MULTIVERSE: Mining Collective Data Science Knowledge from Code on the Web to Suggest Alternative Analysis Approaches

Mike A. Merrill
mikeam@cs.washington.edu
University of Washington
Seattle, Washington, USA

Ge Zhang
zhangge9194@pku.edu.cn
Peking University
Beijing, China

Tim Althoff
althoff@cs.washington.edu
University of Washington
Seattle, Washington, USA

ABSTRACT

Data analyses are based on a series of *"decision points"* including data filtering, feature operationalization and selection, model specification, and parametric assumptions. *"Multiverse Analysis"* research has shown that a lack of exploration of these decisions can lead to non-robust conclusions based on highly sensitive decision points. Importantly, even if myopic analyses are *technically* correct, analysts' focus on one set of decision points precludes them from exploring alternate formulations that may produce very different results. Prior work has also shown that analysts' exploration is often limited based on their training, domain, and personal experience. However, supporting analysts in exploring alternative approaches is challenging and typically requires expert feedback that is costly and hard to scale.

Here, we formulate the tasks of identifying decision points and suggesting alternative analysis approaches as a classification task and a sequence-to-sequence prediction task, respectively. We leverage public collective data analysis knowledge in the form of code submissions to the popular data science platform Kaggle to build the first predictive model which supports Multiverse Analysis. Specifically, we mine this code repository for 70k small differences between 40k submissions, and demonstrate that these differences often highlight key decision points and alternative approaches in their respective analyses. We leverage information on relationships within libraries through neural graph representation learning in a multitask learning framework. We demonstrate that our model, MULTIVERSE, is able to correctly predict decision points with up to 0.81 ROC AUC, and alternative code snippets with up to 50.3% GLEU, and that it performs favorably compared to a suite of baselines and ablations. We show that when our model has perfect information about the location of decision points, say provided by the analyst, its performance increases significantly from 50.3% to 73.4% GLEU. Finally, we show through a human evaluation that real data analysts find alternatives provided by MULTIVERSE to be more reasonable, acceptable, and syntactically correct than alternatives from comparable baselines, including other transformer-based seq2seq models.

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467455

CCS CONCEPTS

• **Applied computing**; • **Information systems** → *Web mining*; • **Computing methodologies** → *Neural networks*;

KEYWORDS

Robust Data Science, Metascience, Multiverse analysis, Garden of Forking Paths, seq2seq, Code Representation Learning

ACM Reference Format:
Mike A. Merrill, Ge Zhang, and Tim Althoff. 2021. MULTIVERSE: Mining Collective Data Science Knowledge from Code on the Web to Suggest Alternative Analysis Approaches. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467455

1 INTRODUCTION

A recent study asked twenty-nine teams of well-trained data analysts a narrowly phrased question: do referees give penalties to dark-skinned soccer players at a higher rate than light-skinned players [8]? Surprisingly, while individual teams arrived at highly confident conclusions, there was no overall consensus among these teams. Crucially, despite access to identical datasets and no appreciable technical errors, teams arrived at their conclusions through unique analyses which varied in their independent and dependent variables, model selection, statistical assumptions, and more.

Multiverse Analysis is an emerging concept in statistics and metascience which attempts to describe this phenomenon by demonstrating that analysts must navigate a series of *"decision points"* in order to draw conclusions from data [10, 36]. While each of the choices made at a decision point (such as setting a threshold or specifying a model) may be entirely reasonable and defensible, so may many of its alternatives. The resulting set of options produces a *"garden of forking paths"* from which analysts traditionally select only a single path from raw data to results [10].

A growing body of work has identified the large multiverse of possible analyses as a significant contributor to the reproducibility crisis, since even minor changes to the analytical path can often materially alter subsequent results [8, 35]. Accordingly, in recent years there has been an increased interest in building tools to help data analysts explore alternative analyses through Multiverse Analysis, in order to better understand the robustness of reported outcomes along each path [22, 29].

However, it is very challenging to come up with potential alternative analysis approaches. Previous studies show that analysts' decisions are limited by their methodological experience, education, or domain [28], and that computational tools could help these analysts explore the multiverse more effectively [29]. Critically,

		Code Snippet	Notes
(a)	Original Code	`model = create_my_model(optimizer='rmsprop')`	Our model identifies and augments keyword arguments that could belong to decision points, (such as `optimizer`), but preserves user-defined non-decision point subsequences like `create_my_model`.
	Predicted Alternative	`model = create_my_model(optimizer='adam')`	
(b)	Original Code	`model = DecisionTreeRegressor()` `y_hat = model.fit(X_train,y_train).predict(X_train)`	Our model correctly identifies models that are suitable alternatives for eachother, like `DecisionTreeRegressor` and `RandomForestRegressor`
	Predicted Alternative	`model = RandomForestRegressor()` `y_hat = model.fit(X_train,y_train).predict(X_T)`	

(c)	Original Code	`threshold = 90 #Per definitions`
		`reviews['inspection_result'] = reviews['inspection_score']\`
		`.apply(lambda x: 1 if x>= threshold else 0)`
			Our model finds and provides alternatives for sensitive thresholds whose manipulation could potentially alter downstream analysis

	Predicted Alternative	`threshold = 60`
		`reviews['inspection_result'] = reviews['inspection_score']\`
		`.apply(lambda x: 1 if x>= threshold else 0)`	

Figure 1: Examples of original code snippets from Kaggle and MULTIVERSE's generated alternative, with decision points in the original (x^{DP}) in green and corresponding inserted alternatives ($\hat{y}^{DP}$) in red. MULTIVERSE correctly identifies decision points in input snippets, and copies non-decision point tokens to the output. The model can also provide alternative models, alternative hyperparameters, and identifies and augments thresholds that may impact downstream tasks.

these tools currently expect analysts to identify decision points and generate alternative analyses themselves, which requires expertise or expert feedback that is prohibitively expensive to scale [42].

In this paper, we formalize Multiverse Analysis a composition of two tasks, *Decision Point Classification* (identifying key decision points in code) and *Alternative Generation* (given a decision point, formulating code snippets which provide an alternate analysis) (Section 3). While this formalization permits us to programmatically support Multiverse Analysis, as of yet no large dataset of Multiverse Analyses exists with which to train such a model.

Here we turn to the collective expertise of analysts on the web in the form of popular data science competitions, where thousands of analysts work on the same task. A central idea of this paper is that small differences between submissions to each of these competitions contain potential analysis alternatives, allowing us to create a corpus of 70k alternatives from 40k submissions without additional costly expert supervision (Section 4.1). For example, an analyst might change model definitions or hyperparameters between submissions to explore their impact on the analysis outcome.

A second idea is that code libraries tend to be intentionally designed to represent semantic relationships between objects and functions commonly used in data analysis. For example, `sklearn.clustering.KMeans` and `sklearn.clustering.DBSCAN` are defined in the same submodule (`clustering`) and could be reasonable alternatives to one another. We build on this idea by extracting a large graph (called a *"library graph"*) of these relationships to learn suitable code representations (Section 4.2).

Leveraging this dataset representing collective data analysis expertise, we propose a novel neural architecture that is able to (1) detect decision points in analysis code (Section 5.3.1), and (2) generate potential alternatives for the analyst to consider (Section 5.3.2). We additionally (3) integrate structural library information using a graph neural network approach that informs the learned code representations (Section 5.3.3). Further, we propose a formulation of beam search, called *"Span-Aware Beam Search"*, that limits the generation of new code to decision points, and leaves surrounding code unchanged, improving performance in *Alternative Generation*. Figure 1 shows examples of MULTIVERSE's predictions, which correctly suggest alternative models, parameters, and thresholds.

We evaluate MULTIVERSE's performance on the tasks defined in Section 3. We show that MULTIVERSE achieves up to 0.81 ROC AUC on locating decision points in the *"Decision Point Classification"* task, and 88.7% ROUGE-L-F1 on the *"Alternative Generation"* task (Section 6). We conduct an ablation study to show that MULTIVERSE's library graph , Span-Aware Beam Search, and multitask formulation all strictly improve performance on Decision Point Classification ROC AUC and Alternative Generation ROUGE-L. We also evaluate our model against comparable seq2seq models for code, showing that it performs two to three times better than those models on Alternative Generation (e.g. from 29.2% ROUGE-L Precision to 93.3%). To simulate the setting where an analyst has already identified their decision points, we pass information about the location of decision points to MULTIVERSE and show that under this condition performance improves from 50.3% to 73.4% GLEU. Finally, we show through a human evaluation that real data analysts find MULTIVERSE's alternatives to be more reasonable, acceptable, and syntactically correct than alternatives from comparable baselines, including other transformer-based seq2seq models.

Our work shows the feasibility of learning to recommend alternative analyses by mining collective data science knowledge from the web and has implications for improving reproducibility by supporting Multiverse Analysis. We make all code and data used in this paper available at github.com/behavioral-data/multiverse to encourage future research and tool development for Multiverse Analysis.

2 RELATED WORK

Reproducibility & The Data Analysis Multiverse is similar but distinct from metalearning in that it attempts to describe *all* reasonable alternatives, instead of trying to discover the single *"best"* one. Prior work on the Data Analysis Multiverse has shown that

analysts' decisions can limit reproducibility in individual studies [10], are frequently limited by experience and social pressure (e.g. pressure from a field to perform a "standard" analysis plan rather than another) [28], and can produce drastically different results on the same task [8]. Researchers have developed tools to help analysts visualize alternative pathways [29, 35], log versions of their own work [20, 21], and detect false positives [24]. Prior interviews have indicated that even seasoned data analysts struggle to develop multiverse alternatives [28]. Therefore, we build on this work and propose a potential solution that supports analysts by developing the first model which can automatically suggest multiverse decision points and alternatives.

Seq2seq for Code. Sequence to sequence (seq2seq) models take a sequence (such as natural language or code) as input, and generate a corresponding sequence as output. These models have achieved significant results on complex tasks like machine translation [1, 39], speech recognition [33] and search [16]. One area focuses on learning to represent edits, often involving common fixes for grammar, clarity and style [5, 9, 26].

Deep learning has recently become a powerful tool to apply seq2seq tasks to code. Typical seq2seq tasks for code include bug fixes [4], code transformation [38], bug localization [23], API usage generation [12], etc. SequenceR [4] uses an LSTM encoder-decoder model with attention and copy mechanisms to generate simple one line fixes for bugs. Neural Code Translator [38] uses an encoder-decoder recurrent nerual network (RNN) to learn code changes before and after pull requests on Github. Lam et al. [23] use a revised Vector Space Model for bug localization, creating a representation to relate terms in a bug report to source code tokens. In contrast to these methods that separate the tasks of token classification and code generation, we propose a joint learning method for both Decision Point Classification and Alternative Generation and demonstrate that this multitask framework leads to improved performance (Section 6). Furthermore, by using a BPE Tokenizer and a transformer architecture, we model an open vocabulary and support longer sequences than these alternatives.

Graph Representation Learning. Graph representation learning is a field that aims to embed nodes, edges [11], sub-graphs [7], and full graphs [30, 41] in a high dimensional vector space that captures desirable properties of the original graph, such as node distance, hyperbolicity, or local neighborhoods. In this paper, we incorporate information from external code libraries by jointly learning representations of library structure and tokens from code in order to improve performance (Section 5.3.3). Of particular interest to this paper is distortion, which describes the degree to which the pairwise distance between node embeddings reflects their distance in the original graph [2, 3].

3 MULTIVERSE ANALYSIS TASKS

When a data analyst explores alternative formulations of their analysis they must identify likely decision points in their code and then construct a set of reasonable alternatives for each decision point. These distinct functions could be performed jointly (as in a multitask framework), or sequentially (first identifying decision points and only then providing alternatives). Furthermore, a user may have some strong prior belief about the location of decision points, or they could require substantial guidance to detect them (*e.g.,* due to a lack of experience with alternative approaches).

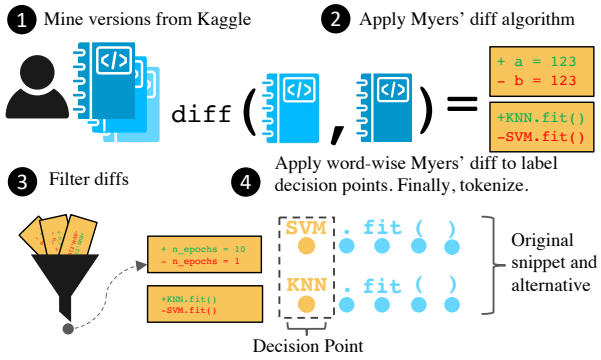

Figure 2: Our process for mining decision points from Kaggle notebooks (Section 4.1)

In this paper we propose to formulate the goal of providing alternative analyses as two distinct prediction tasks: *Decision Point Classification* and *Alternative Generation*. In doing so, we are able to explore not only our model's ability to provide useful alternatives, but also unlock two other avenues for analysis. First, in Section 6 we show that models that solve these tasks independently or treat our goal as a pure seq2seq task perform worse than a multitask model which jointly takes both objectives into account. Second, our multitask framework allows us to bound the model's performance under the condition that it has additional information about the location of the decision points (Section 6.3). This experimental setting mirrors the common paradigm where an analyst has some a priori intuition about which parts of their code they want to *"multiverse"*, that is, for which parts they would like to see alternatives.

3.1 Decision Point Classification

Creating an alternative analysis frequently entails making small edits to existing code, such as changing a hyper-parameter or substituting one statistical model for another. This identification process is often non-trivial, even for human analysts, as knowing which edits to code are likely to materially alter results requires a deep understanding of the underlying methods [28]. We are therefore interested in classifying spans of code which are likely to belong to decision points, and whose modification therefore alters the final result. More formally, given a token vocabulary V, an input code sequence $x = \{x_t \in V\}, t = 0..N$, and a code alternative $y = \{y_t \in V\}, t = 0..N$, we would like to classify the set of subsequences in x that do not appear in y: x^{DP} (these are the subsequences that changed and may constitute a decision point and corresponding alternative). We also define y^{DP} as the analogous set of subsequences that do not appear in x.

3.2 Alternative Generation

When a data analyst creates an alternative analysis, they must not only locate decision points, but also write code to formulate their alternative approach. Like decision point classification, alternative generation is difficult even for seasoned analysts because it requires

a broad knowledge of related statistical methods and creativity to imagine different formulations of the same strategy. We model this process as a sequence to sequence (seq2seq) prediction task, whereby an alternative code snippet can be thought of as a machine translation of a corresponding snippet from the original analysis. More formally, we train our model to produce a code alternative $y = \{y_t \in V\}, t = 0..N$. Our goal in this machine translation task is find some $\hat{y}$ that maximizes $P(\hat{y}|x) = \prod_{t=0}^{t=N} P(\hat{y}_t|x_0...x_N, \hat{y}_0...\hat{y}_{t-1})$.

4 KEY IDEAS

Next we describe two key ideas of this paper. First, we motivate and support our hypothesis that a carefully filtered set of Kaggle submissions represent alternative analyses. Second, we outline how the graphical structure of code libraries can be mined and incorporated into a model to help it learn relationships between relevant semantic entities in code.

4.1 Some Small Differences between Kaggle Submissions Constitute Alternative Analysis Approaches

On the popular data science competition website kaggle.com users compete against each other to answer questions with data, often for cash prizes. While Kaggle is best known for its machine learning competitions, users also clean, visualize, explore, and test data through hosted Jupyter Notebooks. Critically, Kaggle employs a version control system to publicly host all prior versions of public scripts. A key insight of this paper is that when a user submits multiple versions of their analysis using this feature, non-trivial edits between submissions can be considered as attempts to *"multiverse"* their approach. We do not claim that these edits represent *"better"* or more *"correct"* analyses, rather we argue that they represent code snippets with a common goal set by a user's intention. As a user updates their submissions with new data cleaning methods, models, and evaluation metrics they are in effect expressing an alternate formulation of their own analysis. These updates are very common as analysts compete with each other to complete these tasks. Furthermore, since the site contains multiple users' submissions to the same problems, these submissions may share a common context. Taken as a whole, these scripts represent the efforts of tens of thousands of users to solve our tasks of Decision Point Classification and Alternative Generation (Section 3). We detail our process in for mining alternatives in Figure 2.

Method. We crawled all versions of all public submissions to all competitions on Kaggle, yielding 48k submissions with an average of 9.3 versions of each script, or 450k scripts in total. Since Jupyter has more than 8 million users and is the most popular IDE among data analysts, we focus on Python notebook cells as our unit of analysis [17, 19]. However, the entirety the method presented in this paper could be applied to any unit of analysis (e.g. function declarations, individual lines), and any other language that allows imports from external libraries (e.g. Julia, R, Go).

Processing and Data Filtering. We then used the Myers' diff algorithm (which is commonly used to compute git merges) to find edits between sequential versions of submissions [31]. In order to help our model interpret context, we include a line of unchanged code above and below each diff. A manual inspection of the dataset showed that a portion of these diffs were trivial edits such as changing a model's checkpoint directory or reformatting code for readability. To focus our method on the types of edits that most reasonably constitute a multiverse analysis, we filtered out diffs whose edits were changes to whitespace, changes to I/O operations (e.g. `pd.read_csv("v1.csv")` → `pd.read_json("v2.json")`), diffs that simply rearranged code without changing its semantics (e.g. `a + b` → `b + a`), diffs that appeared to handle plotting (e.g. `fig.set_size(10,5)` → `fig.set_size(10,10)`), and diffs that simply renamed variables (e.g. `clf = KNN()` → `model = KNN()`). Initial explorations showed that most decision points and alternatives lead to changes of individual functions, function arguments, and typically involve very few lines of code. Therefore, we remove all diffs whose total size (including context) is more than five lines, leaving us with a final dataset of 70k pairs of original snippets and their alternatives from 40k submissions. For the Decision Point Classification task (Section 3.1), we again apply Myers' algorithm at the word level between the input x and its alternative y to find its x^{DP} and y^{DP}. We make these datasets and all code available at github.com/behavioral-data/multiverse.

Validation. Data analyses include many different types of decisions. While we expect that decision type classifications will continue to evolve, possibly beyond the scope of our dataset, Wicherts et. al [40] contribute a useful taxonomy of nine different decision point types. We find that our dataset includes examples for all nine of these decision point types. Table A.1 in the Appendix shows this taxonomy of decision points with accompanying examples from our dataset, which demonstrates that our method of mining examples from Kaggle covers a diversity of decision points and alternatives. Furthermore, a human evaluation demonstrates that experienced data analysts are likely to accept machine-generated alternatives derived from this corpus (Section 6.4).

4.2 Code Libraries Represent Semantic Relationships

When developers build code libraries, they often organize their projects so that semantically related function and class definitions are contained within the same package or module. For example, in the popular Python scientific computing library `scipy`, the t-test (`scipy.stats.ttest_ind`) is defined in the same module as Mann-Whitney U (`scipy.stats.mannwhitneyu`), one of its non-parametric alternatives. To train our model to learn this information, we capture a graph of these relationships by mining libraries for their structure. We note that while we describe a method for mining this structure from Python, a similar process can be followed for any other object-oriented language like R or Julia.

Method. First, we mine import statements from our dataset (Section 4.1) to find the top ten most frequently used libraries. For each library, we start with the file invoked by the highest level import statement (e.g. `import scipy`) and then recursively create a node in our graph for each file, and then neighbor nodes for each sub-module it imports and each function or class defined in the file. For each class definition, we create neighbor nodes for each of its methods. For example, the class `scipy.stats` is the neighbor of `scipy`, and has a set of neighbors including `scipy.stats.mannwhitneyu` and `scipy.stats.ttest`. We note that this method is not guaranteed to

create a tree, since some sub-modules might be imported from multiple files. Since functions are rarely referenced by their full import path, at each node u we store $name_u$, the name of the class, function, or method. For example, the name of `scipy.stats.mannwhitneyu` is `mannwhitneyu`. Later, we will use these names to relate nodes of the library graph to tokens in code snippets (Section 5.3).

Processing and Data Filtering. Many nodes in the unprocessed graph (e.g. `numpy.testing.decorate_methods`) are unlikely to be of interest to analysts, and so we remove nodes if they have a parent called `testing` or `test` and all private methods. This results in a graph with over 60k nodes representing submodules, functions, and classes in popular data analysis libraries. We make these data available at github.com/behavioral-data/multiverse.

5 METHODS

Here, we describe the core components of MULTIVERSE (Figure A.1). We model code as a sequence of tokens, allowing for increased flexibility with respect to partial or syntactically incorrect snippets (Section 5.1). Our model combines a bidirectional transformer encoder with a left to right decoder to incorporate context from the whole input sequence (Section 5.2). Furthermore, we present a series of objectives, which we ultimately combine into one multitask objective (Section 5.3). Finally, we motivate and develop a novel seq2seq decoding strategy that can decide to simply copy input sequences into the output (Section 5.4).

Notation. Formally, given a token vocabulary V, and and input code sequence $x = \{x_t \in V\}, t = 0..N$, we train our model to produce a code alternative $y = \{y_t \in V\}, t = 0..N$. Our goal in conditional generation is to estimate:

$$P(y|x) = \prod_{t=0}^{t=N} P(y_t|x_0...x_N, y_0...y_{t-1}) \tag{1}$$

5.1 Model Inputs

MULTIVERSE represents code as a series of tokens, allowing it to handle messy, potentially incomplete code that might otherwise be unparseable to an AST-based method. Furthermore, by using a bytewise pair encoding (BPE) tokenizer (which splits rare and unknown words into more commonly occurring subtokens) we are able to model the large vocabularies that code corpora are known for [34, 37]. We also insert special tokens <INSERTED> and </INSERTED> at the beginning and end of subsequences in y^{DP} (Section 3.1), which we use in Span-Aware Beam Search (Section 5.4). Crucially, we only use these special tokens in training labels, and *not* in inputs, to simulate our use case where the user has no knowledge about the location of decision points in their code.

5.2 Bidirectional Encoder and Left-To-Right Decoder

MULTIVERSE combines a bidirectional transformer encoder with a left-to-right autoregressive decoder, similar to BART [25]. In MULTIVERSE and associated ablation studies (Section 6.2) we use four layers in both the encoder and the decoder, and four attention heads in each.

Encoder. MULTIVERSE uses a bidirectional transformer encoder with multi-head attention to calculate representations of the tokens in the input code sequence:

$$h_0 = W_e \cdot x + W_p \tag{2}$$
$$h_i = \text{TransformerBlock}(h_{i-1}), i \in 1...n \tag{3}$$
$$h_i^{norm} = \text{LayerNorm}(h_i) \tag{4}$$
$$E = h_n^{norm} \tag{5}$$

Where $W_e \in \mathbb{R}^{|V| \times d_e}$ is an initial embedding matrix and $W_p \in \mathbb{R}^{N \times d_e}$ is a learned positional embedding matrix. We experimented with embedding sizes, and found $d_e = 128$ to perform as well as larger embedding dimensions. The final input sequence embedding E is passed forward to the decoder.

Decoder. The left-to-right auto-regressive decoder generates a distribution over possible new tokens given the input context x and previously decoded tokens. During training, the decoder takes the alternative y shifted to the left (y^{shift}) as input, such that $y_i^{shift} = y_{i-1} \ \forall i \in 1..N, y_0^{shift} =$ <START_OF_SEQUENCE>. The decoder incorporates information from the input context by attending to the encoder output embedding E in each layer.

$$g_0 = W_e \cdot y^{shift} + W_p \tag{6}$$
$$g_i = \text{TransformerBlock}(g_{i-1}, E), i \in 1...n \tag{7}$$
$$g_i^{norm} = \text{LayerNorm}(g_i) \tag{8}$$
$$D = g_n^{norm} \tag{9}$$

Finally, we calculate:

$$P(y_t|x_0...x_N, y_0...y_{t-1}) = \text{softmax}(D_t) \tag{10}$$

5.3 Objectives

5.3.1 Decision Point Classification Loss. To address our decision point classification task (Section 3.1) we use a token-level classification head composed of a single linear layer with cross entropy loss over the encoder's final hidden states E:

$$x_t^{pred} = \text{SoftMax}(\text{LinearLayer}(E_t)) \tag{11}$$
$$x_t^{label} = \begin{cases} 1 & \text{if } t \in x^{DP} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$
$$\mathcal{L}_{cls}(x^{pred}, x^{label}) = \text{CrossEntropy}(x^{pred}, x^{label}, \gamma) \tag{13}$$

Where γ is a positive class weight we adopt to combat class imbalance, and E_t is the output embedding of the t^{th} input token. In practice, since decision point tokens represent approximately one tenth of the total tokens in our corpus, we set $\gamma = 10$.

5.3.2 Alternative Generation Loss. To train our model to generate alternative analyses (Section 3.2) we adopt a standard seq2seq cross entropy objective over the output of the model's decoder:

$$\mathcal{L}_{alt}(D, y) = \text{CrossEntropy}(\text{SoftMax}(D), y) \tag{14}$$

5.3.3 Library Graph Loss. In order to teach our model about the semantic relationships in libraries, we jointly embed our library graph with tokens from the input sequence. In particular, we optimize for graph distortion, which captures how the pairwise graph distances between n nodes in a graph U differ from their distance under a metric d_V in an embedding space V, where $g : U \rightarrow V$ is a

function that maps nodes to their embeddings:

$$Dist(g) = \frac{1}{\binom{n}{2}} \left(\sum_{u,v \in U; u \neq v} \frac{|d_V(g(u), g(v)) - d_U(u,v)|}{d_U(u,v)} \right) \quad (15)$$

The gradient of $Dist$ is undefined at $d_V(g(u), g(v)) = d_U(u,v)$, and numerically unstable elsewhere, and so prior work has focused on finding embeddings that minimize related objectives. In our setting, we define $d_U(x_t, x_j)$ as the undirected shortest path through the library graph between any two nodes u, v whose names name_u, name_v can be tokenized to include x_t and x_j, respectively. For example, `sklearn`'s KNN and SVM modules boths have methods named `predict_proba`, and so tokens corresponding to KNN and SVM would have a library graph distance of two. On average, 45% of the tokens in an input sequence can be related to the graph through this method. We define library embedding loss as

$$\mathcal{L}_{lib} = \beta \frac{1}{\binom{|x^g|}{2}} \left(\sum_{t,j \in x^g : t \neq j} \left| \frac{d_V(E_t, E_j)^2}{d_U(x_t, x_j)^2} - 1 \right| \right) \quad (16)$$

where x^g is the set of nodes in the graph whose corresponding tokens appear in the input, $E_u \in \mathbb{R}^{d_e}$ is the encoder embedding of token u: E_u, d_V is a metric over the embedding space, and β is a scaling factor. Since large graphs with low hyperbolicity like ours have been shown to exhibit the lowest distortion when embedded in euclidean space, we the L^2 norm as d_V [3]. Intuitively, this objective is minimized by embedding input tokens in a space where they are close to their neighbors in the library graph.

Multitask Objective. In MULTIVERSE we optimize each of these objectives jointly by adding them into a single multitask objective:

$$\mathcal{L} = \lambda_{cls}\mathcal{L}_{cls} + \lambda_{alt}\mathcal{L}_{alt} + \lambda_{lib}\mathcal{L}_{lib} \quad (17)$$

where each λ represents a constant weight on each term. In Section A.2 we detail which weights were used in the final model, and the process for determining these values.

5.4 Span-Aware Beam Search

Since decision points typically span only a fraction of the total tokens in the input sequence, most input tokens (90% in our dataset) also appear in the output. For example, in the code sequence `clf = LogisticRegression(X,y)`, only the tokens corresponding to `LogisticRegression` are likely to belong to a decision point (and are therefore likely to change in the alternative), while most reasonable alternatives would include the tokens corresponding to `clf =` and `(X,y)`. We hypothesize that a model could perform better if it only had to generate in the *"holes"* betweens non-decision points. To test this theory, we propose a modified version of beam search that forces the model to preserve tokens from the input if the decision point classifier indicates that they are unlikely to change (Section 5.3.1). Concretely, for each token if the softmax output of the positive class in the the decision point classification head (Equation 11) is above some threshold, then we only generate new tokens between this decision point and the next. Then, the beams proceed with standard generation until they hit a `</INSERTED>` token (at which point the beams are again constricted) or an `<END_OF_SEQUENCE>` token (generation ends).

6 EVALUATION

MULTIVERSE achieves a ROC AUC of 0.814 on the decision point classification task and a ROUGE-L F1 of 93.3% on the alternative generation task (Section 3). We additionally compare MULTIVERSE's performance on these tasks to ablations of the model and several state-of-the-art neural and non-neural baselines. Furthermore, to directly evaluate how useful MULTIVERSE's alternatives are to end users, we conduct a human evaluation (Section 6.4) and show that MULTIVERSE performs better than various baselines in syntactical correctness, reasonableness, and rate of end-user acceptance.

In the following evaluations MULTIVERSE is trained on submissions to 95% of competitions and is evaluated on the remaining 5% such that there is no overlap in submissions or competitions between the train and test sets. This ensures that the model never saw test-time analysis tasks during training, creating a conservative but realistic evaluation paradigm.

6.1 Task 1: Decision Point Classification

Baselines. To evaluate our model's performance on the decision point classification task relative to simple non-neural baselines and models from prior work, we benchmark MULTIVERSE against the following. Full results are available in Table 1.

- **How well does MULTIVERSE's encoder classify decision points on its own?** To answer this question, we independently train the MULTIVERSE encoder without loss from the decoder. We note that this experiment is comparable to BERT [6]. We denote this model as **MULTIVERSE No Decoder**.
- **How does random guessing perform on this decision point classification task?** To answer this question, we choose 10% (the proportion of decision points in the train set) of the token positions and take these tokens as decision points. We denote this model as **Random Guessing**.
- **How does a simple baseline that uses heuristics from the train set perform?** To answer this question, we construct a baseline that classifies any token that was labeled as a decision point in the train set as a decision point in evaluation.
- **How well does a sequential neural network for sequence labeling in NLP work on our task?** To answer this question, we use a **BiLSTM** model as our baseline. LSTMs have been used to model source code in the past [13]. This baseline is a multilayer bi-directional LSTM (BiLSTM) where the input sequence is processed both forward and backward [15]. After concatenating the forward and backward hidden states for each LSTM layer, we pass them through a linear layer to make the predictions of which tokens are decision points.
- **BiLSTM-CRF** How well does a BiLSTM-CRF model for sequence labeling tasks works on our task? While BiLSTM only considers the likelihood of the word being a certain tag, a CRF also calculates the transition scores [15]. This baseline considers the likelihood of a token being a certain tag given the previous token.

Discussion. While our two BiLSTM baselines outperform MULTIVERSE on accuracy and by a slim margin on ROC AUC, MULTIVERSE No Decoder achieves the best F1. While these models perform better than MULTIVERSE, only MULTIVERSE can incorporate this information into down-stream Alternative Generation through its multitask objective and Span-Aware Beam Search. In

Model	Accuracy	F1	ROC AUC
MULTIVERSE	64.4%	32.5%	0.814
MULTIVERSE No Decoder	88.7%	**45.7%**	0.831
Sample from Train Set	15.5%	17.4%	0.517
Random Guessing	83.1%	9.1%	0.500
BiLSTM [15]	**91.2%**	38.3%	0.828
BiLSTM-CRF [15]	89.8%	40.7%	**0.834**

Table 1: Comparisons of MULTIVERSE with various neural models and non-neural baselines for Decision Point Classification.

	ROUGE-L			Span-ROUGE-L		
Model	Precision	Recall	F1	Precision	Recall	F1
MULTIVERSE	**93.3%**	**85.9%**	**88.7%**	**30.5%**	**32.1%**	**25.6%**
Neural Code Translator [38]	29.2%	12.3%	15.7%	4.7%	5.7%	4.0%
SequenceR [4]	50.7%	24.9%	31.8%	5.3%	5.2%	4.1%
Copy Original Input	79.2%	76.7%	77.1%	0.0%	0.0%	0.0%
Randomly Replace Tokens	54.1%	55.0%	53.7%	1.3%	1.8%	1.3%
Tree Replace Tokens	15.7%	20.7%	17.3%	2.4%	8.4%	3.1%

Table 2: Comparisons of MULTIVERSE with various neural code seq2seq models and non-neural baselines. Our model consistently outperforms these baselines.

	Alternative Generation							Decision Point Classification		
	ROUGE-L			Span-ROUGE-L						
Model	Precision	Recall	F1	Precision	Recall	F1	GLEU	Accuracy	F1	ROC AUC
MULTIVERSE [1]	**93.3%**	**85.9%**	**88.7%**	30.5%	32.1%	25.6%	50.3%	64.7%	**32.5%**	**0.814**
No Library Graph [2]	92.5%	79.1%	84.5%	29.5%	**33.4%**	22.5%	39.2%	64.8%	31.8%	0.800
No Span-Aware Beam Search[3]	91.4%	83.2%	86.2%	**33.7%**	27.1%	**26.1%**	49.4%	65.3%	32.2%	0.787
No Multitask[4]	87.3%	54.8%	64.7%	20.4%	19.6%	17.2%	11.8%	**88.0%**	**44.0%**	0.811
Alternative Generation Only[5]	92.6%	84.5%	87.4%	31.7%	25.3%	24.1%	**51.1%**	N/A	N/A	N/A
MULTIVERSE- Full Information[6]	95.8%	93.5%	94.2%	35.0%	30.9%	28.6%	73.4%	N/A	N/A	N/A

1. Our complete multitask model, which includes Span-Aware Beam Search and optimizes $\mathcal{L}_{cls} + \mathcal{L}_{alt} + \mathcal{L}_{lib}$
2. A multitask model which includes Span-Aware Beam Search but not the library graph and optimizes $\mathcal{L}_{cls} + \mathcal{L}_{alt}$
3. A multitask model which does not include Span-Aware Beam Search and optimizes $\mathcal{L}_{cls} + \mathcal{L}_{alt} + \mathcal{L}_{lib}$
4. A model in which the MULTIVERSE encode and decode are trained independently
5. A model which only optimizes $\mathcal{L}_{alt}$
6. A model which receives perfect information about the location of decision points (Section 6.3).

Table 3: Comparison of various ablations of MULTIVERSE. We show that MULTIVERSE model performs best on the Alternative Generation task with respect to ROUGE-L, and that other ablations of MULTIVERSE perform best by a slim margin on Span-ROUGE-L metrics. Notably, MULTIVERSE outperforms all ablations when given access to information about the location of decision points (Section 6.3).

the next section, we will see how this multitask objective enables significantly improved generations. Interestingly, this multitask objective seems to trade increased performance on Alternative Generation for decreased performance on Decision Point Classification. This is likely because MULTIVERSE learns to classify tokens to limit the chance that it makes an error on Alternative Generation.

6.2 Task 2: Alternative Generation

Model Ablations. In order to evaluate how our model's individual components contribute to its overall performance, we train several ablations using different combinations of the components of its architecture. The results of the following studies are in Table 3.

- **MULTIVERSE** uses the bidirectional encoder and left-to-right decoder (Section 5.2), the decision point classification loss $\mathcal{L}_{cls}$ (Section 5.3.1), the library graph loss $\mathcal{L}_{lib}$ (Section 5.3.3), and span-aware beam search (Section 5.4). Formally, we optimize $\mathcal{L}$.
- **MULTIVERSE: No Library Graph** In this experiment, we try to understand how much of the model's performance can be attributed to information mined from our graph of library structures (Section 4.2). Accordingly, we optimize our model without the library embedding loss (Section 5.3.3), but maintain the decision point classification loss (Section 5.3.1) and the alternative generation loss (Section 5.3.2). Formally, we optimize $\mathcal{L}_{cls} + \mathcal{L}_{lib}$.
- **MULTIVERSE: No Span-Aware Beam Search** How does our model perform without Span-Aware Beam Search, when it can not explicitly copy likely non-decision-point tokens from the input? Here, we generate with conventional beam search.
- **MULTIVERSE: No Multitask** How does our model perform if we split it by treating our multitask objective as two discrete tasks? We separately train the encoder on Decision Point Classification and the decoder on Alternative Generation, then use predictions from the encoder for Span-Aware Decoding.
- **MULTIVERSE: Alternative Generation Only** How well does our model perform if we decide not to include library embedding loss and decision point classification loss, and focus on Alternative Generation only? To answer this question, we remove the corresponding terms from our objective. This baseline is similar to seq2seq transformer models like BART [25]. Formally, we optimize $\mathcal{L}_{alt}$ (Section 5.3.2).

Baselines. We also compare MULTIVERSE to a set of baselines.

- **Neural Code Translator** How well does a seq2seq neural model designed for code perform on our task? To address this question, we compare MULTIVERSE against the Neural Code Translator model, which applies an RNN to learn code changes implemented by developers [38].

- **SequenceR** How well does a bug fix neural model perform on our task? To answer this question, we compare MULTIVERSE against SequenceR, which uses a BiLSTM encoder-decoder model to generate simple, small patches for buggy code [4].
- **Original Input** Since decision points and alternatives span only a small fraction of the total sequence, even a model repeating the original code sequences could achieve high ROUGE scores. Therefore, we construct a trivial baseline that completely copies the original input for reference.
- **Randomly Replace Tokens** How does a simple baseline that has no knowledge of where decision points are perform? To answer this question, we construct a trivial baseline that randomly replaces input tokens with other tokens in the vocabulary. Consistent with the proportion of decision points in the dataset, we choose 10% of the token positions at random. If the token is chosen, we replace it with a random token from the vocabulary.
- **Tree Replace Tokens** How well does a simple baseline perform that considers structural information of common libraries? To address this questions, we construct a baseline that considers library nodes as decision points and objects and functions that were defined in the same modules as alternatives. If the token appears in the library graph, we randomly replace it with a randomly sampled neighbor in the graph.

Metrics. We adopt the standard seq2seq machine translation metric ROUGE-L for evaluating our ablations and baselines on the alternative generation task [27]. In essence, ROUGE judges similarities between prediction and target sequences by measuring co-occurrences of common n-grams. Since original code snippets often share a significant portion of their tokens with their alternatives, (more than 90% in our corpus) ROUGE-L will inflate the performance of models that simply copy code from the input to the prediction without interpreting the significance of decision points. We therefore also introduce an additional metric *"Span-ROUGE-L"*, which is ROUGE-L calculated only over those tokens that belong to decision points: Span-ROUGE-L$(\hat{y}, y)$ = ROUGE-L$(\hat{y}^{DP}, y^{DP})$. We also include the GLEU metric, which was originally developed to evaluate machine-generated grammatical error corrections [32]. Similar to ROUGE, GLEU measures n-gram overlap between a generated and ground truth sequence, but unlike ROUGE it penalizes models for predicting n-grams from the input sequence that should have changed in the output. To aggregate, we report the average of these metrics on each sequence in the dataset.

Discussion. MULTIVERSE performs up to three times better than other code-specific neural models, such as SequenceR [4] and Neural Code Translator[1]. Ablations show that all components of our approach, library graph, Span-Aware Beam Search, and multitask objective, lead to significant performance improvements across most evaluation metrics. One exception is the Alternative Generation model, which performs marginally better than MULTIVERSE on GLEU. However, this model is incapable of providing predictions on the Decision Point Classification task. Importantly, as shown in the next section, providing these predictions through the multitask formulation significantly boosts performance for MULTIVERSE, from 50.3% to 73.4% GLEU.

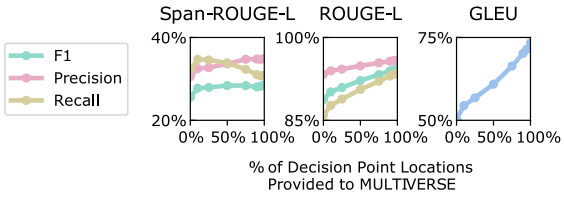

Figure 3: MULTIVERSE's performance under increasing information about the location of decision points (Section 6.3).

Method	Syntax	Reasonableness	Semantic Acceptance
MULTIVERSE(Our Method)	**4.30**	**3.23**	**2.94**
Alternative Generation Only	3.50	2.66	2.42
SequenceR [6]	2.17	1.48	1.42
Kendall's W	0.664	0.534	0.462

Table 4: Mean scores from our user study (Section 6.4). Five is the best possible score in each category. Our method significantly outperforms all baselines ($p < 0.01$, Wilcoxon's Signed Rank). Analysts achieved moderate to substantial agreement on their ratings (Kendall's W).

6.3 Simulating Known Decision Points

What if an analyst knew the spans for which they wanted to generate alternatives? How much better could MULTIVERSE perform? To answer this question, we pass the ground truth decision point labels, x^{label} (Section 5.1), to MULTIVERSE's Span-Aware Beam Search (Section 5.4). We find that under this condition MULTIVERSE's GLEU Score increases significantly from 50.3% to 73.4%. Importantly, on Span-ROUGE F1 and Recall MULTIVERSE's performance increases significantly from 25.6% to 28.5% and 30.5% to 35.0%. This suggests that if MULTIVERSE is given perfect information about the location of decision points by an analyst, it delivers better predictions within those decision points.

How much better does MULTIVERSE's Decision Point classifier need to be to reap these benefits? To answer this question, we take MULTIVERSE's Decision Point Classification predictions and with a given probability change the token prediction to be equal to the ground truth x^{label}. We find that increases in Decision Point Classification accuracy are approximately linear with increases in performance on ROUGE, Span-ROUGE, and GLEU. Results are available in Figure 3. This shows that improvements in Decision Point Classification would benefit Alternative Generation as well, even without additional changes to Alternative Generation models.

6.4 Human Evaluation

Ultimately, we are most interested in how useful a model is to an analyst. In addition to the automatic evaluation, we perform a user study of MULTIVERSE's predictions. Here we evaluate not just how well we do on automated metrics, but how well we perform on giving alternatives to analysts.

Five PhD-level data scientists with significant experience with Python for data science and machine learning were recruited as participants (co-authors were excluded from participation). The participants were asked to blindly evaluate outputs on their syntactical correctness, general "reasonableness" (defined as how well the result provided an alternative that a user might find useful)

and semantic acceptance (defined as how likely each rater would be to accept the suggestion in the context of a multiverse analysis). The five participants evaluated 50 examples of alternates from MULTIVERSE and two state-of-the-art neural baselines including a standard seq2seq transformer-based model (comparable to BART [25]) and a neural code generation model [38]. We chose these models for this evaluation because they had the best performance next to MULTIVERSE on Alternative Generation among the baselines (Table 2) and ablations (Table 3), respectively. Participants used a five point Likert scale for a total of 50x3x3x5=2250 ratings.

Notably, MULTIVERSE outperforms the baselines in all three criteria ($p<0.01$, Wilcoxon's Signed Rank Test). We observe moderate to substantial inter-rater reliability on all criteria (0.46-0.66, Kendall's W). Results can be found in Table 4. This shows that MULTIVERSE's predicted alternatives demonstrate higher syntactical correctness and reasonableness than comparable models. Furthermore, analysts are more accepting of MULTIVERSE's suggestions. We provide a detailed error analysis of MULTIVERSE's predictions in Section A.3.

7 CONCLUSION

In this paper we proposed two prediction tasks that support Multiverse Analysis [10, 36], a novel practice aimed at improving reproducibility in data science. Identifying decision points and suggesting alternative analysis approaches were operationalized as a classification task and a sequence-to-sequence prediction task, respectively. We share datasets to support these tasks, based on mining decision points from kaggle and the graph structures of common libraries (available at github.com/behavioral-data/multiverse).

We showed that by formulating Multiverse Analysis as a multitask problem, MULTIVERSE and compares favorably to neural baselines from prior work on both the decision point classification and alternative generation tasks (Section 6). Furthermore, if MULTIVERSE is given additional information about the location of decision points through Span-Aware Beam Search (guided by the analyst), its performance increases significantly beyond all other models, from 50.3% to 73.4% GLEU (Section 6.3). Finally, we showed through a human evaluation that data analysts find MULTIVERSE's alternatives to be more syntactically correct, reasonable, and acceptable than alternatives from other transformer and LSTM-based models. Our work demonstrates the feasibility of learning to recommend alternative analyses by mining collective data science knowledge from the web and has implications for improving reproducibility by supporting multiverse analysis.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. (2014).
[2] I. Chami, A. Wolf, D. Juan, F. Sala, S. Ravi, and C. Ré. 2020. Low-Dimensional Hyperbolic Knowledge Graph Embeddings. *arXiv:2005.00545* (2020).
[3] Ines Chami, Rex Ying, Christopher Re, and Jure Leskovec. 2019. Hyperbolic Graph Convolutional Neural Networks. *NeurIPS* (2019).
[4] Zimin Chen, Steve James Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. 2019. Sequencer: Sequence-to-sequence learning for end-to-end program repair. *TSE* (2019).
[5] S. Chollampatt and H.T. Ng. 2018. A multilayer convolutional encoder-decoder neural network for grammatical error correction. *arXiv:1801.08831* (2018).
[6] J. Devlin, M.W. Chang, K. Lee, and K. Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805* (2019).
[7] Bijaya et. al. 2018. Sub2Vec: Feature Learning for Subgraphs. In *KDD*.
[8] Silberzahn et. al. 2018. Many Analysts, One Data Set: Making Transparent How Variations in Analytic Choices Affect Results. *Advances in Methods and Practices in Psychological Science* 3 (2018).
[9] Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. 2017. Style transfer in text: Exploration and evaluation. *arXiv:1711.06861* (2017).
[10] Andrew Gelman and Eric Loken. 2014. The garden of forking paths. (2014).
[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*.
[12] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *SIGSOFT*.
[13] Vincent J Hellendoorn and Premkumar Devanbu. 2017. Are deep neural networks the best choice for modeling source code?. In *FSE*.
[14] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The Curious Case of Neural Text Degeneration. arXiv:cs.CL/1904.09751
[15] Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. arXiv:cs.CL/1508.01991
[16] J. Weston Hughes, Keng-hao Chang, and Ruofei Zhang. 2019. Generating Better Search Engine Text Advertisements with Deep Reinforcement Learning. *KDD*.
[17] JetBrains. 2018. JetBrains Data Science in 2018.
[18] Shaojie Jiang, Pengjie Ren, Christof Monz, and Maarten de Rijke. 2019. Improving Neural Response Diversity with Frequency-Aware Cross-Entropy Loss. In *WWW*.
[19] Kyle Kelley and Brian Granger. 2017. Jupyter Frontends: From the Classic Jupyter Notebook to JupyterLab, nteract, and Beyond. *JupyterCon* (2017).
[20] M.B. Kery, B.E. John, P. O'Flaherty, A. Horvath, and B.A. Myers. 2019. Towards effective foraging by data scientists to find past analysis choices. In *CHI*.
[21] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *CHI*.
[22] Mary Beth Kery and Brad A. Myers. 2018. Interactions for Untangling Messy History in a Computational Notebook. In *VL/HCC*.
[23] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2015. Combining deep learning with information retrieval to localize buggy files for bug reports (n). In *ASE*. IEEE.
[24] Etienne P. LeBel, Randy J. McCarthy, Brian D. Earp, Malte Elson, and Wolf Vanpaemel. 2018. A Unified Framework to Quantify the Credibility of Scientific Findings. *Advances in Methods and Practices in Psychological Science* 3 (2018).
[25] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. 2019. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *arXiv:1910.13461* (2019).
[26] Piji Li, Wai Lam, Lidong Bing, and Zihao Wang. 2017. Deep recurrent generative decoder for abstractive text summarization. (2017).
[27] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*.
[28] Y. Liu, T. Althoff, and J. Heer. 2020. Paths Explored, Paths Omitted, Paths Obscured: Decision Points & Selective Reporting in End-to-End Data Analysis. *CHI* (2020).
[29] Yang Liu, Alex Kale, Tim Althoff, and Jeffrey Heer. 2020. Boba: Authoring and visualizing multiverse analyses. arXiv:cs.HC/2007.05551
[30] Alireza Mohammadshahi and James Henderson. 2020. Graph-to-Graph Transformer for Transition-based Dependency Parsing. *arXiv:1911.03561* (2020).
[31] Eugene W. Myers. 1986. AnO(ND) difference algorithm and its variations. *Algorithmica* 1 (1986).
[32] Courtney Napoles, Keisuke Sakaguchi, Matt Post, and Joel Tetreault. 2016. GLEU without tuning. *arXiv preprint arXiv:1605.02592* (2016).
[33] Daniel Povey and et. al. 2011. The Kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society.
[34] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural Machine Translation of Rare Words with Subword Units. *CoRR* (2015). arXiv:1508.07909
[35] Uri Simonsohn, Joseph P. Simmons, and Leif D. Nelson. 2015. Specification Curve: Descriptive and Inferential Statistics on All Reasonable Specifications. *SSRN Electronic Journal* (2015).
[36] Sara Steegen, Francis Tuerlinckx, Wolf Vanpaemel, and Andrew Gelman. 2016. Increasing Transparency Through a Multiverse Analysis. *Perspectives on Psychological Science* 5 (2016).
[37] A. Svyatkovskiy, S.K. Deng, S. Fu, and N. Sundaresan. 2020. IntelliCode Compose: Code Generation Using Transformer. *arXiv:2005.08025* (2020).
[38] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. 2019. On learning meaningful code changes via neural machine translation. In *ICSE*. IEEE.
[39] Wei-Hung Weng, Yu-An Chung, and Peter Szolovits. 2019. Unsupervised Clinical Language Translation. *KDD*.
[40] J.M. Wicherts, C.L.S. Veldkamp, H.E.M. Augusteijn, M. Bakker, R.C.M. van Aert, and M.A.L.M. van Assen. 2016. Degrees of Freedom in Planning, Running, Analyzing, and Reporting Psychological Studies: A Checklist to Avoid p-Hacking. *Frontiers in Psychology* (2016).
[41] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *NeruIPs*.
[42] G. Zhang, M.A. Merrill, Y. Liu, J.Heer, and T. Althoff. 2020. CORAL: COde RepresentAtion Learning with Weakly-Supervised Transformers for Analyzing Data Analysis. arXiv:cs.LG/2008.12828

Decision Point	Original	Alternative
Missing Data	`train_csv.fillna(method = 'ffill')`	`train_csv.dropna().reset_index()`
Outliers	`max = np.quantile(duration.values(),0.9)`	`max = np.quantile(duration.values(),0.8)`
Filtering	`weapon = train[train["weaponsAcquired"]<25]`	`weapon = train[train["weaponsAcquired"]<=20]`
Data Type	`cat=KBinsDiscretize(encode='original')`	`cat=KBinsDiscretizer(encode = 'ordinal')`
Assumptions	`pvals = ttest_ind(sales)`	`pvals = ttest_ind(sales,equal_vars=True)`
Variable Selection	`smf.ols('SALARY_MILLIONS ~ WINS_RPM')`	`smf.ols('SALARY_MILLIONS ~ POINTS')`
Model Selection	`SVClr = SVC()`	`KNNclr = KNeighborsClassifier()`
Estimation Method	`make_pipeline(StandardScaler(), RidgeCV)`	`make_pipeline(RobustScaler(), RidgeCV)`
Inference Criteria	`final_vars = results.pvalues <=0.25`	`final_vars = results.pvalues <=0.15`

Table A.1: Taxonomy of types of decision points in our dataset, with real examples from Kaggle of each type of decision point.

A REPRODUCIBILITY APPENDIX

A.1 Availability of Data and Code

We make all data, code, and model checkpoints available at github.com/behavioral-data/multiverse.

A.2 Training Details

We trained every version of MULTIVERSE on a single Nvidia GeForce 2080Ti for 20 epochs. Training took about six hours with $d_e = 128$ and a maximum sequence length of 128 tokens. After a grid search , we found that an initial learning rate of $3e^{-5}$ with linear decay to zero worked best, with $\lambda_{class} = 1$, $\lambda_{alt} = 1$, $\lambda_{lib} = 0.1$.

Early experiments showed that if $\mathcal{L}_{lib}$ was used early in training MULTIVERSE diverged and ultimately settled into poor local optima. To compensate, in the final objective we only add this term after a one epoch burn-in period.

A.3 Error Analysis

We conducted a qualitative error analysis of MULTIVERSE's predictions to identify areas for future improvement. MULTIVERSE's common errors can be grouped into three categories: degeneration, semantic failure, and input copying.

Degeneration. Degeneration is a common problem in seq2seq models where generation gets *"stuck"* in a loop. In one example alternative, MULTIVERSE predicts `y_train, y_test, y_train, y_test` until it reaches its maximum length threshold. One likely explanation is that `y_test` and `y_train` are the most likely tokens to follow one another under the language model. Solutions may include alternative sampling methods (such as top-k sampling or nucleus sampling) instead of beam search [14].

Semantic Failure. One feature of MULTIVERSE is that it can easily be trained on any computer language. However, this flexibility comes at the expense of built-in information about underlying language syntax. While MULTIVERSE scores highest against comparable models in syntactical correctness (Section 6.4), we observe that the model still occasionally produces malformed Python code, such as `df =, pd.DataFrame(()`. Prior work indicates that current-generation language models like ours can learn to generate nearly perfectly correct code given millions of training examples [37]. We believe that additional pre-training on large unsupervised corpora could mitigate this error.

Input Copying. In some cases, MULTIVERSE appears to have learned to exactly repeat the input sequence as its output. One explanation is that since input sequences in our dataset on average share 90% of their tokens with their corresponding output sequences, the model can score reasonably well by naively copying the input. Alternate formulations of cross entropy loss, such as frequency-aware cross entropy loss, have been shown to promote output diversity [18], and could penalize MULTIVERSE for repeating its input during training. However, in an early version of the model we implemented this loss function and saw no significant improvement in performance.

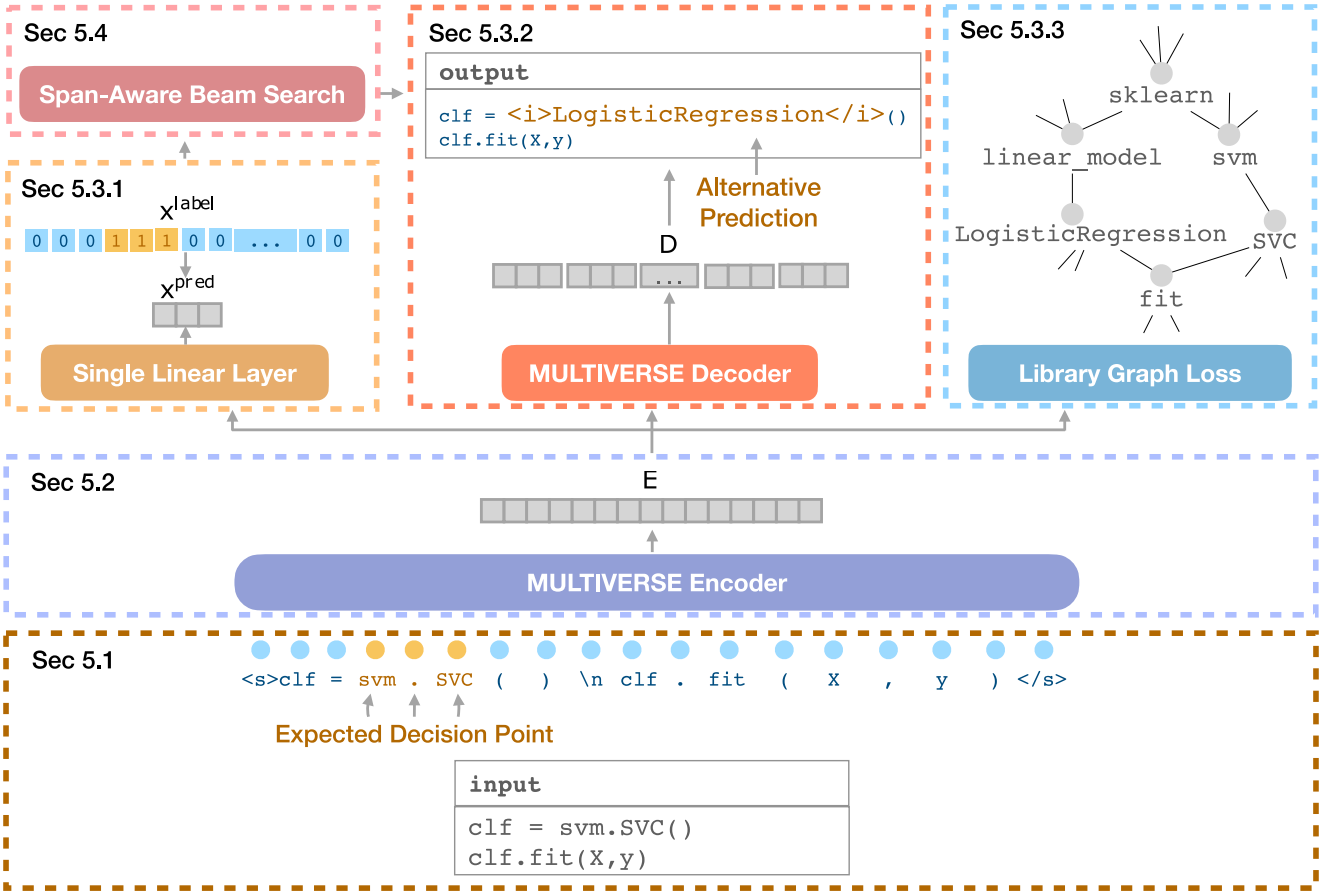

Figure A.1: Our MULTIVERSE model, which combines Decision Point Classification, Alternative Generation and library embedding into a multitask objective.