aligned} E_t &= \text{Attention}(h_{t-1}, e(X)), \\ h_t &= \text{LSTM}(h_{t-1}, E_t). \end{aligned}$$

Here $e(X)$ is the sequence of hidden states after processing the specifications with an LSTM encoder, and Attention(Q, V) denotes the scaled dot-product attention with query Q and key-value sequence V (Bahdanau et al., 2016). In the case of X being multiple I/O examples, the RobustFill model of Devlin et al. (2017) uses double attention

$$\begin{aligned} s_{t,i}^I &= \text{Attention}(h_{t-1}, e(I_i)) \\ s_{t,i}^O &= \text{Attention}(\text{Concat}(h_{t-1}, s_{t,i}^I), e(O_i)) \\ h_{t,i} &= \text{LSTM}(h_{t-1}, \text{Concat}(s_{t,i}^I, s_{t,i}^O)) \quad \forall 1 \leq i \leq N, \end{aligned}$$

and hidden states are pooled across examples before being fed into the final softmax layer, or $h_t = \text{maxpool}_{1 \leq i \leq N} \tanh(V(h_{t,i}))$, where V is another projection.

B EXAMPLES OF GENERATED PROGRAMS AND LATENT CODES

Inputs	Outputs	LP Outputs
"Mason Smith"	"Smith M"	"Smith M"
"Henry Myers"	"Myers H"	"Myers H"
"Barry Underwood"	"Underwood B"	"Underwood B"
"Sandy Jones"	"Jones S"	"Jones S"
LP	GetToken_PROP_CASE_2 ConstStr(" ") GetToken_CHAR_1(GetToken_PROP_CASE_1)	
LP Latent	TOK_30 TOK_13 TOK_39 TOK_30	

Inputs	Outputs	LP Outputs
"January 15"	"jan 15"	"jan 15"
"february 28"	"feb 28"	"feb 28"
"march 1"	"mar 1"	"mar 1"
"October 31"	"oct 31"	"oct 31"
LP	ToCase_LOWER(SubStr(1, 3)) ConstStr("") GetToken_NUMBER_1	
LP Latent	TOK_11 TOK_26 TOK_17	

Inputs	Outputs	LP Outputs
"(321) 704 3331"	"321.704.3331"	"321.704.3331"
"(499) 123 3574"	"499.123.3574"	"499.123.3574"
"(555) 580 8390"	"555.580.8390"	"555.580.8390"
"(288)225 6116"	"288.225.6116"	"288.225.6116"
LP	GetToken_NUMBER_1 ConstStr(.) Replace_" ".(SubStr(-8, -1))	
LP Latent	TOK_17 TOK_27 TOK_24 TOK_16	

Inputs	Outputs	LP Outputs
"Milk 4, Yoghurt 12, Juice 2, Egg 5"	"M.P."	"M.P."
"US:38 China:35 Russia:27 India:1"	"U.I."	"U.I."
"10 Apple 2 Oranges 13 Bananas 40 Pears"	"A.P."	"A.P."
"parul 7 rico 12 wolfram 15 rick 19"	"P.R."	".."
LP	GetToken_CHAR_1(GetToken_PROP_CASE_1) Const(.) GetToken_CHAR_-1(GetAll_ALL_CAPS) Const(.)	
LP Latent	TOK_39 TOK_30 TOK_6 TOK_38 TOK_30	

Figure 8: Latent codes and programs found by Latent Programmer in string transformation tasks. Red denotes I/O where the predicted program mapped input to an incorrect output.

Docstring	Program
get an environment variable	def <code>set_key</code> (key, <code>val</code> , key_prefix=None): return return environ.get(key, key_prefix)
return a list of the words in the string s	def <code>split</code> (s, sep=None, maxsplit=-1): return s.split(sep, maxsplit)
mean squared error function	def <code>mean_squared_error</code> (y_true, y_pred): return tf.reduce_mean(tf.square((y_true - y_pred)))
read a python file	def <code>read_file</code> (fname): <code>f = open(fname)</code> with open(fname, 'r') as f: <code>f.seek(0)</code> return f.read()
pickle dump	def <code>pickle_save</code> (filename, data): with open(filename, 'r') as f: pickle.dump(data, f)
takes a timedelta and returns the total number of seconds	def <code>total_seconds</code> (delta): return ((delta. <code>microseconds</code> + ((delta.days * 24) * 3600) * (10**6))/(10**6))

Figure 9: Programs found by Latent Programmer in Python code generation dataset. Red denotes areas where the predicted program deviates from human code.