A. Extended Description of DSL and RobustFill Model

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

Figure 9. The DSL for string transformation tasks (Devlin et al., 2017)

The DSL for string transformations we use is the same as used in RobustFill (Devlin et al., 2017), and is shown in Figure 9. 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 occurence of regex r_1 and i_2-th occurence 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 a 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 $Y = [y_1, \ldots, y_T]$ 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 $p(y_t|y_{<t}, X) = \text{Softmax}\left(W(h_t)\right)$ with W being the projection onto logits, or unnormalized log probabilities. The hidden representation h_t is an LSTM hidden unit given by,

$$E_t = \text{Attention}\left(h_{t-1}, e(X)\right),$$
$$h_t = \text{LSTM}\left(h_{t-1}, E_t\right).$$

Here $e(X)$ is the sequence of hidden states after processing the specifications with an LSTM encoder, and $\text{Attention}\left(Q, V\right)$ 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

$$s^I_{t,i} = \text{Attention}\left(h_{t-1}, e(I_i)\right)$$
$$s^O_{t,i} = \text{Attention}\left(\text{Concat}\left(h_{t-1}, s^I_{t,i}\right), e(O_i)\right)$$
$$h_{t,i} = \text{LSTM}\left(h_{t-1}, \text{Concat}\left(s^I_{t,i}, s^O_{t,i}\right)\right) \ \forall 1 \le i \le N,$$

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

B. Latent Programmer Architecture

Recall that the LP architecture consists of three modules: a program encoder, latent predictor, and latent program decoder.

Program Encoder The program encoder $\mathrm{ec}(Y)$ is a Transformer encoder, followed by a stack of convolutions of stride 2, each halving the size of the sequence. We apply the convolution ℓ times, which reduces a T-length program to a latent sequence of length $\lceil T/2^\ell \rceil$. This provides temporal abstraction, since the high-level planning actions are made only every 2^ℓ steps. In summary, the program encoder is given by $\mathrm{ec}(Y) \leftarrow h_\ell$, where

$$\begin{aligned} h_0 &\leftarrow \mathrm{TransformerEncoder}(Y) \\ h_m &\leftarrow \mathrm{Conv}(h_{m-1}) \text{ for } m \in 1 \dots \ell \end{aligned} \tag{4}$$

Here $\mathrm{TransformerEncoder}(\cdot)$ applies a stack of self-attention and feed-forward units on input embeddings via a residual path, described in detail by Vaswani et al. (2017). This will be used, along with the latent program decoder, as an autoencoder during training (see Section 3.3).

Latent Predictor The latent predictor $\mathrm{lp}(X)$ is a Transformer that autoregressively outputs probabilities over latent tokens, which can be decoded using search algorithms such as beam search to generate a predicted latent code Z'. This is different than the program encoder, which outputs a single sequence Z, because we use the latent predictor to organize search over latent codes; at test time, we will obtain a L-best list of latent token sequences from $\mathrm{lp}(X)$.

Latent Program Decoder The latent program decoder $d(Z, X)$ is a Transformer that jointly attends to the latent sequence and program specification, to autoregressively generates a distribution over program tokens. This is performed via two separate attention modules, whose outputs are concatenated into the hidden unit. Formally, given a partially generated program $Y' = [y'_1, y'_2, \dots, y'_{t-1}]$, and the encoded specification $E = \mathrm{TransformerEncoder}(X)$, the latent program decoder performs

$$\begin{aligned} e_t &\leftarrow \mathrm{TransformerDecoder}(Y', E)_{t-1} \\ z_t &\leftarrow \mathrm{TransformerDecoder}(Y', Z)_{t-1} \\ h_t &\leftarrow \mathrm{Concat}\left(e_t, z_t\right), \end{aligned} \tag{5}$$

where $\mathrm{TransformerDecoder}(x, y)$ denotes a Transformer decoder applied to outputs y while attending to inputs encoding x, and the subscript indexes an entry in the resulting output sequence. Finally, the distribution over output token k is given by $\mathrm{Softmax}\left(W(h_t)\right)$, where W is a learned parameter matrix. When X is multiple I/O examples, each example is encoded as $E_i = \mathrm{TransformerDecoder}(I_i, O_i)$. Then, a separate hidden state per I/O is computed following equation 5, followed by a late max-pool to get the final hidden state.

C. Interpretability Experiments on Toy DSL

$$
\begin{array}{rcl}
\text{Program } Y & := & \texttt{Concat}(e_1, e_2, \ldots) \\
\text{Expression } e & := & \texttt{GetSpan}(r_1, i_1, r_2, i_2) \\
\text{Regex } r & := & t_1 \mid \ldots \mid t_n \mid \delta_1 \mid \ldots \mid \delta_m \\
\text{Type } t & := & \texttt{NUMBER} \mid \texttt{WORD} \mid \texttt{ALPHANUM} \\
\text{Index } i & := & -1 \mid 1 \mid 2 \\
\text{Delimiter } \delta & := & \&\,, . \\
\text{Character } c & := & A-Z \mid a-z \mid 0-9 \mid \&\,, .
\end{array}
$$

Figure 10. Toy DSL for string transformation tasks

Inputs	Outputs
“,C,XoC"	“C"
“.G73,NT"	“G73"
“.Uvg t7MXI"	“Uvg"
“.tLqFJ .dMKlh"	“tLqFJ"

LP	`GetSpan_ALPHANUM_1_ALPHANUM_1`
LP Latent	`TOK_6`

Inputs	Outputs
“,3okM5,„"	“3okM5 ,3okM53okM5 ,3okM5"
“„ ,.O8p"	“O8p ,.O8pO8p ,.O8p"
“„ , ,IBpU"	“IBpU ,IBpUIBpU ,IBpU"
“„„mUV"	“mUV „„mUVmUV „„mUV"

LP	`GetSpan_ALPHANUM_1_ALPHANUM_1 | GetSpan_,_-1_ALPHANUM_1 | GetSpan_ALPHANUM_1_ALPHANUM_-1 | GetSpan_,_2_ALPHANUM_1`
LP Latent	`TOK_6 | TOK_5 | TOK_6 | TOK_5`

Inputs	Outputs
“,CNBA,uJke.00 Hm 6938"	“CNBA,uJke.00CNBA,6938"
“.Xp.sYH ,46,Rj ,330"	“Xp.sYH ,46Xp.sYH ,46,Rj ,330"
“,gYR 85296 LRgJX,15,eWEeu"	“gYR 85296gYR 85296 LRgJX,15,15"
“.BPYVr ALVbf wEvm 86,103"	“BPYVr ALVbf wEvm 86BPYVr ALVbf wEvm 86,103"

LP	`GetSpan_WORD_1_NUMBER_1 | GetSpan_WORD_1_,_-1 | GetSpan_NUMBER_2_NUMBER_2`
LP Latent	`TOK_9 | TOK_9 | TOK_4`

Figure 11. Latent codes and programs found by Latent Programmer in toy DSL.

Inputs	Outputs
“r, 6150,XLQPl"	“6150r, 6150r, 6150"
“.ERYlM, 80,Iejg"	“80ERYlM, 80ERYlM, 80"
“sqd,.xJx,01928"	“01928sqd,.xJx,01928sqd,.xJx,01928"
“.w Nqk.42,"	“42w Nqk.42w Nqk.42"

LP	`GetSpan_NUMBER_-1_NUMBER_-1 | GetSpan_WORD_1_NUMBER_1 | GetSpan_WORD_1_,_-1`
LP Latent	`TOK_4 | TOK_9 | TOK_9`

Inputs	Outputs
“.VyPL 3785.0933,Xj EFSjp"	“VyPL37853785.0933,Xj EFSjp"
“.023 Jz Suz.t .4"	“Jz023023 Jz Suz.t"
“TyCBs,803 TjtA,4 .qH"	“TyCBs803803 TjtA,4 .qH"
“.cCr,3248 L ,QPLd.6472"	“cCr32483248 L ,QPLd"

LP	`GetSpan_WORD_1_WORD_1 | GetSpan_NUMBER_1_NUMBER_1 | GetSpan_NUMBER_1_WORD_-1`
LP Latent	`TOK_6 | TOK_7 | TOK_9`

Inputs	Outputs
“.Eu.F IgKFs,XD.011"	“011F"
“.U0Z,aVEzk,KNq 08,UqlhR"	“0Z"
“44 j.Oz.peQy,l"	“44Oz"
“,FtAz CIHLB V 851.oR8l"	“851CIHLB"

LP	`GetSpan_NUMBER_1_NUMBER_1 | GetSpan_WORD_2_WORD_2`
LP Latent	`TOK_7 | TOK_3`

Inputs	Outputs
“.9312 ..767"	“7679312 ..76793129312"
“.,04194,47460"	“4746004194,474600419404194"
“.4940..3646"	“36464940..364649404940"
“. .180,5275"	“5275180,5275180180"

LP	`GetSpan_NUMBER_2_NUMBER_-1 | GetSpan_NUMBER_1_NUMBER_-1 | GetSpan_NUMBER_1_NUMBER_1 | GetSpan_NUMBER_1_NUMBER_1`
LP Latent	`TOK_4 | TOK_9 | TOK_7 | TOK_7`

Figure 12. More latent codes and programs found by Latent Programmer in toy DSL.

D. 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"
“febuary 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.E."	“M.E."
“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 13. 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 set_key(key, val, key_prefix=None):` `  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()`

pickle dump	`def pickle_save(filename, data):`
	`  with open(filename, 'r') as f:`
	`  pickle.dump(data, f)`

takes a timedelta and returns the total number of seconds	`def total_seconds(delta):` `  return ((delta.microseconds + ((delta.days * 24) * 3600) * (10**6))/(10**6))`

Figure 14. Programs found by Latent Programmer in Python code generation dataset. Red denotes ares where the predicted program deviates from human code.