Proposal information extraction

General Information			
Name of experiment	PtrGNCMsg	Authors	Qin Liu, Zihe Liu, Hongming Zhu, Hongfei Fan, Bowen Du, Yu Qian
Year of publication	2019	Available online	Available at link - https://zenodo.org/records/2542706#.XECK8C277BJ
Publication paper name	Generating Commit Messages from Diffs using Pointer-Generator Network	Notes on availability	N/A

Proposal Information	
Preprocessing process	Obtain pair <diff,message> Convert to vector Input to DNN embedding layer
Input Format	Vector (X1...Xn) of tokens. Tokenized according to citations 32,33,34, which are Seq2Seqk, AST for code and semantic parsing and review about networks for code generation
Input Size	512 tokens (between 30 and 100 words)
DNN Architecture	
Approach	Transform commit generation to Seq2Seq translation problem from code to natural language Takes into consideration the Out of Vocabulary words and reducing the DNN vocab. Size by using a cover rate that discards rarely used words to enhance commit accuracy and performance while keeping the important information for the commit messages. Also it uses a probability formula to copy words from source sentence to deal with OOV
Overall architecture	RNN Encoder-Decoder with attention Mechanism (Transformer)
Layers	4 layers of GRU cells (embedding, encoder,decoder,output)
Cells per Layer	512 cells per layer
Weights and Biases	No explicit method (Random) for weights. Seed=3 and Initial Weights=0.1 Biases initialized with Xavier/Glorot from Tensorflow
Other Architecture hyperparameters	Min input size = 30 (words) / Max input size = 100 (words)
Loss function	Negative conditional log-likelihood

Dataset	
Dataset Origin	Jiang et.Al top 1,000 Java projects + Self-retrieval until top 2,081 top java projects from GitHub
Dataset Composition	32,663 pairs <diff, message> from Jiang et Al. 32,208 pairs from self collection
Dataset Adjustment/Processing	Initial extraction from projects order by stars Remove merge,callbacks, lowercase to normalize and tokenize. Filter max. length and filter by Verb-DirectObject Split in training, validation and test
Dataset Division	Random division: Training: 50,052 (77%) Validation: 7,989 (12%) Test: 6,920 (11%)
Dataset available online	Available at link - https://zenodo.org/records/2542706#.XECK8C277BJ
Technological infrastructure	
Hardware	Computer equiped with GTX 1080Ti 12G GPU
Software	TensorFlow 1.9.0 absl-py 0.3.0 / astor 0.7.1 / astroid 2.0.1 / gast 0.2.0 / grpcio 1.13.0 / isort 4.3.4 / javalang 0.11.0 / lazy-object-proxy 1.3.1 / Markdown 2.6.11 / mccabe 0.6.1 / nltk 3.3 / numpy 1.15 / protobuf 3.6 / pylint 2.0.1 / pymongo 3.7.1 / siz 1.11 / tensorboard 1.9 / termcolor 1.1 / typed-ast 1.1 / werkzeug 0.14.1 / wrapt 1.10.11
Time and epochs	220 minutes to complete 45 epochs
Optimization Hyperparameters	Max epochs = 1000 / Learning rate = 0.001 / Adam optimizer / Beam width = 3 / Forget Bias = 1.0 / Batch size = 16 / Vocabulary Cover Rate = 0.9
Regularization Hyperparameters	Implicit L1/L2 regularization in Adam Optimizer / Dropout rate = 0.3 / Early stopping if BLEU does not improve on validation on last 10 epochs
Training process	Use different sets of hyperparameters until selection of final ones. Apart from described hyperparameters, dataset is randomly shuffled after each epoch. Model to start next epoch is only saved if BLEU improves on validation set. Early stopping is also set if BLEU does not improve on validation on last 10 epochs.

General Information			
Name of experiment	ATOM	Authors	Shangqing Liu, Cuiyun Gao, Sen Chen, Lun Yiu Nie, Yang Liu
Year of publication	2022	Available online	Not available - link provided https://github.com/shangqing-liu/ATOM
Publication paper name	ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking	Notes on availability	Link not working 404 error on GitHub

Proposal Information		
Preprocessing process	Performed on Embedding module - processes pairs <diff,message>: For diff - lines divided in added/deleted, enhance syntactic comprehension via CTAGS, parse to an AST using JAVAPARSER For message - only first sentence considered, filenames and digits replaced by placeholders and tokenization, lemmatization using NLTK to reduce vocabulary size	
Input Format	Lemmatized and preprocesse commit message (only training) AST representing the code diff (regular input) - using JAVAPARSER	
Input Size	Max 128 tokens (+128 of initial context added by DNN AST2Seq which doesnt come from diff)	
DNN Architecture		
Approach	Combine generation and retrieval approaches in order to increase the metrics and quality of commit message generation by ranking which alternative performs better for each input	
Overall architecture	4 modules: Preprocessing Module - clears and processes the input in the right format Generation Module - Generates message using AST2Seq DNN Retrieval Module - no DNN, retrieves via cosine similarity the most similar to diff. Uses TDF-IF score for comparison and retrieval Ranking Module - convolutional DNN, ranks which alternative is more suitable. Uses similarity matrix as input for ConvNet	
Contained DNNs	AST2Seq (Generation Module)	ConvNet (Ranking Module)
Architecture	RNN Encoder-Decoder with attention mechanism (Transformer)	2D convolution + Maxpooling + Fully connected layer
Layers	3 layers Bidirectional LSTM(Encoder, Attention, Decoder)	3 layers (2D convolution, maxpooling, relevance score calculation via fully connected layer)
Cells per Layer	256 in embedding for encoder. Rest of layers not mentioned	16 kernels in convolution for 256 input
Weights and Biases	Initialization not mentioned, model not stored	Initialization not mentioned, model not stored
Other Architecture hyperparameters	Max. AST paths in one AST = 80 / Hidden state = 128 Attention mechanism = Luong Grid search to optimize values	Kernel size for convolution (3,3) / Stride size at maxpooling = (2,2) / RELU function after convolution / Ranking with LTSM+Attention scoring Grid search to optimize values
Loss function	Softmax cross-entropy with logits	Mean Square Error

Dataset		
Dataset Origin	New dataset created for the paper - Available at https://zenodo.org/records/4077754#.X4K2b5MzZTY Potential leakage possibilities between sets due to division criteria	
Dataset Composition	~160k pairs commit message and diff from top 56 Java projects on GitHub, including Neo4j, Structs, Antlr4... projects in a formatted way attending to id, message, subject, project...	
Dataset Adjustment/Processing	Recovery of noisy data from GitHub, filter out merge and rollbacks Remove commits above 5 chunks and non .java files AST extraction for remaining data and removal of message over 20 words and duplicates	
Dataset Division	Random division: Training: 128,000 (80%) Validation: 16,000 (10%) Test: 16,000 (10%) Also divided by Timestamp and Project for last part of paper	
Dataset available online	Available at https://zenodo.org/records/4077754#.X4K2b5MzZTY	
Technological infrastructure		
Hardware	Server with 36 cores and 4 NVIDIA GPUs models TELSA P40 and M40, 22GB VRAM each	
Software	Tensorflow 1.12 Pytorch 1.4 No further specification in paper and not available online to extract more	
Time and epochs	Training took 16 hours and testing 257 seconds, no number of epochs mentioned	
Contained DNNs	AST2Seq (Generation Module)	ConvNet (Ranking Module)
Optimization Hyperparameters	Max epochs = 3000 / Batch size = 256 / Adam optimizer/ Learning rate = 0.0001 / Beam width = 5 Rest of hyperparameters according to citations 58 and 20 (Code2Seq).	Adam optimizer / Learning rate) 0.0001 Rest of hyperparameters according to citations 58 and 20 (Code2Seq)
Regularization Hyperparameters	Dropout rate = 0.4 / Implicit L1/L2 regularization on Adam optimizer / Early stopping at 20 epochs Rest of hyperparameters according to citations 58 and 20 (Code2Seq)	Implicit L1/L2 regularization on Adam optimizer Hyperparameters set according to citations 58 and 20 (Code2Seq)
Training process	Both DNN are trained at same time as they are part of the whole system. Grid Search is used to adjust the fitting of Learning rate, embedding size, encoder/decoder layer size and kernel sizes. Also trained in dataset separated by Timestamp and project to compare results	

General Information			
Name of experiment	NMT	Authors	Siyuan Jiang, Ameer Armaly, and Collin McMillan
Year of publication	2017	Available online	NOT AVAILABLE - Link provided https://sjiang1.github.io/commitgen
Publication paper name	Automatically Generating Commit Messages from Diffs using NMT	Notes on availability	Link provided but error 404 not found in GitHub when trying to open it

Proposal Information	
Preprocessing process	Extract pair of code diff and commit message from dataset Use NEMATUS (variation of WNMT) to obtain factors and the embedding as the concatenation of those factors
Input Format	Not mentioned in Paper (referenced to Nematus). From Nematus is Matrix resulting of concatenating vectors of each factor/feature at each source position. See Nematus: A toolkit for NMT
Input Size	Input size from embedding layer is 512 (max length for code diff set at 100)
DNN Architecture	
Approach	The paper approachs the commit generation problem as an application of the translation mechanism using the capabilities of Neural Machine Translation to translate a message between languages. NMT is adapted to translate from "code" to "english" or message.
Overall architecture	Uses a RNN Encoder-Decoder with Bahdanau Attention Mechanism (also called Transformer) implemented by adapting Nematus. Adaptations means feed-forward hidden layer in decoder. Also no bias on word embedding layers.
Layers	Not mentioned in Paper (referenced to Nematus). From Nematus the number of layers is a hyperparameter that is not mentioned and cannot be extracted from link since it is not accesible.The layers are composed of GRU cells (changed from LSTM cells used in Nematus)
Cells per Layer	512 for embedding layer. 1024 for hidden layer
Weights and Biases	Not mentioned, referenced to Sennrich et al. WNMT. WNMT paper references it to Bahdanau et al. In that paper, initialization of bias are vectors with 0 and recurrent weight matrices initialized as random orthogonal matrices.
Other Architecture hyperparameters	Minibatch training / learning algorithm = SGD with AdaDelta
Loss function	Cross-entropy Minimization

Dataset	
Dataset Origin	Dataset used is a preprocessing adaptation from 2M Dataset proposed by Jiang and McMillan.
Dataset Composition	Content of dataset is 32,000 preprocessed pairs of <diff,message> extracted from top 1,000 java projects in order of stars, extracted from 2M+ commits from Jiang and McMillan dataset
Dataset Adjustment/Processing	Take datapoint from reference dataset of Jiang and McMillan Filter first sentence, remove commit ID, issue ID, merge and rollback commits Remove commits larger than 1Mb, filter out commits longer than 30 words (msg) or 100 tokens (diff) Apply a VDO (Verb-Direct Object) filter - using Standford CoreNLP
Dataset Division	Random division of dataset into: Training set - 26,000 pairs (81.25%) Validation set - 3,000 pairs (9.375%) Test set - 3,000 pairs (9.375%)
Dataset available online	Not available, said it is with link https://sjiang1.github.io/commitgen - Not accesible
Technological infrastructure	
Hardware	Computer with Nvidia GeForce GTX 1070 with 8Gb of VRAM. No further details mentioned.
Software	No mention of software versions used. As they reference Nematus, looking paper and repository, we extract: Python 3.5.2 Tensorflow versions from 1.15 to 2.0
Time and epochs	Training took 38 hours and stopped at 210,000 minibatches of 80 elements fed to the model. Testing on same GPU took 4.5 minutes
Optimization Hyperparameters	Training settings taken from Sennrich et al. (for rest of hyperparameters) Minibatch size = 80 / Max epochs = 5,000 / Max number of minibatches = 10M / Early stopping / Vocabulary truncated at 50k tokens / Adadelta optimization for learning rate
Regularization Hyperparameters	Dataset reshuffled each epoch to avoid overfitting Implicit adaptative learning rate decay in Adadelta optimizator Rest of hyperparameters set to Sennrich et al.
Training process	The model is configured taking the settings from Sennrich et al. WNMT paper. The reference dataset is preprocessed to extrac relevant and quality information to train NMT. Vocabulary truncated to 50k out of 65k to increase performance. Dataset reshuffled after each epoch. Training feeds 30k minibatches to model and saves weights and bias, checks for early stopping, if not possible, continues.

General Information			
Name of experiment	CoDiSum	Authors	Shengbin Xu , Yuan Yao , Feng Xu , Tianxiao Gu , Hanghang Tong and Jian Lu
Year of publication	2019	Available online	Available at https://github.com/SoftWiser-group/CoDiSum - Link found not provided
Publication paper name	Commit Message Generation for Source Code Changes	Notes on availability	Lots of aspects not mentioned in the paper are mentioned because they were extracted from link

Proposal Information	
Preprocessing process	Preprocessing made to the dataset proposed by Jiang and McMillan to create datset for CoDiSum Input is taken from dataset, code structure extracted From code structure, identifiers are replaced by placeholders and semantics assigned to code Semantics and structure fed to separate bidirectional GRU layers Input consists on combination of placeholders /semantics from code and structure
Input Format	For code semantics, input format is vector [x1...xn] with xi being the embedding of word i in a lookup table For code structure, both vector of embeddings and output of semantics is fed to bi-GRU layer
Input Size	Input from semantics layer set to max. 5 tokens Input from structure layer set to 200 max length
DNN Architecture	
Approach	The approach tries to modify the first element of the Transformer (the Encoder layer) to learn semantics and structure of the code diff separatedly and combine the knowledge extracted in the attention mechanism. Instead of just concatenating the information, the approach tries to combine the knowledge extracted from the semantics and the structure to send a representation to the Decoder that better represent the meaning of the code changes. For the rest, it still approaches the problem as a translation using a Transformer to obtain the message, but decoder also incorporates a placeholder substitution module to copy identifiers from source.
Overall architecture	RNN Encoder-Decoder with Attention mechanism (Transformer) with modification on Encoder and Decoder. Two separate bi-GRU layers deal with code structure and representation. Decoder adds placeholder substitution to copy identifiers to the final result. Attention layer uses Convolutional operation
Layers	3 layers for Encoder components (both bi-GRU layers) and 3 layers for Decoder (attention and hidden),finally, copying mechanism according to See et al. 2017
Cells per Layer	In bi-GRU layer (Encoder) - Initial hidden state of 128 for each dimension (total 256) and hidden layer size of 256 In Decoder - hidden layer size of 256
Weights and Biases	Biases initialized to 0. Random initialization of weights with seed = 1. From code file CopyNetPlusGen.py Final model not mentioned to be stored in the paper.
Other Architecture hyperparameters	Structure length = 200 / Semantics length = 5 / message length (in training) = 20 / word embedd dimension = 150 / stride size 1,1 for Convolution / kernel initializer = glorot_uniform
Loss function	Categorical Cross Entropy Loss

Dataset	
Dataset Origin	Dataset used for CoDiSum comes from 509k+ Jiand and McMillan dataset preprocessed and filtered to enchance quality of resulting dataset. Comes from top 1,000 Java projects ordered by stars
Dataset Composition	The dataset, after preprocessing, contains 90,661 pairs of <diff, message> extracted from preprocessing Jiang and McMillan dataset.
Dataset Adjustment/Processing	From Jiang and McMillan dataset: Remove non .java file related commits and remove code of files with no code diffs Use NLP to remove punctuation and special symbols Tokenize diffs and message and remove < 3 words Delete duplicate commits and various modifications in short period of time
Dataset Division	Random division of pairs from dataset into: Training set - 75,000 pairs - (82.7%) Validation set - 8,000 pairs - (8.8%) Test set - 7,661 pairs - (8.5%)
Dataset available online	Dataset available at same link for experiment - https://github.com/SoftWiser-group/CoDiSum
Technological infrastructure	
Hardware	Not specified in the paper neither in the code repository
Software	From code repository: Python >= 3.6.2 Numpy >= 1.15.0 / Keras >= 2.1.5 / Tensorflow >= 1.7.0
Time and epochs	No time or epochs to train specified neither on the paper or the GitHub repository
Optimization Hyperparameters	Batch size = 100 / max epochs = 50 / test batch size = 1 / patience = 2 / Beam size = 2 / RMSProp Optimizer
Regularization Hyperparameters	Implicit adaptive learning decay regularization in RMSProp Optimizer Dropout rate = 0.1
Training process	Training is slightly describe in the paper and the repository. Authors set the hyperparameters describe in section 3 and run the model to train. No more mention of infrastructure or training process further than hyperparameters mentioned is provided. They say training is stopped by them when loss is no longer decreasing.

General Information			
Name of experiment	CommitBERT	Authors	Tae-Hwan Jung
Year of publication	2021	Available online	Available at link https://github.com/graykode/commit-autosuggestions
Publication paper name	CommitBERT: Commit Message Generation Using Pre-Trained Programming Language Model	Notes on availability	Available and some important information not contained in the paper extracted from GitHub repository

Proposal Information	
Preprocessing process	Dataset used contains added and deleted pairs of diffs and the reference message Input gets to the Encoder layer using Byte Pair Encoding (BPE) Message is fed using BPE to Decoder in training and validation Code diff is fed also using BPE but on a sequence of starting token + added + separator + deleted
Input Format	The input format is Byte Pair Encoding of the code diff structure cls+added+separator+deleted
Input Size	Input size is 256 for the code diff
DNN Architecture	
Approach	This approach explores how to adapt a well known solution for NLP to commit message generation taking as inspiration the variation CodeBERT, which is an implementation of BERT for code. CodeBERT is pretrained on learned relationship between NL and code and calculates probability of next word of code. The authors proposal is to adapt a variation of CodeBERT called RoBERTa, in combination with CodeBERT, and adapt those pretrained models to the task of generating commit messages following the NMT inspirating principles.
Overall architecture	The model is based on the Transformer implemented in CodeBERT and RoBERTa (which ultimately are RNN Encoder-Decoder with Attention mechanism). The model uses the following layers: Embedding layer based on BPE - from CodeBERT Encoder layer based in Transformers - from RoBERTa Decoder layer based in Transformer - from CodeBERT - also uses SOFTMAX function to output
Layers	12 layers for encoder and 3 layers for decoder, based in Transformer architecture, which typically uses bidirectional LSTM cell layers
Cells per Layer	For the encoder, RoBERTa uses same size as CodeBERT, so it is 768 cells per layer. For the decoder, according to CodeBERT Feng et al., is also 768 cells per layer
Weights and Biases	Weights and biases are initialized as CodeBERT weights, which will then be trained according to Code-to-NL task in CodeSearchNet experiment to reduce gap betweem code and NL. Model saved after training available in link https://github.com/graykode/commit-autosuggestions/blob/master/commit_autosuggestions.ipynb
Other Architecture hyperparameters	Max source input size = 256 / Targe input max length = 128 / Rest of hyperparameters loaded from microsoft/codenert-base and microsoft/roberta-base
Loss function	Not specified in paper. From file model.py extracted Cross-Entropy Loss

Dataset	
Dataset Origin	Dataset is created by the author to train and test the proposed approach, it comes from 52,000 projects on GitHub in languages Python, PHP, Go, Java, JavaScript, and Ruby
Dataset Composition	The dataset contains 345,000 pairs of <diff,message> where diff is divided in added and deleted lines. From each project of 52k a max of 50 commits can be in the dataset.
Dataset Adjustment/Processing	Top 52,000 projects in languages Python, PHP, Go, Java, JavaScript, and Ruby cloned using multithreading Max 50 commits taken from each project, filter out merge commits and commits not related to main extension (.py, .php, .js, .java, .go, .ruby) Only added and deleted lines taken from remaining commits Filter out only 1-2 lines changed, commits with issue number and non-English commits Remaining commits truncated to first sentence and filter out more than 32 token messages Only keep commits starting with VDO add, fix, use, update, remove, make, change, move, allow, improve, implement, create and upgrade
Dataset Division	Dataset divided randomly as follows: Training set - 276,606 - (80%) Validation set - 34,576 - (10%) Test set - 34,576 - (10%)
Dataset available online	Dataset available at provided link https://github.com/graykode/commit-autosuggestions
Technological infrastructure	
Hardware	V100 GPU, models have 16-32 GB of VRAM but not specified which. No mention of other hardware elements
Software	No mention in paper of software settings. From code we can extract: CUDA version 10 and CUDNN (support for AI development) version 7 sentencepiece==0.1.91 / transformers==3.4.0 / flask==1.1.2 / gitpython / jsonlines / tqdm / pydriller / transformers==3.4.0 / wandb / knockknock / pytorch
Time and epochs	Paper states "About 23 hours for added model and about 20 hours for diff model". No mention of epochs
Optimization Hyperparameters	Batch size = 64 in code (32 stated in paper but does not correspond) / Learning rate = 5e-5 / Beam width = 10 / max steps = 50000 / max eval steps = 1000 / epochs = 10 / AdamW SGD optimizer
Regularization Hyperparameters	L1/L2 regularization implicit in AdamW SGD optimizer No mention of other regularization techniques
Training process	Main training fact is that input is passed to model and model configuration passed using CodeXGLUE pipeline library https://github.com/microsoft/CodeXGLUE. No more information specified in the paper about the training process

General Information			
Name of experiment	COME	Authors	Yichen He, Liran Wang, Kaiyi Wang, Yupeng Zhang, Hang Zhang, Zhoujun Li
Year of publication	2023	Available online	Yes - at link https://github.com/hyc2026/come/tree/master
Publication paper name	COME: Commit Message Generation with Modification Embedding	Notes on availability	Link for intermediate models stored and final experiment

Proposal Information	
Preprocessing process	This proposal takes data from two datasets, which provide the code diffs for the approach First take the input and rearranging the tokens based on the edit distance (min edits to go from previous to target string in code) Then tokens are given a scored (1=Deleted/2=Added/0=Unchanged) Input for the DNN is the sum of rearranged tokens and their respective scores
Input Format	Is a vector containing the sum of the score and the rearranged token. Emphasis on the fact that it is a sequential structure instead of a complex one like an AST to enhance performance (it is a fine grained approach) by avoiding additional complexity (simple sequential structure)
Input Size	Max. Length of input 512 positions (from code repository file run.sh)
DNN Architecture	
Approach	This approach is proposed after some previous approaches which combine generation and retrieval. The approach aims to solve the imprecisions of previous approaches combining these techniques in two main aspects: input representation and generation-retrieval integration. To do this, it formats the input in a fined-grained format (contains additional information about code diff) but using a simple structure. To address the integration of generation and retrieval approaches, it uses the encoder from generation to improve retrieval and a decision algorithm to deal with the outputs of both approaches to decide about the output. They also add noise to some embeddings during training to enhance generalization capabilities of the model
Overall architecture	COME has 4 modules combining retrieval and generation approaches: • Embedding Module - Processes the format (described in preprocessing section) • Translation Module - for the generation approach. RNN Encoder-Decoder with Attention mechanism (Transformer). Differences because learns the context from input after embedding by using a self-supervised task in training, based on Code-T5 model parameters • Retrieval Module - uses Encoder from Translation Module to extract contextual information at same time at it is extracted for this module and recovers the most similar message comparing code snippets. Recovers 10 candidates by maxpooling operation and selects best according to their BLEU score • Decision Module - It uses a combination of an algorithm and a SVM classifier premaded from SKLearn library to select the output from candidates from Translation and Retrieval. (algorithm in page 5, or 796 in ISSTA '23)
Layers	12 layers in Encoder (it takes after RoBERTa's design) and 12 layers in Decoder (takes after T5's Decoder with layer normalization and attention mechanism)
Cells per Layer	Not mentioned, RoBERTa uses special processing unit with attention head and linear transformation, so it is assumed layers of COME uses the same. Mentioned 12 attention heads in code repository.
Weights and Biases	Initialization of weights using Code-T5-base model. No mention for bias but assumed by paper context same initialization from model. Models' weights and biases stored after training. One for each dataset and in MCMD Dataset one per programming language, links to download at https://github.com/hyc2026/come/tree/master
Other Architecture hyperparameters	Layer normalization epsilon = 1e-06 / model type = t5 Specific parameters for self-supervised task in Encoder: Early stopping / length penaly = 2.0 / min-max length = 30-200 / no repeat 3gram / beam width = 4
Loss function	Negative log likelihood

Dataset 1: MCMD	
Dataset Origin	Subset of dataset proposed in paper "On the Evaluation of Commit Message Generation Models: An Experimental Study" that is used in RACE Experiment and filtered by RACE's authors.
Dataset Composition	Contains filtered pairs <diff,message> from dataset proposed in mentioned paper. It contains entries in 5 programming languages (C++, C#, Java, Python and JavaScript) from top 100 projects on GitHub. For each language: C++ - ~200,000 entries C# - ~188,000 entries Java - ~200,000 entries Python - ~257,000 entries JavaSript - ~250,000 entries
Dataset Adjustment/Processing	Original dataset contains entries for each of the 5 programming languages Filter redundant messages such as merge and rollback commits Filter out noisy message as stated in Liu et al. 2018 To balance size data, they retain 450,000 commits for each programming language Authors of RACE filter out commits with multiple files and non-parseable files (.mp3,.jar...)
Dataset Division	Dataset is divided in the 5 programming languages (C++/C#/Java/Python/JavaScript): Training set - 160,948/149,907/ 160,018/206,777/ 197,529 - Total 875,179 (~80%) Validation set - 20,000/18,688/19,825/25,912/24,899 - Total 109,324 (~10%) Test set - 20,141/18,702/20,159/25,837/24,773 - Total 109,612 (~10%)
Dataset available online	Available at link https://github.com/DeepSoftwareAnalytics/CommitMsgEmpirical
Dataset 2: CoDiSum	
Dataset Origin	Dataset taken from CoDiSum experiment, same dataset used for CodiSum used for COME, same division of dataset also maintained
Dataset Composition	The dataset, after preprocessing, contains 90,661 pairs of <diff, message> extracted from preprocessing Jiang and McMillan dataset by the authors of CoDiSum.
Dataset Adjustment/Processing	From Jiang and McMillan dataset: Remove non .java file related commits and remove code of files with no code diffs Use NLP to remove punctuation and special symbols Tokenize diffs and message and remove < 3 words Delete duplicate commits and various modifications in short period of time
Dataset Division	Random division of pairs from dataset into: Training set - 75,000 pairs - (82.7%) Validation set - 8,000 pairs - (8.8%) Test set - 7,661 pairs - (8.5%)
Dataset available online	Dataset available at same link https://github.com/SoftWiser-group/CoDiSum
Technological infrastructure	
Hardware	Dell workstation with Intel Xeon Gold 6130 CPU @2.10GHz with 1 NVIDIA Tesla V100 32GB VRAM GPU
Software	Machine running OS Debian 5.4.143.bsk Installed libgcc-ng 9.1.0 / libffi 3.3 / pip 21.2.2 / python 3.6.13 / openssl 1.1.1s / readline 8.1.2 / setuptools 58.0.4 / sqlite 3.38.5 / tk 8.6.12 / wheel 0.37.1 / xz 5.2.5 / zlib 1.2.12 AI-related libraries installed by pip are huggingface-hub 0.4 / matplotlib = 3.3.4 / nltk 3.6.7 / numpy 1.19.5 / beautifulsopu4 4.11.1 / rouge 1.0.1 / scikit-learn 0.24.2 / scipy 1.5.4 / torch 1.10.0 / tensorboard 2.10.1 / tokenizer 0.10.3 (Rest of versions in environment.yml file in repository)
Time and epochs	15hours aprox. per training session. 2 Training sessions performed. 30 hours total. No number of epochs mentioned
Optimization Hyperparameters	Original vocab. size = 32,100 / AdamW optimizer with initial leraning rate = 5e-05 / Batch size = 12 / Max. Epochs for first training session = 5 / Max. Epochs for second training session = 10 / SVM Algorithm = Radial Basis Function Kernel / Gamma for radial basis function (γ) = 20 / Early stopping enabled / Beam width for self supervised task = 4
Regularization Hyperparameters	Dropout rate = 0.1 Implicit L1/L2 regularization in AdamW optimizer
Training process	The training is performed in two stages. First stage is the self-supervised task in which the Encoder that is going to be used in Translation and Retrieval modules is trained to extract the context representation from the input embeddings. This self-supervised task uses two train sessions (max epochs described in opt. hyperparams). Once the Encoder learnt how to extract the context, the whole system is trained to optimize negative log-likelihood.

General Information			
Name of experiment	CoMeG	Authors	Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu and Hanghang Tong
Year of publication	2022	Available online	Available at link https://github.com/SoftWiser-group/CoMeG.
Publication paper name	Combining Code Context and Fine-grained Code Difference for Commit Message Generation	Notes on availability	N/A

Proposal Information	
Preprocessing process	Aims to obtain more information by extracting two ASTs, the before and the after of a change Only in the training stage, diffs are preprocessed and first sentence of the commit message is extracted, substituting file names and digits by tags and tokenizing the sentences. For the usual code diff preprocessing, only lines containing changes are extracted from diff, then tokenized by whitespaces and punctuation, and end of lines added. Then, context is extracted using tool GumTree. It processes the whole before and after modification files and extract the full code snippet according to predefined rules. After it, GumTree forms the AST and complements its information formatting special names (variable names with type+number for later processing. It outputs two ASTs, one for after and one for before and tuples of <action,position in before AST, position in after AST>. Final AST for the code diff is formed by extrapolating the differences in the tuples which are in non-leaf nodes of the AST to their correspondent leaf nodes, which results in the AST for the code diff.
Input Format	Input is a composition of three elements: - Code diff tokenized - Pair of before and after ASTs - Generated by GumTree - AST of the whole code diff as tuples of <action, position in before AST, position in after AST> - Generated from GumTree output
Input Size	The size of the input is 100 tokens for the diff, 150 for the ASTs for the context (before and after), and 50 for the tuples of the code diff
DNN Architecture	
Approach	CoMeG tries to increase the performance and accuracy of the commit generation by using the existing arquitecture of a Transformer but making fundamental changes to the input format in order to extract the maximum context possible before inputting it to the neural network. It uses pre-defined rules along with tags for the actions to conform the AST. Also, it takes in consideration the before and after situation and the whole code snippet, to try and feed the neural network with lots of information to enhance the resultant commit message. In line with the modifications of the input of the DNN, the Encoder is also modified, incorporating three components: a Diff Encoder, a Context Encoder and an AST diff Encoder in order to compose the overall encoding that will then be sent to the decoder of the Transformer. On the decoder side, a Pointer network is added to add the context information to the typical decoder layers.
Overall architecture	The architecture is a modified RNN Encoder-Decoder with Attention mechanisms (Transformer). Relevant modifications are performed to the Encoder, which is composed by: • Diff encoder - Encodes the tokenized code diff into vectors for the decoder layers • Context encoder (2 of them) - Each with a bidirectional GRU layer to extract semantical and structural information. One of them for the before AST and the other for the after AST. They send the output to the pointer network and the AST Diff Encoder. • AST Diff Encoder - It is fed with the context from Context encoder and the tuples of actions. Encodes vectors with the main changes and send them together with the Diff encoder embeddings to decoder layers. And to the Decoder, which is composed by: • Decoder layers - Typical layers to decode the information of the encoder and generate the commit message • Pointer network - they directly copy from context (before and after ASTs) so the vocabulary size can be reduced
Layers	For the context encoder, diff encoder and decoder number of layers is 3, LSTM layers. For the AST diff encoder and the bi-GRU layers from the context encoder only 1 hidden layer
Cells per Layer	bi-GRU layers contain 512 cells per layer Hidden state for the rest of the DNN is 128 cells No more information about the cell number is mentioned
Weights and Biases	Initialization of weights and biases randomly according to section 4.1.5 of paper No mention of any weights and biases model saved
Other Architecture hyperparameters	Vocabulary size = 300 / Diff vocabulary size = 200 / Attention heads for Transformer = 4 / Layer normalization epsilon = 1e-06 / Max word length = 250 / number of subwords per word = 8 / Size of hidden state vectors = 256 / seed = 42 / Max number of nodes in path = 8 / max length of diff = 100 / max length of context = 150 / max length of AST diff = 50 / max length of message = 20
Loss function	Categorical cross-entropy

Dataset	
Dataset Origin	New dataset created by the authors for the experiment because of the need of before and after version of the code for the input layer. Data collected by the authors, not coming from previous papers. Data comes from top 500 Java GitHub projects by star order
Dataset Composition	Final composition of the dataset contains 160,327 entries of commits. Each entry contains the code diffs, the context information (before and after ASTs) and the AST diffs, as mentioned in preprocessing section
Dataset Adjustment/Processing	Data is collected from top 500 Java projects on GitHub with automatic cloning. Only 480 projects could be cloned Commits extracted using Jgit tool - This also extract changed files and their history (before and after files) Filter out merge and revert commits, non .java associated, commits with < 4 words, and duplicates Lastly, remaining commits are preprocessed to comply with the required format for CoMeG
Dataset Division	Dataset is partitioned by three criteria: **Random, Timestamp** and **Project.** All of them are partitioned in 80-10-10 rule: Training set - 128,263 - (80%) Validation set - 16,032 - (10%) Test set - 16,032 - (10%)
Dataset available online	Dataset available in link https://drive.google.com/file/d/1qnjaltqAzwEeX4ZH0nNGJqVrzIqBTCNX/view
Technological infrastructure	
Hardware	Server with 4 GPUs NVIDIA Tesla T4, they offer 16GB of VRAM each (not mentioned). No additional information mentioned.
Software	No mention on the paper. From code repository: nltk 3.7 / numpy 1.21.2 / pycocoevalcap 1.2 / torch 1.4.0 / tqdm 4.62.3 / transformers 4.6.0
Time and epochs	No mention of the time or epochs used to perform training of the proposal
Optimization Hyperparameters	Mentioned in paper: Epochs = 50 / batch size = 32 / Adam optimizer / Beam size = 1 Other extracted from run.py: evaluation batch size = 8 / weight decay = 0.0 / adam epsilon = 1e-8 / Max. Gradient norm = 1.0 / Training steps to perform = 1 / linear warmup steps = 0
Regularization Hyperparameters	Implicit L1/L2 regularization on Adam optimizer Dropout rate = 0.1
Training process	The training process includes the configuration of the mentioned hyperparameters and limitations to the model and the run of 1 step of trainin, 1 step of validation and 1 test step (according to code). This process is repeated for each of the dataset partitions, obtaining 3 different models, one for each partition. They also use early stoppying mechanism in the training of the three experimetns, which is conducted on same hardware and software

General Information			
Name of experiment	CoRec	Authors	Haoye Wang,Xin Xia,David Lo,Qiang He,Xinyu Wang,John Grundy
Year of publication	2021	Available online	Available at https://zenodo.org/records/3828107
Publication paper name	Context-aware Retrieval-based Deep Commit Message Generation	Notes on availability	Link provided in paper (shorted by tiny.cc) not working, no mention of any other link in the paper for the proposal

Proposal Information	
Preprocessing process	Commit messages (in training) are tokenized according to punctuation and whitespaces. For the code diffs, dataset containes already preprocessed data. From dataset it is only tokenized to be fed to the neural network. It is tokenized using an embedding layer before the encoder input, which represents each word as a fixed length vector, which is later concatenated to obtain the variable length vector which is the input for the encoder. Fixed length vectors are fed one at a time until the end of the sequence of vectors is reached
Input Format	The input format is a vector of variable length (x1...xn) conformed by the concatenation of fixed length vectors for each word from the code diff.
Input Size	Commit message (in training) max length = 30, maximum length of code diff = 100
DNN Architecture	
Approach	The approach of the proposal is to create a hybrid system that combines generation and retrieval approaches to enchance the commit messages, with the inclusion of a new mechanism in training phase to reduce the exposure bias (gap between training and test). To this end, they include a decay sampling mechanism that randomly selects a ground truth sample and assigns it to the decoder, so the model can learn to predict next words also using the previous output of the decoder. With this mechanism, authors aim to enhance the accuracy of the predicted messages by taking in consideratin previously decoded parts of the code diff to influence the prediction of next words. Also, combining the generation and retrieval approach after passing through the attention mechanism, along with the previous output from decoder, the model is able to better predict the probability of generating or copying a word into the final output. If candidates from generation and retrieval are not similar, priority is given to generation approach. Project is implemented using OpenNMT-py framework.
Overall architecture	Architecture is a double RNN Encoder-Decoder architecture with Bahdanau Attention mechanism: one for the generation approach, and the other for the retrieval approach. The generation approach acts as a Tranformer with the additional modification for the Decoder previous alignment, while the retrieval module first encodes the code diff to retrieve from training data using global max-pooling, beam search and cosine similarity approach, to then use the Encoder-Decoder architecture with Bahdanau attention (and the context from previous Decoder alignment) to generate the retrieval candidate. In the end, prediction for the output word for the message is predicted by combining the probability of each approach.
Layers	2 layers of bidirectional LSTM cells for Encoders and 2 layers of LSTM cells for Context-aware Decoders
Cells per Layer	Hidden state size and embedding size is set to 512 cells per layer
Weights and Biases	No mention on the document of initialization of weights and biases and no mention of saved model available after training. According to code, models after training are saved to /models but folder not in the code available, though script to replicate it is available.
Other Architecture hyperparameters	Global Attention mechanism = mlp / max target length = 30 / min target length = 2 / λ (specific hyperparameter for probability equation 14) = 0.8 and 0.5(for two datastets)
Loss function	Maximum Likelihood Estimation (MLE)

Dataset 1: Top 1,000	
Dataset Origin	Taken from Liu et al. Dataset, which is a depurated version of Jiang et al. Dataset of commit messages. This dataset is referred to as top 1,000 in the paper and is the dataset used in NNGen paper, which is a cleansed version of NMT's dataset
Dataset Composition	The dataset contains 27,144 entries of commit code diffs and their respective messages from top 1,000 Java projects ordered by stars on GitHub **Also variation of dataset with noisy data is used in section 7.1 on CoRec**
Dataset Adjustment/Processing	Original Jiang et al. Dataset contained more than 2M commits, to process it: Extracted first sentence of each message, remove commit ID and issue ID from beginning of sentence Remove merge, rollbacks and commit larger than 1Mb. Tokenized messages and diffs without further preprocessing Additionally, Liu et al. in NMT had to filter out diffs longer than 100 and messages longer than 30 Introduced a VDO filter to filter out commits not starting with verb or direct object using CoreNLP Cleansed from bot and trivial messages by NNGen authors
Dataset Division	Dataset divided randomly in: Training set - 22,112- (81,40%) Validation set - 2,511 - (9,25%) Test set - 2,521- (9,35%)
Dataset available online	Available at link https://drive.google.com/drive/folders/1HSQ2HWfIzf886Zh4X9NPAFKV-indpjZ7
Dataset 2: Top 10,000	
Dataset Origin	Created by the authors of the paper due to the alledgeance that top 1,000 dataset only covers 1,75% of commits. Dataset aims to cover up to top 10,000 Java repositories created between Jan 2012 and Dec 2018.
Dataset Composition	Final dataset contains around 107,400 commits pairs from filtering out the contents of top 10,000 Java repositories on GitHub between Jan 2012 and Dec 2018, ordered by descending number of stars. **Also 2 variations of dataset (with noisy data and without VDO filter) are used in section 7.1 on CoRec**, but esentialy dataset is still the same
Dataset Adjustment/Processing	Commits recovered from top 10,000 Java projects by star order • For messages, non-English commits are removed, remaining commits lowercased and parsed with NLTK Punkt tool Only first sentence is extracted from commit message From messages, brackets removed, commit and issue IDs replaced by placeholders Merge, rollback are filtered out. Noisy commits are filtered out and VDO filter applied - using Stanford CoreNLP Messages tokenized by whitespaces and punctuation, length > 30 are filtered out • For code diffs, diff header deleted and code diff lowercased. Commit ID replaced by placeholders Commits larger than 1 Mb removed, second check of merge and rollback commits removed Length > 100 are filtered out. Duplicated commits are removed Keywords + patterns used to filter out noisy data
Dataset Division	Dataset is ensured to be disjoint and divided into: Training set - 96,704 - (90%) Validation set - 5,372 - (5%) Test set - 5,372 - (5%)
Dataset available online	Dataset available at link https://zenodo.org/records/3828107 - Link provided in paper not accesible
Technological infrastructure	
Hardware	Server with 12 cores of a 3.6GHz CPU, 64GB of RAM memory and a GPU NVIDIA GTX 1080 with 8GB of VRAM
Software	Server running Ubuntu 16.04 with environment dependencies (extracted from code repository): python >= 3.5 / pytorch 0.4.1 / torchtext 0.3.1 / ConfigArgParse 0.15.2 / nltk 3.4.5 / numpy 1.14.2
Time and epochs	On top 1,000 dataset - Training took 2 hours and 57 minutes On top 10,000 dataset - Training took 13 hours and 5 minutes No mention of epochs used in training
Optimization Hyperparameters	Batch size = 64 (32 appears in paper but not in code) / Adam Optimizer / initial learning rate = 0.001 / Max number of epochs for top 1,000 = 100,000 / Max number of epochs for top 10,000 = 400,000 / Beam size = 5
Regularization Hyperparameters	Implicit L1/L2 optimization in Adam Optimizer Dropout rate = 0.1
Training process	To train the proposal, they set the hyperparameters for each dataset (top 1,000 and top 10,000) and train the model using a python script with the set of parameters for each situation (some of them are different, such as training epochs). During training phase, Decay sampling mechanism is used to simulate the previous output of decoders so the model learns how to predict also using informatio from previous decoder alignment. Dropout rate is used to avoid overfitting. Also, when training is finished, weight of previous decoder input is reduced to enchance the capabilities of the trained model.

General Information			
Name of experiment	CoreGen	Authors	Lun Yiu Nie, Cuiyun Gao, Zhicong Zhong, Wai Lam, Yang Liu, Zenglin Xu
Year of publication	2021	Available online	Available at link https://github.com/Flitternie/CoreGen
Publication paper name	CoreGen: Contextualized Code Representation Learning for Commit Message Generation	Notes on availability	Accesible at link, link provided in paper

Proposal Information	
Preprocessing process	CoreGen classifies and deals with commits on 2 categories: Explicit code changes and implicit binary file changes For explicit code changes, they are detected because they are marked with special tokens, commit sequence is extracted and before and after versions of the commit are extracted. For implicit changes in binary files, a mask process of a random fragment of the file is extracted to learn the contextual information. For this, sequence of lines in the file is divided using token <nl> and longest line is randomly masked for a fragment, which is the input of the system
Input Format	Not explicitly mentioned in the paper. From code pretrain.py, mentioned elements from explicit code changes and implicit changes in binary files are represented as indexes contained in dictionary structures, and tho overall input of the network is a vocabulary file and the dictionaries of indexes pointing to that vocabulary.
Input Size	Input dimension of the tokens for each input is set to 512 tokens
DNN Architecture	
Approach	Approach proposed tries to mitigate 3 problems from previous approaches: lack of contextual information, no attending explicitly to the changes but to the whole snippet, and using RNNs with long-term dependency issues. To mitigate that, CoreGen approaches the problem of commit message generation using a two phase approach, which in the first phase is trained to learn and predict code representartions and classify commits on categories explained in preprocessing, and in second stages the learnt parameters are transfered to the final Transformer and fine-tuned with backpropagation to all layers. By separating learning the contextual information and then fine-tuning the Transformer, authors aim to enhance existing approaches, but also to provide a flexible mechanism that can be incorporated to other approaches to enchance their performance (referring especially to Stage I)
Overall architecture	The model uses a Transformer, a variation with a self-attention mechanism of a RNN Encoder-Decoder architecture. This Transformer, nevertheless, is trained in different ways depending of the stage: In stage I, the commits are classified into explicit changes and implicit changes to binary files and the Transformer is trained in tasks to learn the contextualization of these two categories In stage II, parameters are initialized with the previous learned values and fined tuned by using backpropagation
Layers	Both Encoder and Decoder of Transformer are composed of 2 layers of self-attention cells
Cells per Layer	Not mentioned in paper, from code, embedding layer dimension is 512 while hidden layer default value is 2,048
Weights and Biases	Input embedding for code tokens randomly initialized in stage I based on seed (from code). In stage II initialization is to parameters learnt from stage I Final trained model is set to be stored in the code but not contained in code repository or not available
Other Architecture hyperparameters	Vocabulary size = 55,732 / Multi-attention heads per layer = 6 / Mask rate for Equation 4 = 0.5 /
Loss function	In stage I when learning code representations - Log-likelihood both for explicit and implicit changes In stage II - Negative Log-likelihood

Dataset	
Dataset Origin	Dataset is extracted from NNGen original paper, and is the cleansed version made by Liu et al. Of the Jiang et al. Dataset of top 1,000 Java projects on GitHub
Dataset Composition	Cleansed 27,144 pairs of <diff,message> from top 1,000 Java projects in order of descending stars. **Also dataset variation cleaning ~15% from validation set and ~15% from test set for data deduplication analysis**, but esentially same dataset.
Dataset Adjustment/Processing	Take original dataset from NMT Detect bot-generated messages according to liferay-continuous-integration pattern - around 13% in each set Detect trivial messages according to trivial patterns proposed by the authors - around 3% in each set Remove detected messages from original dataset
Dataset Division	Dataset divided randomly in: Training set - 22,112- (81,40%) Validation set - 2,511 - (9,25%) Test set - 2,521- (9,35%)
Dataset available online	Dataset available in online annex at link https://mycuhk-my.sharepoint.com/personal/1155079751_link_cuhk_edu_hk/_layouts/15/onedrive.aspx?id=%2Fpersonal%2F1155079751%5Flink%5Fcuhk%5Fedu%5Fhk%2FDocuments%2Fdata%2Ezip&parent=%2Fpersonal%2F1155079751%5Flink%5Fcuhk%5Fedu%5Fhk%2FDocuments&ga=1
Technological infrastructure	
Hardware	No mention of hardware used in the paper.
Software	No mention of software used in the paper. From code repository, extracted: Conda environment running python 3.6.13 with pip 21.0.1 - created with miniconda3 cudatoolkit 10.2.89 / libgcc 9.1.0 / mkl 2020.2 / ncurses 6.2 / ninja 1.10.2 / numpy 1.19.2 / pytorch 1.5.0 with dependencies for CUDA 10.2 and CUDNN 7.6.5 / setuptools 52.0.0 / sqlite 3.35.4 / torchvision 0.6.0 / tk 8.6.10 / wheel 0.36.2 / zlib 1.2.11 Rest of verions in environment.yml file
Time and epochs	No mention of time or epochs needed to train the model
Optimization Hyperparameters	Batch size for stage I = 32 (from code) / batch size for stage II = 64 / Max epochs for stage 1 = 30 / Max epochs for stage II = 100 / Adam optimizer / warm-up steps = 4000 Defaults from code (not specified in paper): seed = 42
Regularization Hyperparameters	Implicit L1/L2 reularization in Adam optimizer Dropout rate = 0.1 (default from code)
Training process	Training process consists on the two mentioned stages. First stage with batch size of 32 and 30 epochs is used to train the transformers in the prediction of explicit code changes and implicit binary file changes to generate the messages from randomly initialized code tokens. In the second stage the Transformer is initialized with the trained weights from stage I and it is fined tuned with batch size 64 and 100 epochs using backpropagation on all layers. The hyperparameters used and previously mentioned are said to be tuned on the validation set by the authors.

General Information			
Name of experiment	FIRA	Authors	Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li,Wenjie Zhang, Dan Hao
Year of publication	2022	Available online	Available at link https://github.com/DJjjjhao/FIRA-ICSE
Publication paper name	FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation	Notes on availability	Link provided in paper, badges from ACM for experiment availability in paper

Proposal Information	
Preprocessing process	Preprocessing processess the code diff in order to extract a final grapth composed of four phases: First phase extracts two ASTs (before and after) only for each of the hunks of the file (changed lines), the initial graph (graph AST) is formed by the two chopped ASTs from the hunk (before and after). This ASTs are extracted by using GumTree tool Then, for integral token nodes (i.e. variable names), subtokens are extracted according to common conceptions (camel case and snake case) and subtokens are connected to the integral token nodes of graph AST, forming graph TOKEN After it, for all nodes, the different edit operations performed to get to after AST from before AST are marked in five categories [added, deleted, moved, updated and match(unchanged)]. These edit nodes connect corresponding nodes from before and after chopped ASTs in graph TOKEN, this new graph is called graph EDITION Finally, as sequential structure also contains contextual meaning, a graph containing the sequential structure is merged with graph EDITION connecting nodes in a sequence, composing graph FINAL. To get to the DNN, graph is structured by an embedding layer which changes the format from graph to lookup table and vectors No mention of preprocessing for messages in training phase, assumed tokenized by whitespaces and punctuation as authors use Transformer architectue in decoder
Input Format	The input format of the graph contains two elements after going through the embedding layer: -A lookup table with the actual information of code nodes and edit nodes -An embedding vector composed by concatenation of code nodes followed by edit nodes from the graph FINAL as references to the lookup table
Input Size	The max. Input size for the graph in that format is 650 nodes, being 370 code nodes and 280 edit nodes. For the commit message in the training phase, 30 length as max. Length.
DNN Architecture	
Approach	The approach of FIRA is to solve some of the problems that existent approaches present when generating commit messages, which are that they cannot correctly detect subtle edit operations in large files, and they cannot copy subtokens from out of vocabulary words to enchance commit message meaning. To solve this, FIRA incorporates two main mechanisms: a new input representation based in graphs (along with a convolutional variation of an Encoder to interpret them) and a novel dual copying mechanism that allow to copy subtokens from input. To match it with the architecture of the DNN, FIRA also adapts the Encoder to a Graph variant of a convolutional system, and uses a Transformer decoder as it shows some advantages in previous approaches.
Overall architecture	FIRA uses a novel variation of an RNN Encoder-Decoder architecture, where the Encoder is part of a CNN while the Decoder is taken from a Transformer with self-attention architecture: Encoder is a Graph Convolution Network (GNC) Encoder that leverages the ChebNet approach to aggregate the neighbors' information to the encodings. It applies symetric normalization and normalized adjacency matrix to avoid gradient explosion. Decoder is taken from a Transformer architecture, using self-attention cells with multi-attention heads to iteratively decide which token to generate next from the vocabulary using a softmax layer. Also, decoder incorporates the dual copy mechanism and joins to the vocabulary tokens the subtokens associated to the nodes from the input, so the probability can also decide that the next token is copied from input subtokens.
Layers	6 layers of RePU (Rectified Power Unit) cells for GNC Encoder, 6 layers of self-attention cells for Transformer Decoder
Cells per Layer	Number of cells in GNC Encode not mentioned or found in code. For Transformer decoder, hidden size dimension is 256 cells per layer.
Weights and Biases	No mention in paper of weights and bias initialization method. From code of run_model.py, weights initialized using Xavier normal initialization, no mentioned bias initialization in code (probably default zeros) No mention of model being saved after training and not in code repository
Other Architecture hyperparameters	All previously mentioned in preprocessing max. yength and layers/cells per layer + attention heads number = 8 From code: seed (default) = 0
Loss function	Cross-Entropy

General Information			
Name of experiment	RACE	Authors	Ensheng Shi, Yanlin Wang, Wei Tao, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, Hongbin Sun
Year of publication	2022	Available online	Available at https://github.com/DeepSoftwareAnalytics/RACE
Publication paper name	RACE: Retrieval-Augmented Commit Message Generation	Notes on availability	Code accesible and link provided in paper

Proposal Information	
Preprocessing process	RACE has two modules, input is fed to the first module, the Retriever module, for it to retrieve the most similar commit from the training corpus via cosine similarity. To this end, authors first train a RNN Encoder-Decoder based on the proposed model for NMT, to then use the encoder to learn the semantics of the code diff that acts as an input and retrieve from the encoded vector rather than the preprocessed input. Input to train the RNN and then passed to the trained encoder as the input of the system is said to be from the dataset containing pairs of <diff, message> and to be made according to citations for NMT paper and commitBERT. As explained in section 4.2 of the paper, token-level action tokens are added to the input to represent the code diff with more information. There are 9 special tokens which are [keep, keep_end, insert, insert_end, delete, delete_end, replace_old, replace_new and replace_end]. To extract and add this tokens to the input, difflib is used. Then, the second module. the Generation Module (which is the core of the system), receives the encoding of the input code diff, the encoding of the most similar entry from the training corpus and the paired message to the most similar code diff encoding. This could be considered the "second input" of the model
Input Format	For the Retrieval Moule, data input is assumed to be a composition of a token-level action tag and tokens encoded. No further mention of the format. The input format for the Generation model is two encoded vectors and a tokenized commit message
Input Size	For the input layer of the Retrieval module in the training phase is 200 tokens for code diff and 30 (from code, 50 specified in paper) for message.
DNN Architecture	
Approach	The approach is based on a two module design that deviates from the traditional approach of previous proposals to deal with the commit message generation. According to authors, the idea is to retrieve and agument the retrieval to obtain a well-formed commit message. To this end, they include a Retrieval module which is expected to retrieve the most similar result from the training corpus. Opposite to previous approaches, the retrieved entry is not used as the candidate for the output of the system, but passed to a Generation module that uses the most similar commit diff and message, along with the original code diff, to generate the final output based on the reference from the most similar entry but incorporating characteristics from the original code diff. This task from the Generation module is performed by a novel component that the authors name as exemplar guider, which acts between the Encoder and the Decoder of a Transformer architecture.
Overall architecture	Architecture of the proposal is a 2 module design, containing: • Retrieval module - first trained as a RNN Encoder-Decoder based on NMT architecture and initialized according to CommitBERT paper, which is trained on a large dataset of commits to obtained a trained Encoder. Once this is done, the Encoder encodes the input (a code diff) and passes it to the retriever component, which retrieves using cosine similarity the most similar k entries from the training corpus. Then the entries are compared according to cosine similarity and the first-ranked is sent to the Generation module, along with encoding of original input and the paired commit message. • Generation module - this module has three encoders from a Transformer architecture, one for each of the inputs of the module, initialized on the weights of CodeT5-base (as well as the decoder). They encode the inputs and the encoding from original input an most similar entry are sent to the exemplar guider component, which is first learns the semantic similarity and then incorporates information of the similar commit to the original encoding according to Equation 4. Result weights, along with encoding of commit message, is used to obtain the final encoding of the diff, which is passed to a Transformer Decoder to use MLP+softmax operator to find the final probability of the tokens of the vocabulary to be in the final output.
Layers	Number of layers not explicitly mentioned. For encoder based on NMT and CommitBERT, assumed 12 layers as in CommitBERT (as it is said to be initialized as it). For the Encoders and Decoders based on CodeT5-base, also 12 layers are assumed to be use since in the T5 paper (Raffel et al.) it is said to also be initialized in BERT with this number of layers for encoder and decoder
Cells per Layer	Not mentioned in the paper. According to citations to CommitBERT and CodeT5-base, 768 cells for the hidden layers and 3072 for the output layer (from T5 paper)
Weights and Biases	Initialization of Encoder for Retrieval module based on weights of CommitBERT Initialization of diff Encoders and Decoder for Generation module based on CodeT5-base model Models are not mentioned to be saved after training and they are not available at code repository
Other Architecture hyperparameters	Most hyperparameter configuration not mentioned in the paper and referred to citations of CodeT5-base, NMT and CommitBERT papers. For the hyperparameters mentioned in the paper: Vocabulary size = 32,109 (original CodeT5-base+9 action tags) / seed = 0-1-2 (run 3 times)
Loss function	Cross-Entropy both for Retrieval module and Generation module

Dataset	
Dataset Origin	Filtered version of large dataset called MCMD proposed by Tao et al. ICSME '21.
Dataset Composition	Contains filtered pairs <diff,message> from dataset proposed in mentioned paper. It contains entries in 5 programming languages (C++, C#, Java, Python and JavaScript) from top 100 projects on GitHub. For each language:
Dataset Adjustment/Processing	Original dataset contains entries for each of the 5 programming languages Filter redundant messages such as merge and rollback commits Filter out noisy message as stated in Liu et al. 2018 To balance size data, they retain 450,000 commits for each programming language Authors of RACE filter out commits with multiple files and non-parseable files (.mp3,.jar...)
Dataset Division	Dataset is divided in the 5 programming languages (C++/C#/Java/Python/JavaScript): Training set - 160,948/149,907/ 160,018/206,777/ 197,529 - Total 875,179 (~80%) Validation set - 20,000/18,688/19,825/25,912/24,899 - Total 109,324 (~10%) Test set - 20,141/18,702/20,159/25,837/24,773 - Total 109,612 (~10%)
Dataset available online	Available at link https://zenodo.org/records/7196966#.Y0juJHZBxmM
Technological infrastructure	
Hardware	Server with 4 GPUs NVIDIA Tesla V100. No more hardware mentioned
Software	No mention in paper of software versions. From code repository extracted Conda environment with Python 3.6 torch 1.10 / transformers 4.12.5 / tdqm 4.64.1 / prettytable 2.5.0 / gdown 4.5.1 / more-itertools 8.14.0 / tensorboardX 2.5.1 / setuptools 59.5.0 / tensorboard 2.10.1
Time and epochs	No mention of final epochs but around 1.2 hours for each epoch of training (max epochs set to 20 but 10 specified on code)
Optimization Hyperparameters	AdamW Optimizer (both in Retrieval and Generation modules) / Initial learning rate = 2e-5 / Batch size = 32 / Max. epochs = 20 / Beam size = 10 Rest of hyperparameters referred to CodeT5-base
Regularization Hyperparameters	Implicit L1/L2 regularization on AdamW optimizer Rest of potential regularizations referred to the ones applied in CodeT5-base
Training process	Training process consists on various stages. On first place. Whole RNN initialized on CommitBERT for the Retrieval module is created and trained in a large corpus of around 0.9 M pairs of <diff.message> from the original version of Tao et al. ICSME'21 dataset, which then is filtered to train the whole system. When encoder has learnt to extract code contextualization, it is used in the training of the Generation module. For the training of the Generation module, the inputs from the dataset are fed to the Retrieval module, which recovers the most similar entry from previous training corpus, and sends the three inputs (original diff, most similar diff and paired message) to the Generation module, Transformer-based Encoders and Decoder initialized on CodeT5-base, as well as Exemplar Guider, are trained on the main dataset encoded by the already trained Encoder for the Retrieval module. Generation model also initialized on mentioned hyperparameters if they do not match CodeT5-base hyperparameters

General Information			
Name of experiment	ReGenSD	Authors	Zhihan Li, Yi Cheng, Haiyang Yang, Li Kuang, Lingyan Zhang
Year of publication	2022	Available online	Not available at link provided https://anonymous.4open.science/r/regensd-61E5/
Publication paper name	Retrieve-Guided Commit Message Generation with Semantic Similarity And Disparity	Notes on availability	Anonymous 4open science site shows expired repository when trying to access to the source code of the experiment

Proposal Information	
Preprocessing process	Approach has 3 modules, first input to the system is Retrieval module, but actual DNN contained in last module, Guide module. The preprocessing from the dataset for the retrieval module consists on same preprocessing from NNGen reused to form the "bag of words" structure fed to this retrieval module. For the actual input that the DNN receives in the Guide module, it depends wether it is a simple generation or a retrieval-based one. If its a simple generation, only token sequence for original code diff is received by Encoder of Guide module. If its a retrieval-based, it is a composition of 3 elements received as a token sequence derived from this "bag of words": the one from the original code diff, the one for the most similar retrieved diff and the paired message for the most similar retrieved diff.
Input Format	The input format is a "bag of words", structure taken from NNGen that correspond to a vector of the word tokens where order and grammatical structure is not considered, but term frequency is kept
Input Size	For Retrieval module, input size not specified as NNGen is reused and it can handle different sizes since is not a DNN and does not depend on an input layer For Guide module, as settings are referred to citation 19, embedding size of system should be 100 For decoder input (reference message) in training, length set to 15
DNN Architecture	
Approach	The approach is the first approach that tries to change the methodology and incorporate the generation approach based on the retrieval results instead of running both approaches in paralell. To this end, it first uses NNGen structure to retrieve the most similar code diff, but also takes in consideration its related commit message, which previous approaches omitted. Then, it incorporates a novel approach as a Selection module that decides wether to use a retrieval-based generation or a simple one. Lastly, the DNN used is based on previous works' architecture to generate the final output. Nevertheless, according to the selection module, the number of components and the amount of information changes, using (or not) the previously-retrieved information.
Overall architecture	The approach uses 3 modules: • Retrieval module - taken from NNGen, it uses same algorithm and information representation, it is fed the training set as the retrieval corpus to obtain the most similar entry. For it, it uses cosine similarity to recover the K best approaches and decides the one with the highest BLEU score • Selection module - It receives as "bags of words" the retrieved diff and message, as well as the original, and, based on on a threshold named α and the n-gram precision for 1 to 4 n-grams it decides wether to perform a simple generation of the output or use the information from retrieval to guide the generation of the output • Guide module - is the DNN component. Its architecture follow a RNN Encoder-Decoder with Bahdanau Attention mechanism based on bi-LTSM layers. It uses a different number of Encoders depending on if its a simple (1 encoder) or a retrieval-based (3 encoders) generation. Followed by a decoder using Attention mechanism. When generation is retrieval-based, two additional mechanisms are included: Relation gates to check the semantic relevance between the encoding of original and retrieval diff and message, producing the difference vector, which is used in the attention mechanism to enchance the weight of differences for the final output. Decoder also uses beam search to generate better messages.
Layers	No mention on paper or suggestet citation 19 of the number of layers of the Bi-LTSM architecture
Cells per Layer	The dimension of cells per layer in the Bi-LTSM architecture is set to 256 according to paper
Weights and Biases	No mention of initialization of weights and biases in paper of citation 19. No available information as repository is not accesible. No mention on paper of saved weights and biases model after training
Other Architecture hyperparameters	OOV Token = <UNK> / α(Selection module) = 0.4 / K(number of retrieval entries) = 5 No additional mention of not-already-presented hyperparameters. Cannot extract more since repository is not accesible
Loss function	Conditional Cross-Entropy

Dataset	
Dataset Origin	Dataset used is well-known cleansed dataset proposed in NNGen paper from Liu et al., which is a cleansed version of Jiang and McMillan dataset
Dataset Composition	Cleansed 27,144 pairs of <diff,message> from top 1,000 Java projects in order of descending stars.
Dataset Adjustment/Processing	Take original dataset from NMT Detect bot-generated messages according to liferay-continuous-integration pattern - around 13% in each set Detect trivial messages according to trivial patterns proposed by the authors - around 3% in each set Remove detected messages from original dataset
Dataset Division	Dataset divided randomly in: Training set - 22,112- (81,40%) Validation set - 2,511 - (9,25%) Test set - 2,521- (9,35%)
Dataset available online	Dataset available in online annex of NNGen paper at link https://drive.google.com/drive/folders/1HSQ2HWfIzf886Zh4X9NPAFKV-indpjZ7
Technological infrastructure	
Hardware	Computer with NVIDIA GeForce RTX 2080 Ti with 11 GB of VRAM. Not further mention of hardware
Software	Running on Ubuntu 16.04. Uses Tensorflow framerowk. No additional information can be extracted since repository is not accesible
Time and epochs	No mention of time or epochs used in training
Optimization Hyperparameters	Stochastic Gradient Descent optimization alg. / Initial learning rate = 0.5 / Batch size = 256 Rest of hyperparameters reference to citation 19: Decaying rate = 0.95 every epoch / Beam size = 5 / Training epochs = 20
Regularization Hyperparameters	No regularization techniques mentioned in paper. From citation 19: Dropout rate = 0.2 / Gradient norm clip = 5
Training process	Training process consists on feeding the training set to the Retrieval module, which acts like NNGen retrieving for each entry the most similar entry in the retrieval corpus (which is the training set). The training parameters are set to the decided values and the Guide module is trained on the training set, validated and tested. During training, Out of Vocabulary words are substituted by the token UNK.

General Information			
Name of experiment	CommitGen	Authors	Pablo Loyola, Edison Marrese-Taylor and Yutaka Matsuo
Year of publication	2017	Available online	Available at link https://github.com/epochx/commitgen
Publication paper name	A Neural Architecture for Generating Natural Language Descriptions from Source Code Changes	Notes on availability	Experiment available at link, link provided in paper

Proposal Information	
Preprocessing process	Data preprocessing from dataset content consists on using a script (preprocessing.py) to parse and filter commit messages contained in dataset. For the commit message used in the training on the decoder, it is tokenized using Penn Treebank tokenizer based on punctuation and text marks. For the code diff, it first parses the modified lines and then uses a lexer tokenizer to obtain the representation of the diff lines using a per-line fashion. For each sentence, tokens with less frequency than 3 aparisons in vocabulary are substituted by UNK token, while the beginning and end of lines are marked by START and END tokens
Input Format	Stream of tokens both for code diff and message
Input Size	Input size for code diff is 100 tokens and for commit message is 20 tokens
DNN Architecture	
Approach	The main approach consists on leveraging the advantages of Encoder-Decoder neural networks to help in the task of commit message generation as the problem nature allows to recover code changes (changed lines) and there is enough data of paired commit messages. Main objective is to achieve a model than can automatically generate semantically sound descriptions of the code changes
Overall architecture	The model is a RNN Encoder-Decoder architecture with Luong Attention mechanism
Layers	No mention in the paper. From code 400 cells for encoder and decoder size
Cells per Layer	No mention of number of cells per layer, library functions used to instantiate the encoder layer, also decoder layer is instantiated as 1 layer of regularized-by-dropout LSTMcells.
Weights and Biases	No mention in paper. From code initialization is defaulted to torch version settings. According to paper, model saved each epoch after validation score and final best model saved
Other Architecture hyperparameters	seed = 112300 / Attention mechanism = Luong / Use beam search No additional mention on paper of architecture-related hyperparameters
Loss function	Cross-Entropy Loss (extracted from code MaskedLoss.lua)

Dataset	
Dataset Origin	Dataset proposed by the authors for the study. Collected by the authors to test the effectiveness of the presented model
Dataset Composition	Dataset contains data from 12 open source popular projects on GitHub, 3 for each of the following languages: python, java, javascript and C++. Data contains diffs and metadata from commit history of these projects. No further specification of content presented in paper Also **variation called "atomic" containing only diffs that modify a single line in project**
Dataset Adjustment/Processing	Selection of 3 well-kown open-source projects for each language: java (CoreNLP, elasticsearch, guava), python (Theano, keras, youtube-dl), javascript (node, angular, react) and C++ (opencv, CNTK, bitcoin) Extraction of commit history from projects. Downloaded diffs and metadata (when atomic commit, dataset filtered over 1 line change commits)
Dataset Division	No mention of dataset division on paper
Dataset available online	Available at link https://osf.io/67kyc/?view_only=ad588fe5d1a14dd795553fb4951b5bf9
Technological infrastructure	
Hardware	No mention of harware used in paper
Software	No mention of software used on paper. On code repository, mentioned needed packages are Torch, Cutorch, unidiff and pygments. No version mentioned
Time and epochs	No mention of time and epochs used in training
Optimization Hyperparameters	Beam size = 10 / Max gradient normalization = 5 (from code) / Decay rate = 0.8 (from code) No mention of optimization algorithm used
Regularization Hyperparameters	Dropout-regularizated LSTM Decoder / Dropout at embeddings and NL layer / Dropout rate = 0.5 (from code)
Training process	Training process consists of preprocessing the data and feeding the training set to the model (no specification of which is the training set). After it, validation set is used to get best model by saving after each epoch and evaluating on METEOR. Testing performed on BLEU-4. Beam search used to improve output of model

General Information			
Name of experiment	Mucha	Authors	Chuangwei Wang, Yifan Wu, and Xiaofang Zhang
Year of publication	2023	Available online	Link provided https://github.com/cmgads/Mucha - Not working
Publication paper name	Mucha: Multi-channel based Code Change Representation Learning for Commit Message Generation	Notes on availability	Link provided show error 404 not found when trying to access. Not possible to find repository from online search

Proposal Information	
Preprocessing process	Given a code diff, the following preprocessing is performed: Code is split into code before changes and after changes using python library difflib. This library is also used to perform line alignment and record the alignment information. Lines are marked as [KEEP], [ADD] or [DEL] if they are mantained, added or deleted between the two versions. Token is added at beginning of line, also, lines that changed are marked with a 1, while unchanged remain with a 0. For the lines marked as changed, use difflib to perform token alignment and mark tokens with 1 flag if they have changed or 0 if not. Finally, GumTree tool is used to extract ASTs from before and after, and then same tool used to compare ASTs' nodes for differences. The comparison result in one of five cathegories (Match, Insert, Delete, Move, and Update). Input for the embedding layer consists on three elements: the code change composition from marked and flagged before and after versions, and the marked before and after ASTs. All three are tokenized using tokenizer in pre-trained model used to extract context embeddings. In this case, tokenization from UniXcoder (specific tokenization function provided in paper for UniXcoder)
Input Format	Input format is tokenization of code changes and the two marked ASTs for before and after code according to UniXcoder tokenization function (given UniXcoder in paper)
Input Size	Input size is 512 tokens according to paper and performance study on different lengths (section 5.3)
DNN Architecture	
Approach	The general approach of the proposal is to use multiple-channel mechanism, in combination with querying back technique, to better build the semantics and syntax information of the code given. Also, the approach initializes the contextual embedding (encoders) of the model using a pre-trained model for code tasks to boost the performance. This way, approach tries to avoid limitations of not using the full contextual information offered by the code diff in the message generation. Also, input representation is enriched by extracting three levels of granularity (Line,Token and AST levels) information, which is all then processed by the multi-channel and the query back mechanisms to generate more precise commit messages
Overall architecture	The architecture presented for the model is based on Huggingface's Transformer architecture package for python, using self attention cells in an Encoder-Decoder architecture. Regarding this, the model includes: • Two Encoders, one for code changes and the other for the ASTs (which is a merged Encoder that can deal both with the before and after ASTs since their format is the same). This Encoders produce 3 different encoded vectors of information. Two of these vectors come from the encoder for the code changes and explore the line and token level, while the AST encoder outputs one encoded vector with the information of both ASTs. The encoder of the ASTs use mask attention mechanisms to enchance the code representation for structures like ASTs • Three multi-channels to decode the encoded information in the three levels (line, token and AST) and sends the output to an attention mechanism. Both line and token level channels use a combination of query back and mechanism and the encoding of contextual changes, while the ASTs' channel, after using it, linearly projects the result into a feature space, use layer normalization and merge them to obtain the joint representation of both ASTs • An attention mechanism that modifies the weigths of the encoded vectors and sends the encoding to the decoder component • A Decoder component based on python Transformer package that decodes and generates the final output of the system
Layers	Layers for both Encoder and Decoder components are 12 layers per component, using self-attention cells
Cells per Layer	Not mentioned in paper. Transformer self-attention heads are configured differently in each implementation of the HuggingFace's Transformer package. Since code repository is not available, not possible to determine
Weights and Biases	Initialization of weights and biases using UniXcoder model Not mentioned in paper if final model is stored after training. Since code repository is not available, not possible to check if model is available online.
Other Architecture hyperparameters	No additional parameters regarding architecture mentioned in paper
Loss function	No mention of loss function in the paper

Dataset	
Dataset Origin	Dataset adapted by authors from Tufano et al. Dataset at method level, as proposal also need some of the characteristics in the dataset to extract the desired level of granularity. Data in Tufano et al's dataset comes from popular Java projects on GitHub, as mentioned in the paper
Dataset Composition	Original dataset from Tufano et al. Contains 167,000 triplets of <method submitted for review, reviewer comment, revised mention of method>. Authors deal with the triplet format as <code before, change description (commit message), code after>. From the filtering process of the dataset a 3.2% of the dataset is discarded, leaving 161,656 entries in the dataset
Dataset Adjustment/Processing	Dataset is extracted from Tufano et al., which is already preprocessed by adding <START>,<END> and <technical_language> for the codes and the message. Using regular expressions, content is converted to final format of tags and flags defined in preprocessing section A script is then applied to the dataset to validate the format transformation and discard the data that cannot be built into an AST
Dataset Division	No mention in paper about the dataset division
Dataset available online	Dataset said to be available but not available since link provided is not working
Technological infrastructure	
Hardware	Server with 24 cores at 3.8 GHz CPU and a NVIDIA RTX 3090 GPU
Software	HuggingFace's Transformer python package used and PyTorch deep learning framework used. No further mention of software used. Since repository is not accesible it is not possible to extract more specifications for software
Time and epochs	No mention of time or epochs used in training
Optimization Hyperparameters	Adam optimizer / Initial learning rate = 5e-05 / Linear warmup
Regularization Hyperparameters	L2 regularization with coefficient λ = 10e-05 Usually implicit L1/L2 regularization in Adam optimizer, but since L2 regularization is manually set and code not available, cannot assume.
Training process	Training process consists on initializing the model using UniXcoder's weights and biases, set the mentioned hyperparameters and feed the GumTree's preprocessed entries of the dataset to the model. No further mention of additional configurations for the training in the paper

General Information			
Name of experiment	CCT5	Authors	Bo Lin, Shangwen Wang, Zhongxin Liu, Yepang Liu, Xin Xia, Xiaoguang Mao
Year of publication	2023	Available online	Available at link https://github.com/Ringbo/CCT5
Publication paper name	CCT5: A Code-Change-Oriented Pre-trained Model	Notes on availability	Code available, link provided in paper

Proposal Information	
Preprocessing process	Data entries in dataset used for training consist on pair <diff, message> Pairs are taken from dataset and passed to tokenizer - Tokenizer taken from CodeT5 model [CLS] token is added at the beginning, followed by code changes with tokens [ADD], [DEL] or [KEEP] in front of tokens of each line, followed by the tokenized message with the token [MSG] added at the beginning to separate code and message tokens
Input Format	Vector of tokens with format [[CLS], [ADD/DEL/KEEP]cc1,cc2...[MSG],msg1,msg2...]
Input Size	Not mentioned in paper as it changes for each task. Extracted for commit generation task from script finetune_msggen.sh. Input max length for code diff = 512 and for commit message = 128
DNN Architecture	
Approach	The approach is based on pre-training a popular baseline model, named T5, specifically the version used in initialization and trained for code-related tasks, CodeT5, to fine-tune over a large dataset of code changes information for different tasks related to code changes treatment, amongst which commit message generation is included. The main assumption consists on improving the results on the tasks overcoming the limitations of single-purpose models on one side, and the limitations caused by the use of small training corpus on the available baseline approaches. For this, CCT5 is trained in five different code changes-related tasks (Masked Lang. for code changes, Masked Lang. for commit message, Code diff generation, Generation from NL to code and Generation from code to NL) to acquire generalization capabilities in a large dataset to outperform state-of-the-art approaches in each task.
Overall architecture	Arquitecture is taken from T5 model, uses a RNN Encoder-Decoder architecture with self-attention mechanisms, often referred to as Transformer. The model architecture uses attention heads to perform multi-head attention on the layers of both Encoder and Decoder
Layers	Both Encoder and Decoder, following model T5, have 12 layers of multi-attention calculation cells
Cells per Layer	No specific cells type, mechanisms of cells used are attention head mechanisms. 12 attention heads per layer in Encoder and Decoder
Weights and Biases	Initialization of weights and biases takes after the code variation of model T5, named CodeT5, cited in paper Model stored after training not mentioned in paper, in code repository, model stored file not found but reference in script to https://www.zenodo.org/record/7964370 found, where models are hosted
Other Architecture hyperparameters	Parameters size = 220M Other parameters extracted from code repository for commit message generation: tokenizer = Salesforce/codet5-base / Gradient accumulation steps = 4 / Save model flag ON but saved model not available
Loss function	Loss function is different for each fine-tuning task. For commit message generation (called code to NL generation), loss function is Negative Log-likelihood

Dataset 1: CodeChangeNet	
Dataset Origin	Dataset created for the training purposes of CCT5 model by the authors and presented in the paper. Dataset is large in data volume to, according to the authors, solve the issues derived from small training datasets in existent approaches. Also, to solve the issues related to only having one programming languages, dataset contains various languages to train the model
Dataset Composition	Dataset contains ~1.5M pairs <code diff, commit message> from 35,530 GitHub projects in six programming languages (Go, Java, JavaScript, PHP, Python, and Ruby) with more than 500 stars, and filtered out according to data processing
Dataset Adjustment/Processing	Projects with more than 500 stars in GitHub, in six languages (Go, Java, JavaScript, PHP, Python, and Ruby), were filtered using GitHub REST API Projects with restrictive licenses were removed from results Same API was used to extract JSON file with commit history of selected projects History was parsed into pairs <code diff, message> for initial data recovered Initial data was filtered out by removing messages shorter than 3 tokens, code diffs larger than 100 tokens and code diffs for test files. Authors also filter out pairs from projects used to build fine-tuning tasks to prevent data leakage after dataset division
Dataset Division	Dataset is not divided into training, validation and test as it is only used to fine-tune the model in downstream tasks presented in model approach, while next dataset is used in experimentation
Dataset available online	Dataset available in https://www.zenodo.org/record/7964370, link extracted from code repository, direct link not provided in paper but script in code repository provided
Dataset 2: MCMD	
Dataset Origin	Filtered version of large dataset called MCMD proposed by Tao et al. ICSME '21.
Dataset Composition	Contains filtered pairs <diff,message> from dataset proposed in mentioned paper. It contains entries in 5 programming languages (C++, C#, Java, Python and JavaScript) from top 100 projects on GitHub. For each language: C++ - ~200,000 entries C# - ~188,000 entries Java - ~200,000 entries Python - ~257,000 entries JavaSript - ~250,000 entries
Dataset Adjustment/Processing	Original dataset contains entries for each of the 5 programming languages Filter redundant messages such as merge and rollback commits Filter out noisy message as stated in Liu et al. 2018 To balance size data, they retain 450,000 commits for each programming language Authors filter out commits with multiple files and non-parseable files (.mp3,.jar...)
Dataset Division	Dataset is divided in the 5 programming languages (C++/C#/Java/Python/JavaScript): Training set - 160,948/149,907/ 160,018/206,777/ 197,529 - Total 875,179 (~80%) Validation set - 20,000/18,688/19,825/25,912/24,899 - Total 109,324 (~10%) Test set - 20,141/18,702/20,159/25,837/24,773 - Total 109,612 (~10%)
Dataset available online	Available at link https://zenodo.org/records/7196966#.Y0juJHZBxmM

Dataset 3: FIRA	
Dataset Origin	As FIRA (treatment) need to use java dataset, original dataset from FIRA's paper is took to train this approach, which corresponds to CoDiSum dataset
Dataset Composition	The dataset, after preprocessing, contains 90,661 pairs of <diff, message> extracted from preprocessing Jiang and McMillan dataset.
Dataset Adjustment/Processing	From Jiang and McMillan dataset: Remove non .java file related commits and remove code of files with no code diffs Use NLP to remove punctuation and special symbols Tokenize diffs and message and remove < 3 words Delete duplicate commits and various modifications in short period of time
Dataset Division	Random division of pairs from dataset into: Training set - 75,000 pairs - (82.7%) Validation set - 8,000 pairs - (8.8%) Test set - 7,661 pairs - (8.5%)
Dataset available online	Dataset available at link of CoDiSum repository - https://github.com/SoftWiser-group/CoDiSum
Technological infrastructure	
Hardware	Server with 2x NVIDIA GeForce 4090 GPUs
Software	Paper only states popular framework PyTorch From code repository, extracted: Conda environment with Python version 3.8 pytorch=2.0.0 / torchvision=0.15.1 / torchaudio / datasets==1.16.1 / transformers==4.21.1/ tensorboard==2.12.2 / tree-sitter==0.19.1 / nltk=3.8.1 / scipy=1.10.1
Time and epochs	No mention of epochs or time used in approach training process
Optimization Hyperparameters	General model - Learning rate = 5e-05 (pre-training) and 2e-05 (fine-tuning) / Batch size = 32 From code repository (commit message generation task): Warmup steps = 500 / AdamW optimizer with initial learning rate = 3e-04 / Evaluation batch size = 8 / Max. training steps = 300,000 / Beam size = 5
Regularization Hyperparameters	No mention of regularization hyperparameters in paper From code repository: Implicit L1/L2 regularization in AdamW optimizer Dropout rate = 0.1
Training process	The training process starts by initializating the model with the weights and biases from final CodeT5 model, to start the training with a certain understanding of code-related tasks. From them, input is preprocessed and tags added and input enters the Encoder component. CCT5 is pre-trained and fine-tuned in five different code changes-related tasks: • Masked Language modelling for Code Changes: mask lines from code diff and input the message, for the model to predict the masked tokens. 15% of diff lines are masked • Masked Language modelling for Commit Message: mask words from commit message, input code diff and ask the model to predict tokens in message. 15% of tokens masked, 80% with MASK tag, 10% with random token and 10% unchanged token • Natural Language to Progamming Language generation: learn how to generate the new code based on the commit message and the old code • Programming Language to Natural Language generation: this is the commit message generation. Given a code diff, generate the commit message • Code Diff generation: generate the code diff from old and new data flows and old code Model is trained and fine-tuned in these tasks using proposed large dataset