A. Search Algorithms

In the following, we describe the algorithm that we use to calculate child model fitness with hurdles (Algorithm 1) and evolution architecture search with Progressive Dynamic Hurdles (Algorithm 2).

Algorithm 1 Calculate Model Fitness with Hurdles

```
inputs:
  $model$: the child model
  $s$: vector of train step increments
  $h$: queue of hurdles

append $\infty$ to $h$
TRAIN_N_STEPS($model$, $s_0$)
$fitness \leftarrow$ EVALUATE($model$)
$i \leftarrow 0$
$hurdle \leftarrow h_i$

while $fitness > hurdle$ do
  $i \leftarrow i + 1$
  TRAIN_N_STEPS($model$, $s_i$)
  $fitness \leftarrow$ EVALUATE($model$)
  $hurdle \leftarrow h_i$
end while
return $fitness$
```

Algorithm 1 Calculating fitness with hurdles takes as arguments a child model, a vector of train step increments (s) and a queue of hurdles(h). The child model is the candidate model in our neural architecture search. The vector of step increments describes the number of steps between each hurdle; its length must be greater than 0. The queue of hurdles describes what hurdles have already been established; its length must be in [0, $length(s)$).

The algorithm starts by first training the child model a fixed number of s_0 steps and evaluating on the validation set to produce a fitness, as is done in Real et al. (2019). After this baseline fitness is established, the hurdles (h) are compared against to determine if training should continue. Each h_i denotes the fitness a child model must have after $\sum_{j=0}^{i} s_j$ train steps to continue training. Each time a hurdle h_i is passed, the model is trained an additional s_{i+1} steps. If the model's fitness ever falls below the hurdle corresponding to the number of steps it was trained for, training ends immediately and the current fitness is returned. If the model never falls below a hurdle and all hurdles have been passed, the child model receives one final training of $s_{length(h)}$ steps before fitness is returned; this is expressed in Algorithm 1 with ∞ being appended to the end of h.

Algorithm 2 Evolution Architecture Search with PDH

```
inputs:
  $s$: vector of train step increments
  $m$: number of child models per hurdle

$h \leftarrow$ empty queue
$i \leftarrow 0$
$population \leftarrow$ INITIAL_POPULATION()

while $i <$ LENGTH($s$) - 1 do
  $population \leftarrow$ EVOL_N_MODELS($population$,
                                      $m$, $s$, $h$)
  $hurdle \leftarrow$ MEAN_FITNESS_OF_MAX($population$)
  append $hurdle$ to $h$
end while

$population \leftarrow$ EVOL_N_MODELS($population$,
                                    $m$, $s$, $h$)
return $population$
```

Algorithm 2 Evolution architecture search with PDH takes as arguments a vector of train step increments (s) and a number of child models per hurdle (m). It begins as Real et al.'s (2019) evolution architecture search with a fixed number of child model train steps, s_0. However, after m child models have been produced, a hurdle is created by taking the mean fitness of the current population and it is added to the hurdle queue, h. Algorithm 1 is used to compute each child model's fitness and so if they overcome the new hurdle they will receive more train steps. This process is continued, with new hurdles being created using the mean fitness of all models that have trained the maximum number of steps and h growing accordingly. The process terminates when $length(s) - 1$ hurdles have been created and evolution is run for one last round of m models, using all created hurdles.

B. Search Space Information

In our search space, a child model's genetic encoding is expressed as: [*left input*, *left normalization*, *left layer*, *left relative output dimension*, *left activation*, *right input*, *right normalization*, *right layer*, *right relative output dimension*, *right activation*, *combiner function*] $\times$ 14 + [*number of cells*] $\times$ 2, with the first 6 blocks allocated to the encoder and the latter 8 allocated to the decoder. In the following, we will describe each of the components.

Input. The first branch-level search field is *input*. This specifies what hidden state in the cell will be fed as input to the branch. For each i^{th} block, the input vocabulary of its branches is $[0, i)$, where the j^{th} hidden state corresponds to the j^{th} block output and the 0^{th} hidden state is the cell input.

Normalization. The second branch-level search field is *normalization*, which is applied to each input before the layer transformation is applied. The normalization vocabulary is [LAYER NORMALIZATION (Ba et al., 2016), NONE].

Layers. The third branch-level search field is *layer*, which is the neural network layer applied after the normalization. It's vocabulary is:

- STANDARD CONV wx1: for $w \in \{1, 3\}$
- DEPTHWISE SEPARABLE CONV wx1: for $w \in \{3, 5, 7, 9, 11\}$
- LIGHTWEIGHT CONV wx1 r: for $w \in \{3, 5, 7, 15\}$, $r \in \{1, 4, 16\}$ (Wu et al., 2019). r is the reduction factor, equivalent to d/H described in Wu et al. (2019).
- h HEAD ATTENTION: for $h \in \{4, 8, 16\}$
- GATED LINEAR UNIT(Dauphin et al., 2017)
- ATTEND TO ENCODER: (Only available to decoder)
- IDENTITY: No transformation applied to input
- DEAD BRANCH: No output

For decoder convolution layers the inputs are shifted by $(w - 1)/2$ so that positions cannot "see" later predictions.

Relative Output Dimension. The fourth branch-level search field is *relative output dimension*, which describes the output dimension of the corresponding layer. The Transformer is composed mostly of layers that project to the original input embedding depth (512 in the "base" configuration), but also contains 1x1 convolutions that project up to a dimension of 4 times that depth (2048 in the "base" configuration). The relative output dimension search field accounts for this variable output depth. It's vocabulary consists of 10 relative output size options: $[1, 10]$.

Here "relative" refers to the fact that for every layer i and j, each of their absolute output dimensions, a, and relative output dimensions, d, will obey the ratio: $a_i/a_j = d_i/d_j$. We determine the absolute output dimensions for each model by finding a scaling factor, s, such that for every layer i, $a_i = d_i * s$ and the resulting model has an appropriate number of parameters; at the end of this section, we describe our constraints on number of model parameters. There may be multiple values of s for any one model that satisfy this constraint, and so for our experiments we simply perform a binary search and use the first valid value found. If no valid value is found, we reject the child model encoding as invalid and produce a new one in its stead.

We chose a vocabulary of relative sizes instead of absolute sizes because we only allow models within a fixed parameter range, as described later in this section (Constraints). Using relative sizes allows us to increase the number of configurations that represent valid models in our search space, because we can dynamically shrink or grow a model to make it fit within the parameter bounds. We found that using absolute values, such as $[256, 512, 1024, 2048]$, increases the number of rejected models and thereby decreases the possible models that can be expressed.

This relative output dimensions field is ignored for both the IDENTITY and DEAD BRANCH layers.

Activations. The final branch-level search field is *activation*, which is the non-linearity applied on each branch after the neural network layer. The activation vocabulary is {SWISH (Ramachandran et al., 2017; Elfwing et al., 2018), RELU, LEAKY_RELU (MAAS ET AL., 2013), NONE}.

Combiner Functions. The block-level search field, *combiner function*, describes how the left and right layer branches are combined together. Its vocabulary is {ADDITION, CONCATENATION, MULTIPLICATION}. For MULTIPLICATION and ADDITION, if the right and left branch outputs have differing embedding depths, then the smaller of the two is padded so that the dimensionality matches. For ADDITION the padding is 0's; for MULTIPLICATION the padding is 1's.

Number of Cells. The cell-level search field is *number of cells* and it describes the number of times the cell is repeated. Its vocabulary is [1,6].

Composition. Each child model is defined by two cells, one for the encoder and one for the decoder. The encoder cell contains 6 blocks and the decoder cell contains 8 blocks. Each block contains two branches, each of which takes a previous hidden layer as input, and then applies its normalization, layer (with specified relative output dimensions) and activation to it. The two branches are then joined with the combiner function. Any unused hidden states are automatically added to the final block output via addition. Both the encoder and decoder cells defined this way are repeated their corresponding *number of cells* times and connected to the input and output embedding portions of the network to produce the final model; we use the same embedding scheme described by Vaswani et al. (2017) for all models. See Figure 1 for a depiction of this composition.

Constraints. In the interest of having a fair comparison across child models, we limit our search to only architectures configurations that can contain between 59.1 million and 64.1 million parameters when their relative output dimensions are scaled; in the Tensor2Tensor (Vaswani et al., 2018) implementation we use, the base Transformer has roughly 61.1 million parameters on WMT En-De, so our models are allowed 3 million less or more parameters than that. Models that cannot be represented within this parameter range are not included in our search space.

Additionally, in preliminary experiment runs testing the effectiveness of our search space, we discovered three trends that hurt performance in almost every case. Firstly and most obviously is when a proposed decoder contains no ATTEND TO ENCODER layers. This results in the decoder receiving no signal from the encoder and thus the model output will not be conditioned on the input. Therefore, any model that does not contain ATTEND TO ENCODER is not in our search space. The second trend that we noticed was that models that had layer normalization removed were largely worse than their parent models. For this reason, we remove NONE from the *normalization* mutation vocabulary for each experiment, unless otherwise specified. Lastly, we observed that an important feature of good models was containing an unbroken residual path from inputs to outputs; in our search space, this means a path of IDENTITY layers from cell input to output that are combined with ADDITION at every *combination function* along the way. Our final constraint is therefore that models that do not have unbroken residual paths from cell inputs to outputs are not in our search space.

C. Ablation Study of the Evolved Transformer

To understand what mutations contributed to ET's improved performance we conducted two rounds of ablation testing. In the first round, we began with the Transformer and applied each mutation to it individually to measure the performance change each mutation introduces in isolation. In the second round, we began with ET and removed each mutation individually to again measure the impact of each single mutation. In both cases, each model was trained 3 times on WMT En-De for 300k steps with identical hyperparameters, using the inverse-square-root decay to 0 that the Transformer prefers. Each training was conducted on a single TPU V.2 chip. The results of these experiments are presented in Table 5; we use validation perplexity for comparison because it was our fitness metric.

In all cases, the augmented ET models outperformed the the augmented Transformer models, indicating that the gap in performance between ET and the Transformer cannot be attributed to any single mutation. The mutation with the seemingly strongest individual impact is the increase from 3 to 4 decoder cells. However, even when this mutation is introduced to the Transformer and removed from ET, the resulting augmented ET model still has a higher fitness than the augmented Transformer model.

To highlight the impact of each augmented model's mutation, we present not only their perplexities but also the difference between their mean perplexity and their unaugmented base model mean perplexity in the "Mean Diff" columns:

base model mean perplexity - augmented mean perplexity

This delta estimates the change in performance each mutation creates in isolation. Red highlighted cells contain evidence that their corresponding mutation hurt overall performance. Green highlighted cells contain evidence that their corresponding mutation helped overall performance.

In half of the cases, both the augmented Transformer's and the augmented Evolved Transformer's performances indicate that the mutation was helpful. Changing the number of attention heads from 8 to 16 was doubly indicated to be neutral and changing from 8 head self attention to a GLU layer in the decoder was doubly indicated to have hurt performance. However, this and other mutations that seemingly hurt performance may have been necessary given how we formulate the problem: finding an improved model with a comparable number of parameters to the Transformer. For example, when the Transformer decoder cell is repeated 4 times, the resulting model has 69.6M parameters, which is outside of our allowed parameter range. Thus, mutations that shrank ET's total number of parameters, even at a slight degradation of performance, were necessary so that other more impactful parameter-expensive mutations, such as adding an additional decoder cell, could be used.

Other mutations have inconsistent evidence about how useful they are. This ablation study serves only to approximate what is useful, but how effective a mutation is also depends on the model it is being introduced to and how it interacts with other encoding field values.

Mutation Field	Mutation Block Index	Mutation Branch	Transformer Value	ET Value	Transformer Perplexity	ET Perplexity	Transformer Mean Diff	ET Mean Diff
DECODER ACTIVATION	6	LEFT	RELU	SWISH	4.73 ± 0.01	4.51 ± 0.02	-0.02	0.04
DECODER ACTIVATION	2	RIGHT	RELU	NONE	4.73 ± 0.01	4.48 ± 0.00	-0.02	0.02
DECODER INPUT	1	LEFT	1	0	4.74 ± 0.04	4.46 ± 0.00	-0.01	-0.01
DECODER LAYER	0	LEFT	8 HEAD ATTENTION	16 HEAD ATTENTION	4.75 ± 0.01	4.47 ± 0.01	0.0	0.0
DECODER LAYER	2	LEFT	STANDARD CONV 1X1	SEPARABLE CONV 11X1	4.67 ± 0.01	4.55 ± 0.00	-0.08	0.09
DECODER LAYER	3	LEFT	STANDARD CONV 1X1	SEPARABLE CONV 7X1	4.72 ± 0.01	4.46 ± 0.01	-0.03	0.0
DECODER LAYER	2	RIGHT	DEAD BRANCH	SEPARABLE CONV 7X1	4.71 ± 0.02	4.47 ± 0.00	-0.04	0.01
DECODER NORM	3	LEFT	NONE	LAYER NORM	4.73 ± 0.00	4.45 ± 0.01	-0.02	-0.01
DECODER NORM	7	LEFT	NONE	LAYER NORM	4.73 ± 0.02	4.47 ± 0.02	-0.02	0.01
DECODER OUTPUT DIM	2	LEFT	8	4	4.74 ± 0.01	4.45 ± 0.01	-0.01	-0.02
DECODER NUM CELLS	-	-	3	4	4.62 ± 0.00	4.59 ± 0.01	-0.13	0.12
ENCODER LAYERS	0	LEFT	8 HEAD ATTENTION	GATED LINEAR UNIT	4.80 ± 0.03	4.45 ± 0.02	0.05	-0.01
ENCODER LAYERS	2	LEFT	STANDARD CONV 1X1	SEPARABLE CONV 9X1	4.69 ± 0.01	4.50 ± 0.00	-0.06	0.04
ENCODER LAYERS	1	RIGHT	DEAD BRANCH	STANDARD CONV 3X1	4.73 ± 0.01	4.47 ± 0.03	-0.02	0.01
ENCODER NORMS	2	LEFT	NONE	LAYER NORM	4.79 ± 0.03	4.46 ± 0.02	0.04	0.0
ENCODER OUTPUT DIM	2	LEFT	2	1	4.74 ± 0.01	4.45 ± 0.0	-0.01	-0.01

Table 5. **Mutation Ablations**: Each mutation is described by the first 5 columns. The augmented Transformer and augmented ET perplexities on the WMT'14 En-De validation set are given in columns 6 and 7. Columns 7 and 8 show the difference between the unaugmented base model perplexity mean and the augmented model perplexity mean. Red highlighted cells indicate evidence that the corresponding mutation hurts overall performance. Green highlighted cells indicate evidence that the corresponding mutation helps overall performance.