GOTC 2023
全球开源技术峰会
THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

OPEN SOURCE，INTO THE FUTURE

大模型与软件开发自动化

李 戈

北京大學 AIXcoder

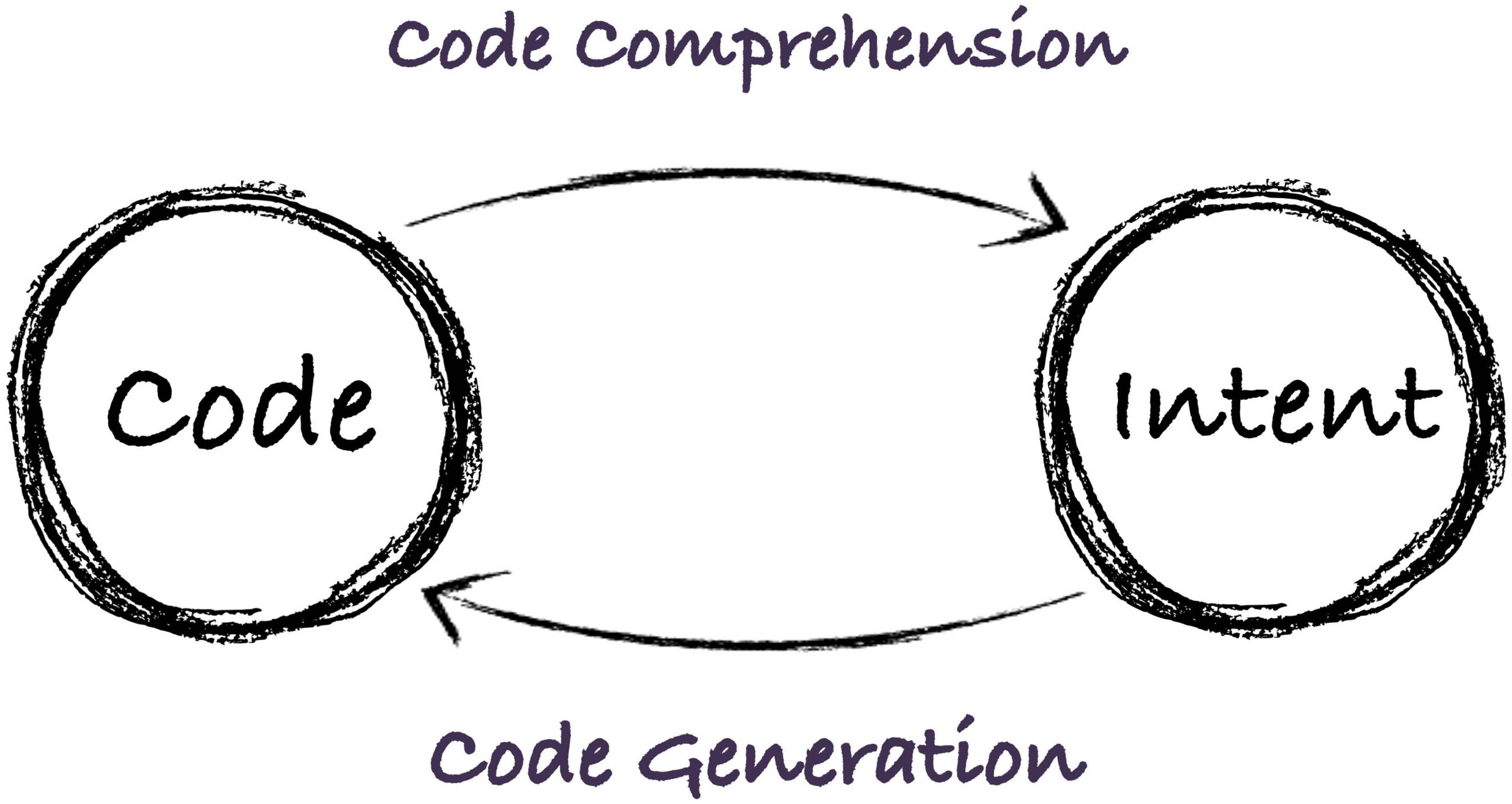
2022年05月27日

报告提纲

① 简要介绍我们的工作

② 大家可能关心的8个问题

My Research



The 1st Step…

Known as the 1st paper on DL based Program Representation

arXiv > cs > arXiv:1409.3358

Search...

Help | Advanced

Computer Science > Software Engineering

[Submitted on 11 Sep 2014]

Building Program Vector Representations for Deep Learning

Lili Mou, Ge Li, Yuxuan Liu, Hao Peng, Zhi Jin, Yan Xu, Lu Zhang

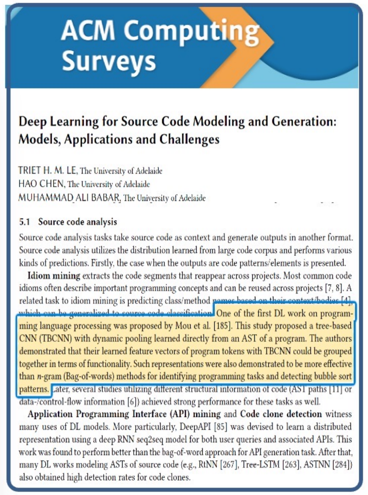
Deep learning has made significant breakthroughs in various fields of artificial intelligence. Advantages of deep learning include the ability to capture highly complicated features, weak involvement of human engineering, etc. However, it is still virtually impossible to use deep learning to analyze programs since deep architectures cannot be trained effectively with pure back propagation. In this pioneering paper, we propose the "coding criterion" to build program vector representations, which are the premise of deep learning for program analysis. Our representation learning approach directly makes deep learning a reality in this new field. We evaluate the learned vector representations both qualitatively and quantitatively. We conclude, based on the experiments, the coding criterion is successful in building program representations. To evaluate whether deep learning is beneficial for program analysis, we feed the representations to deep neural networks, and achieve higher accuracy in the program classification task than "shallow" methods, such as logistic regression and the support vector machine. This result confirms the feasibility of deep learning to analyze programs. It also gives primary evidence of its success in this new field. We believe deep learning will become an outstanding technique for program analysis in the near future.

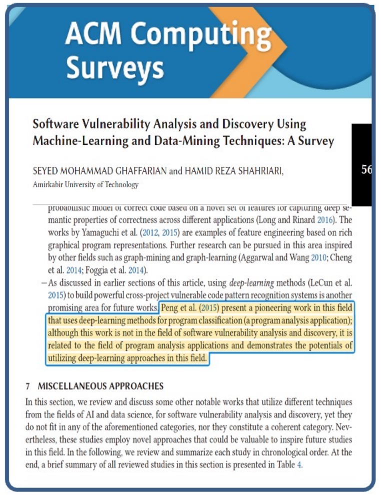
Comments: This paper was submitted to ICSE'14
Subjects: **Software Engineering (cs.SE)**; Machine Learning (cs.LG); Neural and Evolutionary Computing (cs.NE)
Cite as: arXiv:1409.3358 **[cs.SE]**
(or arXiv:1409.3358v1 **[cs.SE]** for this version)
https://doi.org/10.48550/arXiv.1409.3358

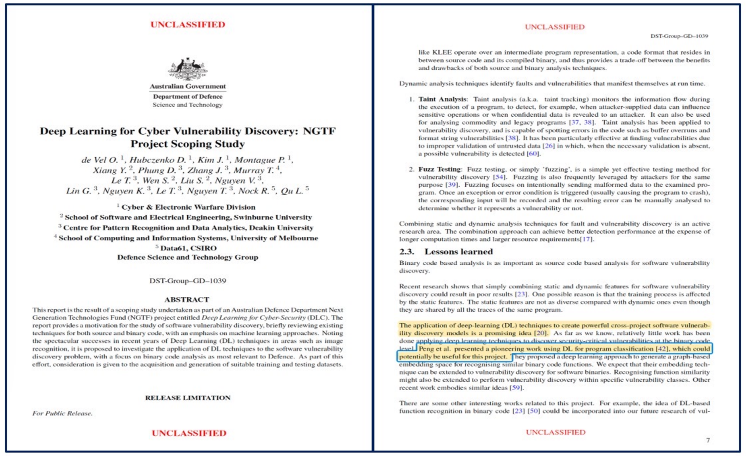
Submission history

From: Lili Mou [view email]
[v1] Thu, 11 Sep 2014 08:44:28 UTC (189 KB)

11 Sep. 2014







Lili Mou, Ge Li, Yuxuan Liu, Hao Peng, Zhi Jin, Yan Xu, Lu Zhang, Building Program Vector Representations for Deep Learning. arXiv preprint arXiv:1409.3358, 2014.

The 1st Step...

Known as the 1st paper on DL based Program Representation

arXiv > cs > arXiv:1409.5718

Search...

Help | Advanced

Computer Science > Machine Learning

[Submitted on 18 Sep 2014 (v1), last revised 8 Dec 2015 (this version, v2)]

Convolutional Neural Networks over Tree Structures for Programming Language Processing

Lili Mou, Ge Li, Lu Zhang, Tao Wang, Zhi Jin

Programming language processing (similar to natural language processing) is a hot research topic in the field of software engineering; it has also aroused growing interest in the artificial intelligence community. However, different from a natural language sentence, a program contains rich, explicit, and complicated structural information. Hence, traditional NLP models may be inappropriate for programs. In this paper, we propose a novel tree-based convolutional neural network (TBCNN) for programming language processing, in which a convolution kernel is designed over programs' abstract syntax trees to capture structural information. TBCNN is a generic architecture for programming language processing; our experiments show its effectiveness in two different program analysis tasks: classifying programs according to functionality, and detecting code snippets of certain patterns. TBCNN outperforms baseline methods, including several neural models for NLP.

Comments: Accepted at AAAI-16

Subjects: **Machine Learning (cs.LG)**; Neural and Evolutionary Computing (cs.NE); Software Engineering (cs.SE)

Cite as: arXiv:1409.5718 **[cs.LG]**

(or arXiv:1409.5718v2 **[cs.LG]** for this version)

https://doi.org/10.48550/arXiv.1409.5718

Submission history

From: Lili Mou [view email]

[v1] Thu, 18 Sep 2014 06:50:52 UTC (220 KB)

[v2] Tue, 8 Dec 2015 12:31:51 UTC (310 KB)

18 Sep. 2014

Lili Mou, Ge Li, Lu Zhang, Tao Wang, Zhi Jin, Convolutional Neural Networks over Tree Structures for Programming Language Processing, arXiv preprint arXiv:1409.5718, 2014.

ACM Computing Surveys

Deep Learning for Source Code Modeling and Generation: Models, Applications and Challenges

TRIET H. M. LE, The University of Adelaide
HAO CHEN, The University of Adelaide
MUHAMMAD ALI BABAR, The University of Adelaide

5.1 Source code analysis

Source code analysis tasks take source code as context and generate outputs in another format. Source code analysis utilizes the distribution learned from large code corpus and performs various kinds of predictions. Firstly, the case when the outputs are code patterns/elements is presented.

Idiom mining extracts the code segments that reappear across projects. Most common code idioms often describe important programming concepts and can be reused across projects [7, 8]. A related task to idiom mining is predicting class/method names based on their context/bodies [4] which can be generalized to source code classification. One of the first DL work on programming language processing was proposed by Mou et al. [185]. This study proposed a tree-based CNN (TBCNN) with dynamic pooling learned directly from an AST of a program. The authors demonstrated that their learned feature vectors of program tokens with TBCNN could be grouped together in terms of functionality. Such representations were also demonstrated to be more effective than *n*-gram (Bag-of-words) methods for identifying programming tasks and detecting bubble sort patterns. Later, several studies utilizing different structural information of code (AST paths [11] or data-/control-flow information [6]) achieved strong performance for these tasks as well.

Application Programming Interface (API) mining and **Code clone detection** witness many uses of DL models. More particularly, DeepAPI [85] was devised to learn a distributed representation using a deep RNN seq2seq model for both user queries and associated APIs. This work was found to perform better than the bag-of-word approach for API generation task. After that, many DL works modeling ASTs of source code (e.g., RtNN [267], Tree-LSTM [263], ASTNN [284]) also obtained high detection rates for code clones.

The first person to eat crabs is not easy.

At that time,

No TensorFlow, No Pytorch, or even No Theano, No GPU.

We have to **build everything from scratch.**

The 1st Step...

Known as the 1st paper on DL based Code Generation

arXiv > cs > arXiv:1510.07211

Search...

Help | Advanced

Computer Science > Software Engineering

[Submitted on 25 Oct 2015]

On End-to-End Program Generation from User Intention by Deep Neural Networks

Lili Mou, Rui Men, Ge Li, Lu Zhang, Zhi Jin

This paper envisions an end-to-end program generation scenario using recurrent neural networks (RNNs): Users can express t
RNN then automatically generates corresponding code in a characterby-by-character fashion. We demonstrate its feasibility th
analysis. To fully make such technique useful in practice, we also point out several cross-disciplinary challenges, including mo
improving model architectures, etc. Although much long-term research shall be addressed in this new field, we believe end-to
a reality in future decades, and we are looking forward to its practice.

Comments: Submitted to 2016 International Conference of Software Engineering "Vision of 2025 and Beyond" track
Subjects: **Software Engineering (cs.SE)**; Machine Learning (cs.LG)
Cite as: arXiv:1510.07211 **[cs.SE]**
(or arXiv:1510.07211v1 **[cs.SE]** for this version)
https://doi.org/10.48550/arXiv.1510.07211

Submission history

From: Lili Mou [view email]
[v1] Sun, 25 Oct 2015 06:52:45 UTC (217 KB)

```
#include<stdio.h>
void main()
{        (1) n
    int x,a[100],i,max1,max2;
    scanf("%d",&n);
    for(i=0;i<=n-1;i++)
    {
        scanf("%d",&a[i]);
        if (a[i]>max1)
            max2=a[i];}
        for(i=1;i<=n;i++)
    {                  (2) 2   (3) <
        if(a[i]>max&&a[i]==max1)
            max2=a[i];
    }

    printf("%d\n%d",max1,max2);
    return 0; (4) [del]
}
```

(a) Generated code

```
#include<stdio.h>
int main(){
     int n,i,j,sz[100],max=0,ci=0;
     scanf("%d",&n);
    for(i=0;i<n;i++){
        scanf("%d",&sz[i]);
        if(sz[i]>max){
           max=sz[i];}}
      for(i=0;i<n;i++){
 if(sz[i]>ci&&sz[i]<max){
        ci=sz[i];}}
 printf("%d\n%d",max,ci);
return 0;
}
```

(b) Training sample 1

```
#include<stdio.h>
void main()
{
    int n,i,a[100],j,max1,max2;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    max1=a[0];
    for(i=0;i<n;i++)
    {
        if(a[i]>max1)
            max1=a[i];
    }
    for(i=0;i<n;i++)
    {
        if(max1==a[i])
            j=i;
    }
    if(max1!=a[0])
    max2=a[0];
    else max2=a[1];
    for(i=0;i<n;i++)
    {
        if(i==j) continue;
        if(a[i]>max2)
            max2=a[i];
    }
        printf("%d\n%d",max1,max2);
}
```

(c) Training sample 2

Figure 2: (a) Code generated by RNN. The code is almost correct except 4 wrong characters (among ~280 characters in total), highlighted in the figure. (b) Code with the most similar structure in the ...ining set, detected by ccfinder. (c) Code with the most similar ...ntifiers in the training set, also detected by ccfinder. Note that ... preserve all indents, spaces and line feeds. The 4 errors are (1) ...e identifier "x" should be "n"; (2) "max" should be "max2"; (3) "==" should be "<"; (4) return type should be void.

25 Oct. 2015

Lili Mou, Rui Men, Ge Li, Lu Zhang, Zhi Jin, On End-to-End Program Generation from User Intention by Deep Neural Networks, arXiv preprint, arXiv:1510.07211, 2015.

The 1st Step...

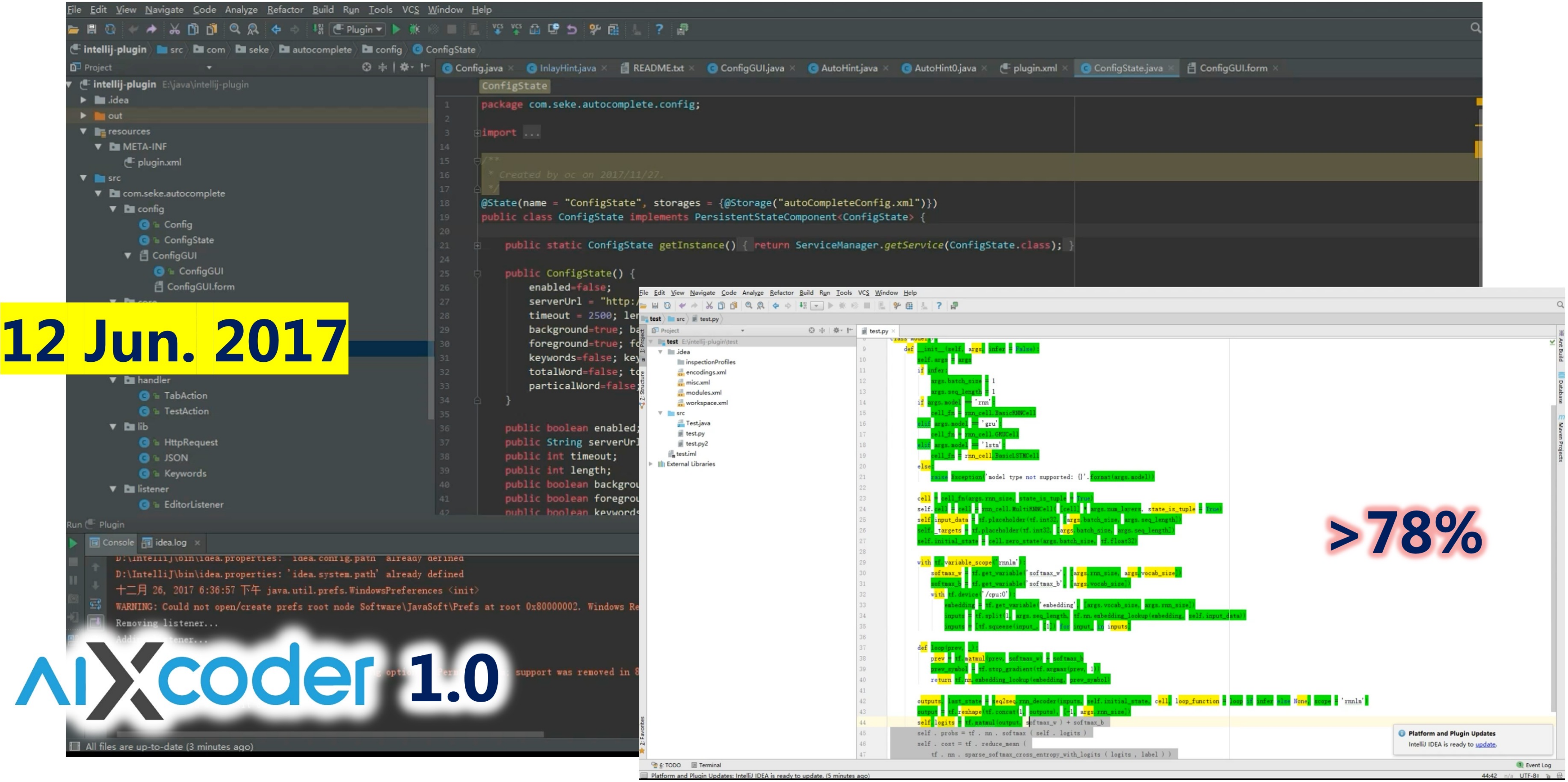
Known as the 1st Tool on DL based Code Generation

```python
import sys

class Model():
    def __init__(self, args, infer = False):
        self.args = args
        if infer :
            args.batch_size = 1
            args.seq_length = 1
        if args.model == 'rnn':
            cell_fn = rnn_cell.BasicRNNCell
        elif args.model == 'gru':
            cell_fn = rnn_cell.GRUCell
        elif args . model == <str> : cell_fn = <UNK> .
```

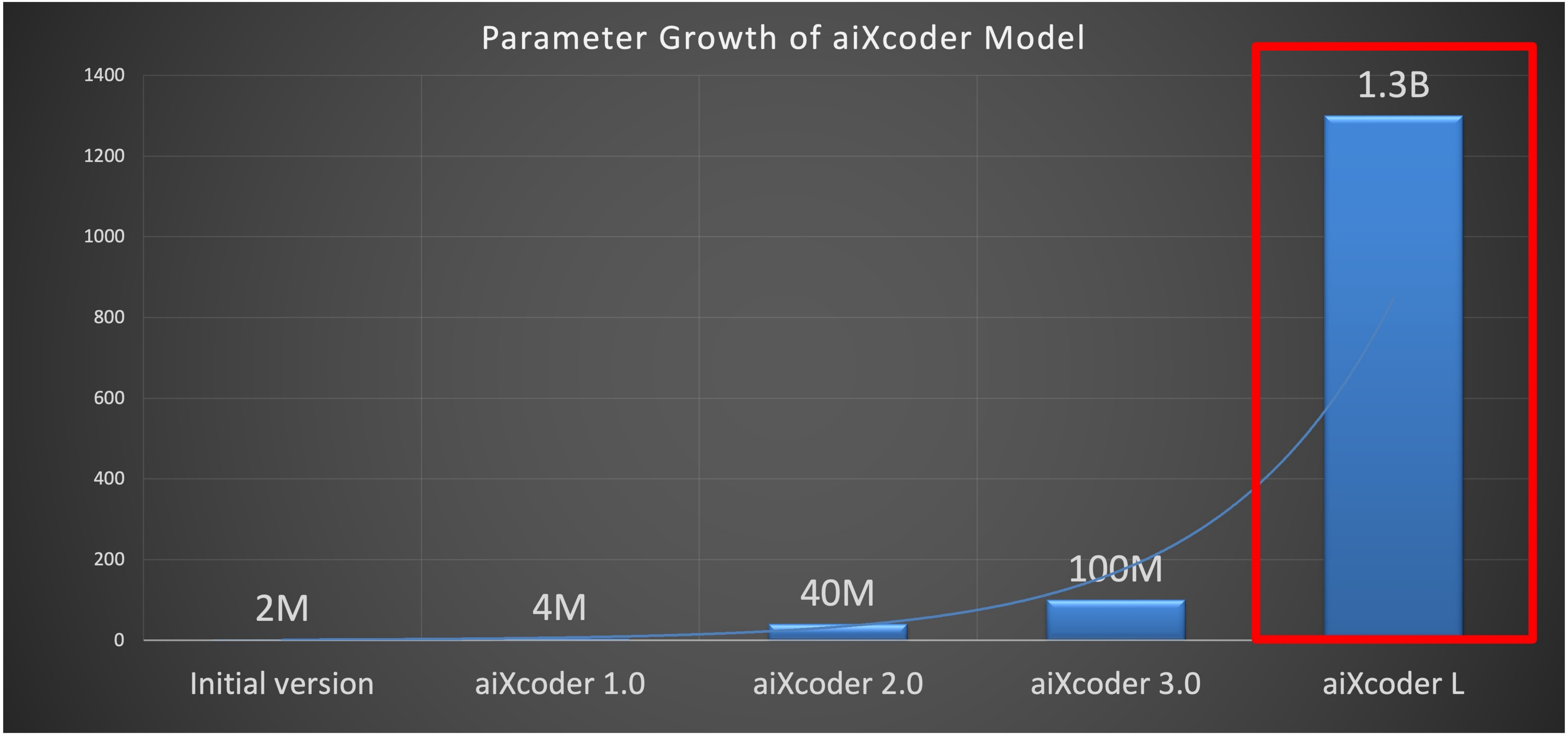
0:41.08

16 May 2017

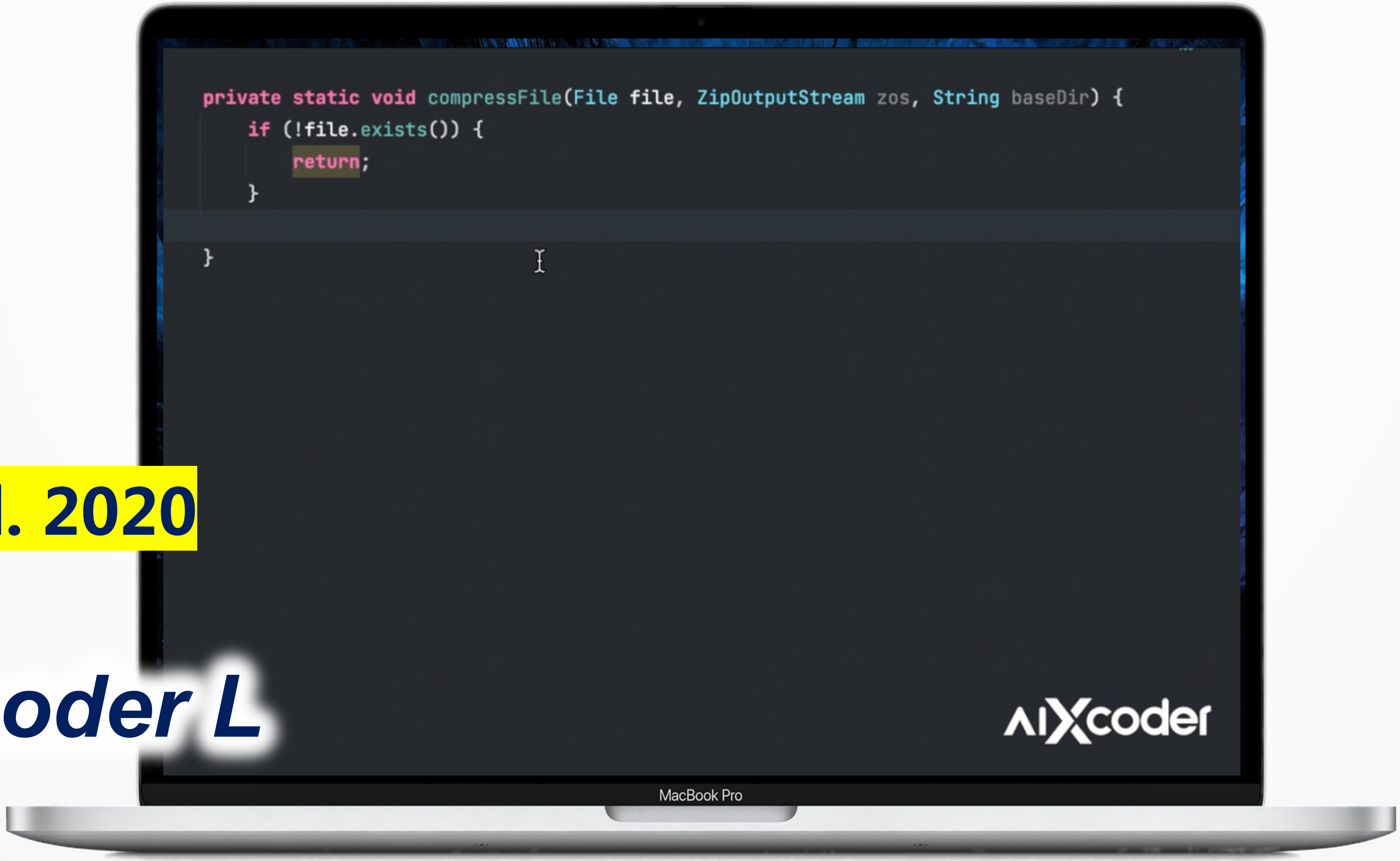
The 1st Step...



Model capabilities vs. Number of parameters



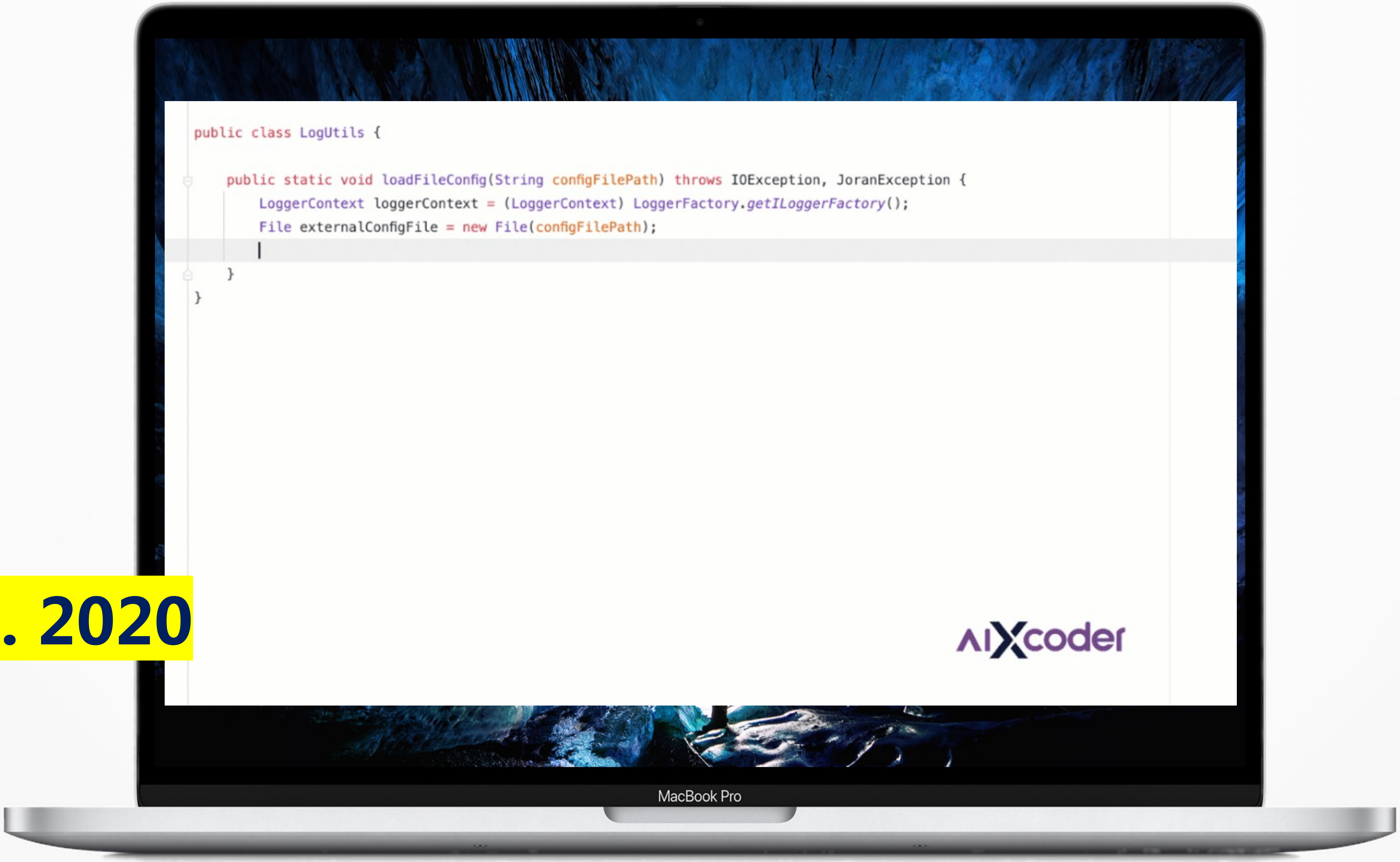
aiXcoder Full Line Code Completion



15 Jul. 2020

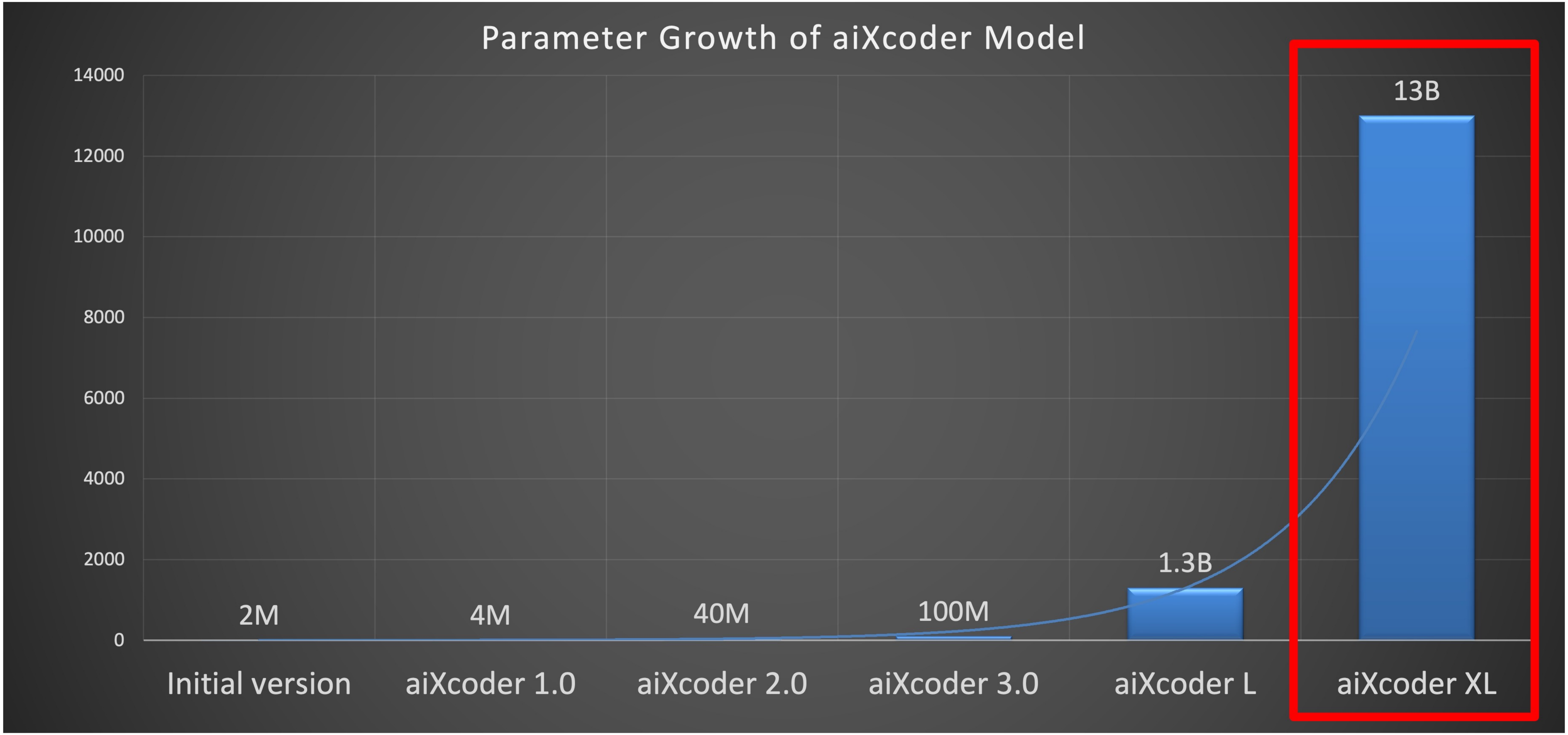
aiXcoder L

aiXcoder Full Line Code Completion



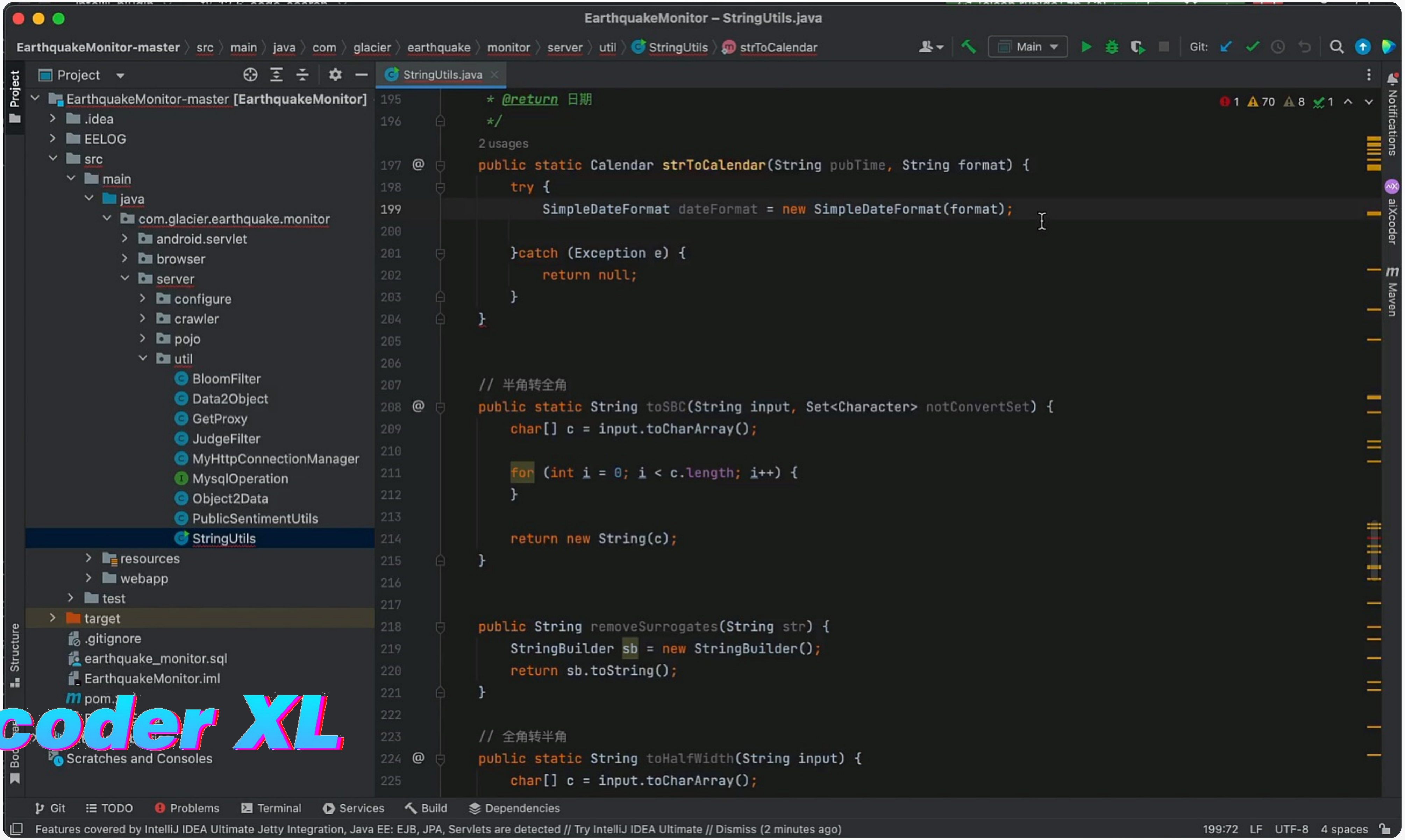
15 Jul. 2020

Model capabilities vs. Number of parameters



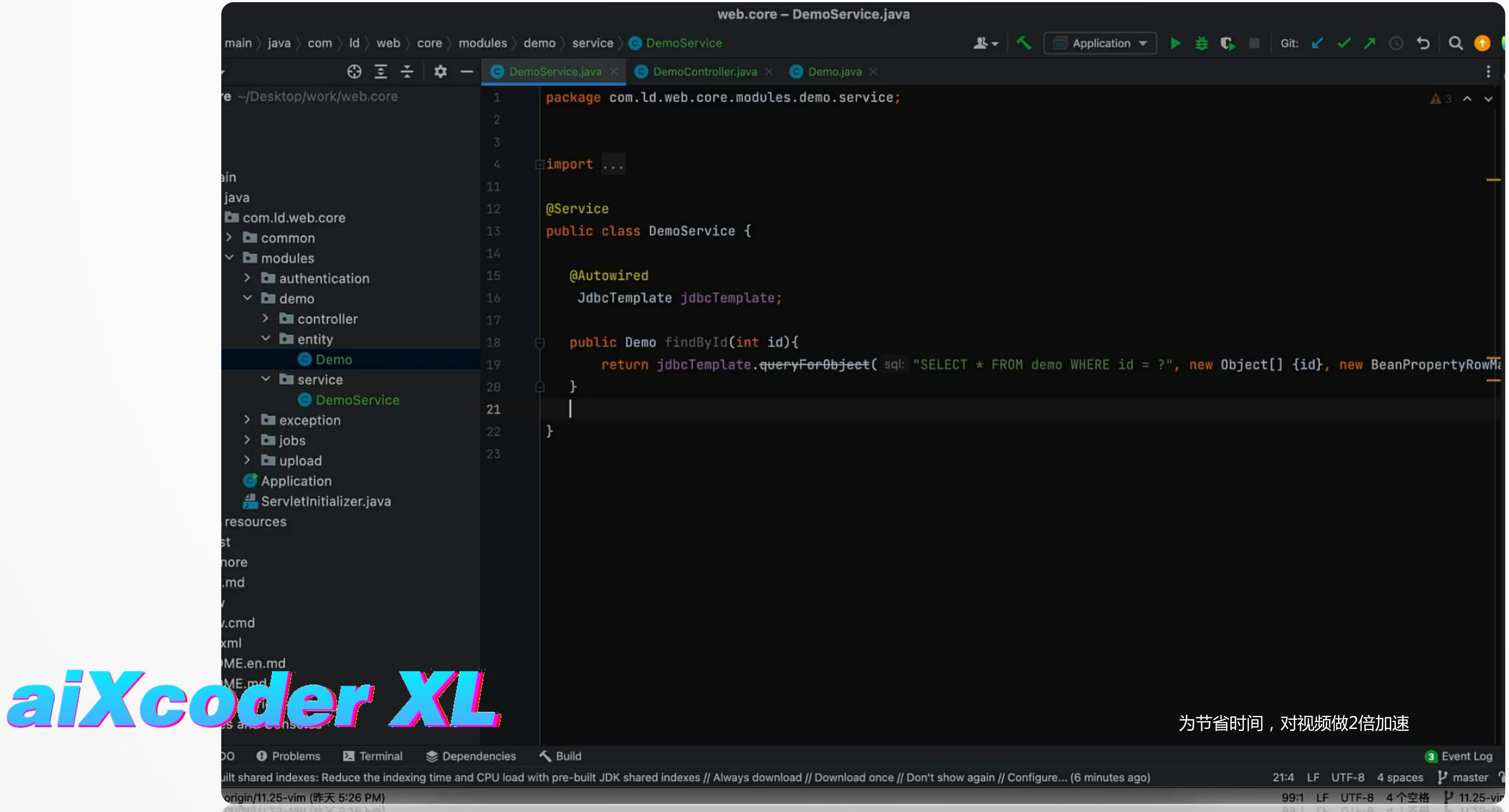

aiXcoder
aiXcoder XL
支持方法级代码生成
根据自然语言功能描述，生成完整程序代码
对标Copilot，国内首个：自然语言一键生成方法级代码 aiXcoder XL…
国内首个能够根据开发者给出的“自然语言描述”一键生成“完整方法级代码”的智能编程应…
aiXcoder XL
支持方法级代码生成
机器之心
22 Jun. 2022

aiXcoder 多行代码自动补全

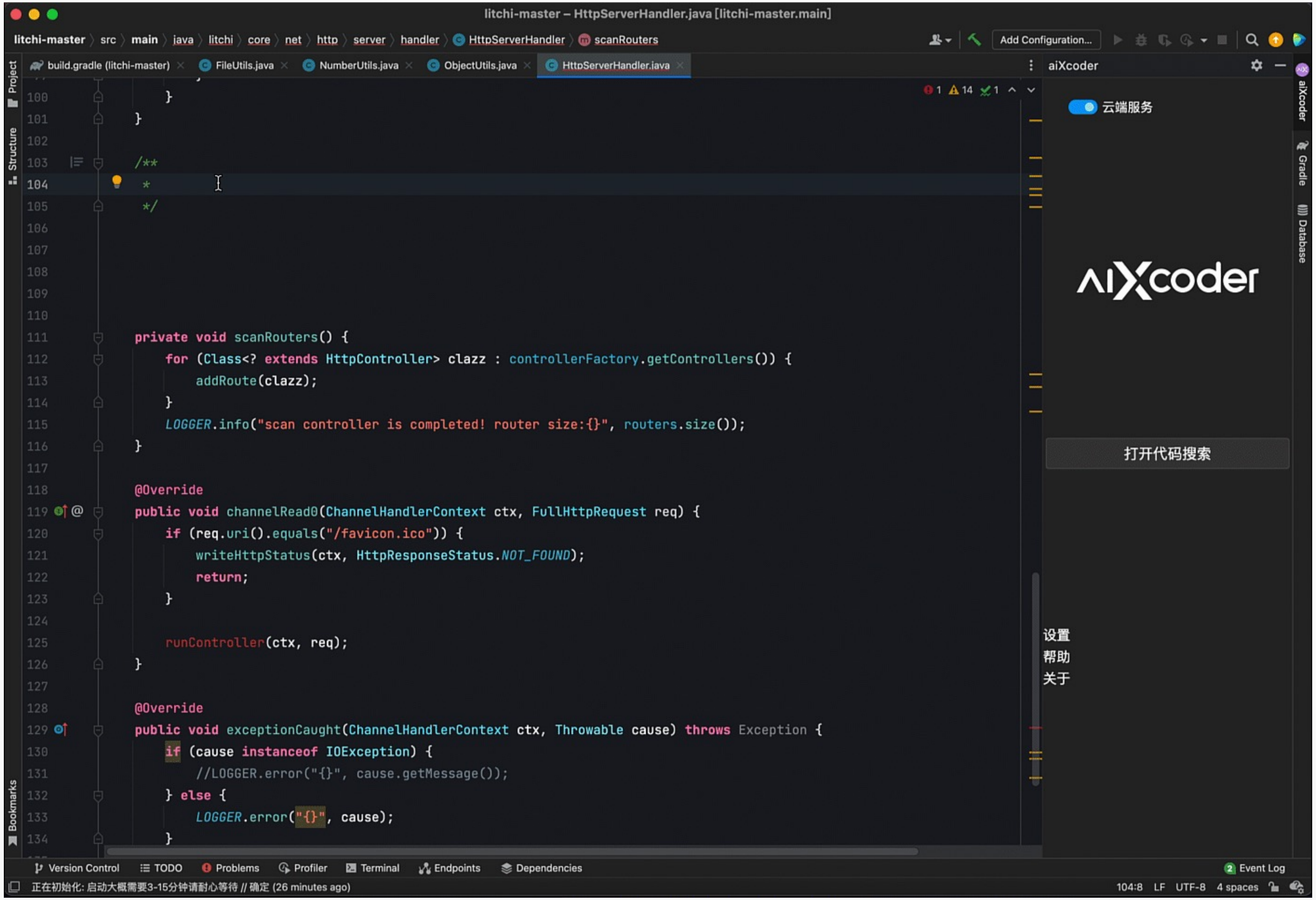


aiXcoder XL

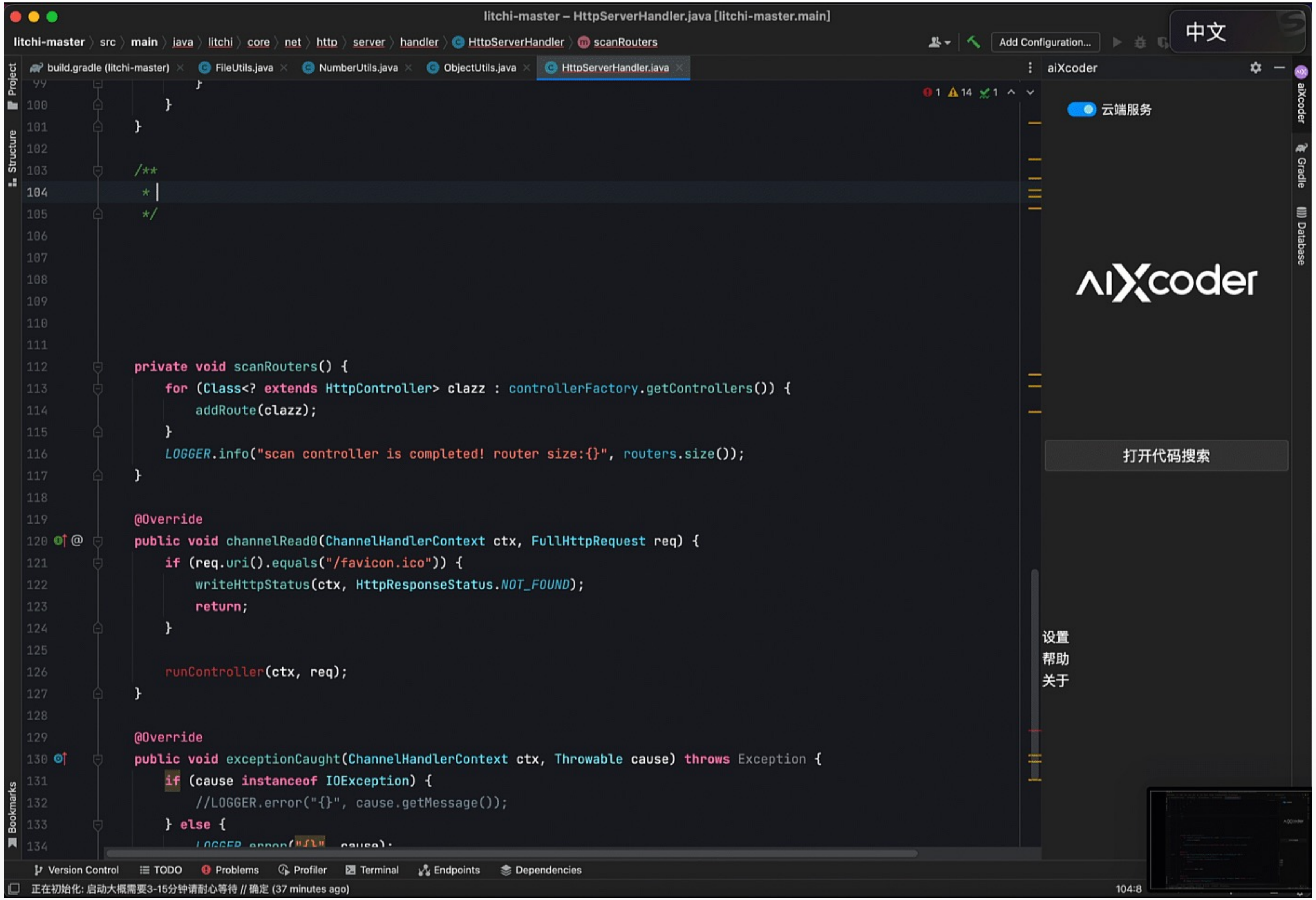
aiXcoder 开发框架代码自动补全



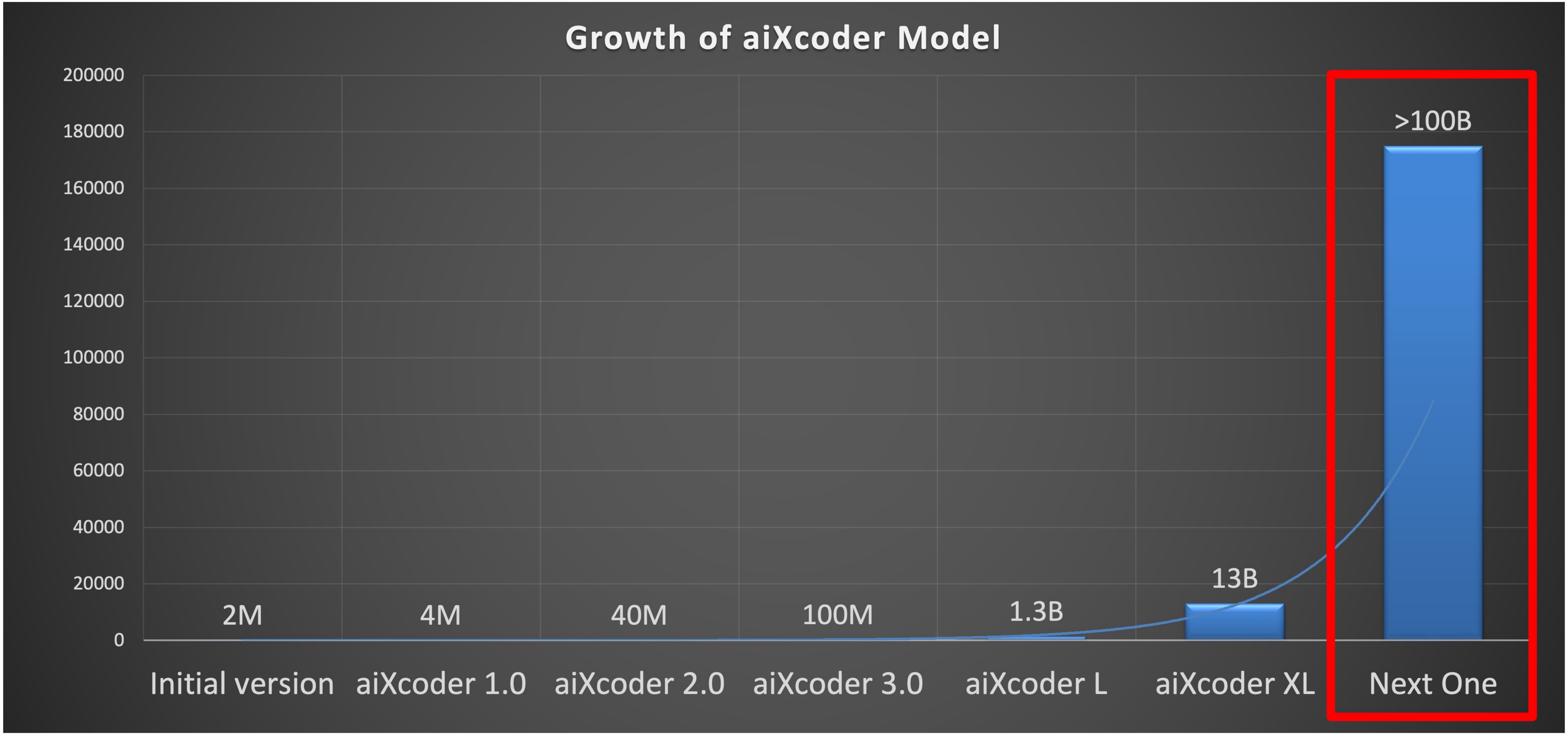
Full Method Code Completion



AIXcoder Full Method Code Completion



What we're trying to do



问题1：大模型的代码生成能力到什么程度？

- 如果开发者能把需求描述得足够清楚，大模型似乎都能生成可用代码。

问题2：大模型的“能力”哪里来？

什么是大模型？

LLM = Large Language Model

- “大模型”是个俗称，如果您理解的大模型是类似于ChatGPT的模型，那么：
  - 错误认知一：参数量“非常大”的模型就叫“大模型”
  - 错误认知二：不依赖自然语言也可以做大模型

Table 1: Summary of PFMs in Text. The pretraining task includes language model (LM), masked LM (MLM), permuted LM (PLM), denoising autoencoder (DAE), knowledge graphs (KG), and knowledge embedding (KE).

Year	Conference	Model	Architecture	Embedding	Training method	Code
2013	NeurIPS	Skip-Gram [66]	Word2Vec	Probabilistic	-	https://github.com/.../models
2014	EMNLP	GloVe [67]	Word2Vec	Probabilistic	-	-
2015	NeurIPS	LM-LSTM [68]	LSTM	Probabilistic	LM	https://github.com/.../GloVe
2016	IJCAI	Shared LSTM [69]	LSTM	Probabilistic	LM	https://github.com/.../adversarial_text
2017	TACL	FastText [70]	Word2Vec	Probabilistic	-	https://github.com/.../fastText
2017	NeurIPS	CoVe [71]	LSTM+Seq2Seq	Probabilistic	-	https://github.com/.../cove
2018	NAACL-HLT	ELMO [51]	LSTM	Contextual	LM	https://allennlp.org/elmo
2018	NAACL-HLT	BERT [13]	Transformer Encoder	Contextual	MLM	https://github.com/.../bert
2018		OpenAI GPT [48]	Transformer Decoder	Autoregressive	LM	https://github.com/...transformer-lm
2019	ACL	ERNIE(THU)	Transformer Encoder	Contextual	MLM	https://github.com/.../ERNIE
2019	ACL	Transformer-XL [72]	Transformer-XL	Contextual	-	https://github.com/.../transformer-xl
2019	ICLR	InfoWord [73]	Transformer Encoder	Contextual	MLM	-
2019	ICLR	StructBERT [74]	Transformer Encoder	Contextual	MLM	-
2019	ICLR	ALBERT [45]	Transformer Encoder	Contextual	MLM	https://github.com/.../ALBERT
2019	ICLR	WKLM [75]	Transformer Encoder	Contextual	MLM	-
2019	ICML	MASS [57]	Transformer	Contextual	MLM(Seq2Seq)	https://github.com/.../MASS
2019	EMNLP-IJCNLP	KnowBERT [76]	Transformer Encoder	Contextual	MLM	https://github.com/.../kb
2019	EMNLP-IJCNLP	Unicoder [77]	Transformer Encoder	Contextual	MLM+TLM	-
2019	EMNLP-IJCNLP	MultiFit [78]	QRNN	Probabilistic	LM	https://github.com/.../multifit
2019	EMNLP-IJCNLP	SciBERT [79]	Transformer Encoder	Contextual	MLM	https://github.com/.../scibert
2019	EMNLP-IJCNLP	BERT-PKD [80]	Transformer Encoder	Contextual	MLM	https://github.com/...Compression
2019	NeurIPS	Xlnet [14]	Transformer-XL Encoder	Permutation	PLM	https://github.com/.../xlnet
2019	NeurIPS	UNILM [58]	LSTM + Transformer	Contextual	LM + MLM	https://github.com/.../unilm
2019	NeurIPS	XLM [81]	Transformer Encoder	Contextual	MLM+CLM+TLM	https://github.com/.../XLM
2019	OpenAI Blog	GPT-2 [49]	Transformer Decoder	Autoregressive	LM	https://github.com/.../gpt-2
2019	arXiv	RoBERTa [53]	Transformer Encoder	Contextual	MLM	https://github.com/.../fairseq
2019	arXiv	ERNIE(Baidu) [59]	Transformer Encoder	Contextual	MLM+DLM	https://github.com/.../ERNIE
2019	EMC2@NeurIPS	Q8BERT [82]	Transformer Encoder	Contextual	MLM	https://github.com/.../quantized_bert.py
2019	arXiv	DistilBERT [83]	Transformer Encoder	Contextual	MLM	https://github.com/.../distillation
2020	ACL	fastBERT [84]	Transformer Encoder	Contextual	MLM	https://github.com/.../FastBERT
2020	ACL	SpanBERT [42]	Transformer Encoder	Contextual	MLM	https://github.com/.../SpanBERT
2020	ACL	BART [43]	Transformer	En: Contextual De: Autoregressive	DAE	https://github.com/.../transformers
2020	ACL	CamemBERT [85]	Transformer Encoder	Contextual	MLM(WWM)	https://camembert-model.fr
2020	ACL	XLM-R [86]	Transformer Encoder	Contextual	MLM	https://github.com/.../XLM
2020	ICLR	Reformer [87]	Reformer	Permutation	-	https://github.com/.../reformer
2020	ICLR	ELECTRA [44]	Transformer Encoder	Contextual	MLM	https://github.com/.../electra
2020	AAAI	Q-BERT [88]	Transformer Encoder	Contextual	MLM	-
2020	AAAI	XNLG [89]	Transformer	Contextual	MLM+DAE	https://github.com/.../xnlg
2020	AAAI	K-BERT [90]	Transformer Encoder	Contextual	MLM	https://github.com/.../K-BERT
2020	AAAI	ERNIE 2.0 [60]	Transformer Encoder	Contextual	MLM	https://github.com/.../ERNIE
2020	NeurIPS	GPT-3 [20]	Transformer Decoder	Autoregressive	LM	https://github.com/.../gpt-3
2020	NeurIPS	MPNet [55]	Transformer Encoder	Permutation	MLM+PLM	https://github.com/.../MPNet
2020	NeurIPS	ConvBERT [91]	Mixed Attention	Contextual	-	https://github.com/.../ConvBert
2020	NeurIPS	MiniLM [92]	Transformer Encoder	Contextual	MLM	https://github.com/.../minilm
2020	TACL	mBART [93]	Transformer	Contextual	DAE	https://github.com/.../mbart
2020	COLING	CoLAKE [94]	Transformer Encoder	Contextual	MLM+KE	https://github.com/.../CoLAKE
2020	LREC	FlauBERT [95]	Transformer Encoder	Contextual	MLM	https://github.com/.../Flaubert
2020	EMNLP	GLM [96]	Transformer Encoder	Contextual	MLM+KG	https://github.com/.../GLM
2020	EMNLP (Findings)	TinyBERT [97]	Transformer	Contextual	MLM	https://github.com/.../TinyBERT
2020	EMNLP (Findings)	RobBERT [98]	Transformer Encoder	Contextual	MLM	https://github.com/.../RobBERT
2020	EMNLP (Findings)	ZEN [62]	Transformer Encoder	Contextual	MLM	https://github.com/.../ZEN
2020	EMNLP (Findings)	BERT-MK [99]	KG-Transformer Encoder	Contextual	MLM	-
2020	RepL4NLP@ACL	CompressingBERT [33]	Transformer Encoder	Contextual	MLM(Pruning)	https://github.com/.../bert-prune
2020	JMLR	T5 [100]	Transformer	Contextual	MLM(Seq2Seq)	https://github.com/...transformer
2021	T-ASL	BERT-wwm-Chinese [61]	Transformer Encoder	Contextual	MLM	https://github.com/...BERT-wwm
2021	EACL	PET [101]	Transformer Encoder	Contextual	MLM	https://github.com/.../pet
2021	TACL	KEPLER [102]	Transformer Encoder	Contextual	MLM+KE	https://github.com/.../KEPLER
2021	EMNLP	SimCSE [103]	Transformer Encoder	Contextual	MLM+KE	https://github.com/.../SimCSE
2021	ICML	GLaM [104]	Transformer	Autoregressive	LM	-
2021	arXiv	XLM-E [105]	Transformer	Contextual	MLM	
2021	arXiv	T0 [106]	Transformer	Contextual	MLM	https://github.com/.../T0
2021	arXiv	Gopher [107]	Transformer	Autoregressive	LM	-
2022	arXiv	MT-NLG [108]	Transformer	Contextual	MLM	-
2022	arXiv	LaMDA [65]	Transformer Decoder	Autoregressive	LM	https://github.com/.../LaMDA
2022	arXiv	Chinchilla [109]	Transformer	Autoregressive	LM	-
2022	arXiv	PaLM [41]	Transformer	Autoregressive	LM	https://github.com/.../PaLM
2022	arXiv	OPT [110]	Transformer Decoder	Autoregressive	LM	https://github.com/.../MetaSeq

语言是人类思维的工具。

—— 记忆中来自马克思

N. Mercer, Words and minds: How we use language to think together. Psychology Press, 2000.

问题3：训练大模型需要多大代价？

Training Cost of Large Language Models

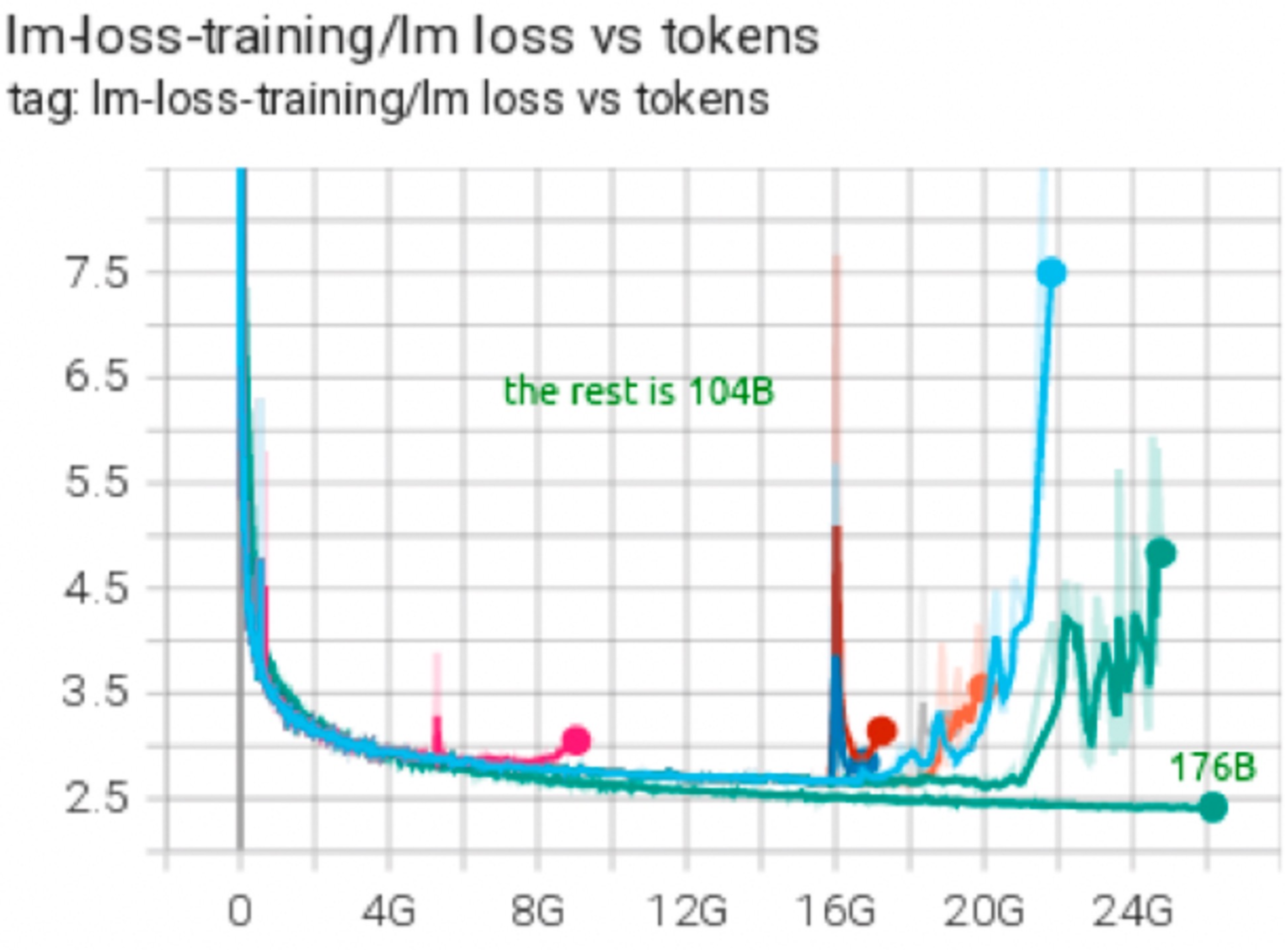
Model	Total train compute (PF-days)	Total train compute (flops)	Params (M)	Training tokens (billions)	Flops per param per token	Mult for bwd pass	Fwd-pass flops per active param per token	Frac of params active for each token
T5-Small	2.08E+00	1.80E+20	60	1,000	3	3	1	0.5
T5-Base	7.64E+00	6.60E+20	220	1,000	3	3	1	0.5
T5-Large	2.67E+01	2.31E+21	770	1,000	3	3	1	0.5
T5-3B	1.04E+02	9.00E+21	3,000	1,000	3	3	1	0.5
T5-11B	3.82E+02	3.30E+22	11,000	1,000	3	3	1	0.5
BERT-Base	1.89E+00	1.64E+20	109	250	6	3	2	1.0
BERT-Large	6.16E+00	5.33E+20	355	250	6	3	2	1.0
RoBERTa-Base	1.74E+01	1.50E+21	125	2,000	6	3	2	1.0
RoBERTa-Large	4.93E+01	4.26E+21	355	2,000	6	3	2	1.0
GPT-3 Small	2.60E+00	2.25E+20	125	300	6	3	2	1.0
GPT-3 Medium	7.42E+00	6.41E+20	356	300	6	3	2	1.0
GPT-3 Large	1.58E+01	1.37E+21	760	300	6	3	2	1.0
GPT-3 XL	2.75E+01	2.38E+21	1,320	300	6	3	2	1.0
GPT-3 2.7B	5.52E+01	4.77E+21	2,650	300	6	3	2	1.0
GPT-3 6.7B	1.39E+02	1.20E+22	6,660	300	6	3	2	1.0
GPT-3 13B	2.68E+02	2.31E+22	12,850	300	6	3	2	1.0
GPT-3 175B	3.64E+03	3.14E+23	174,600	300	6	3	2	1.0

Table D.1: Starting from the right hand side and moving left, we begin with the number of training tokens that each model was trained with. Next we note that since T5 uses an encoder-decoder model, only half of the parameters are active for each token during a forward or backwards pass. We then note that each token is involved in a single addition and a single multiply for each active parameter in the forward pass (ignoring attention). Then we add a multiplier of 3x to account for the backwards pass (as computing both $\frac{\partial params}{\partial loss}$ and $\frac{\partial acts}{\partial loss}$ use a similar amount of compute as the forwards pass. Combining the previous two numbers, we get the total flops per parameter per token. We multiply this value by the total training tokens and the total parameters to yield the number of total flops used during training. We report both flops and petaflop/s-day (each of which are 2.88e+7 flops).

The 175B GPT-3 model trained by OpenAI required 14.8 days [17] of compute on 10,000 V100s, and consumed 3.14+23 FLOPs.

That's just the cost of
taste one time of success!

Multiple tests before success are necessary



(b) BLOOM 176B's experiments

Multiple tests before success are necessary

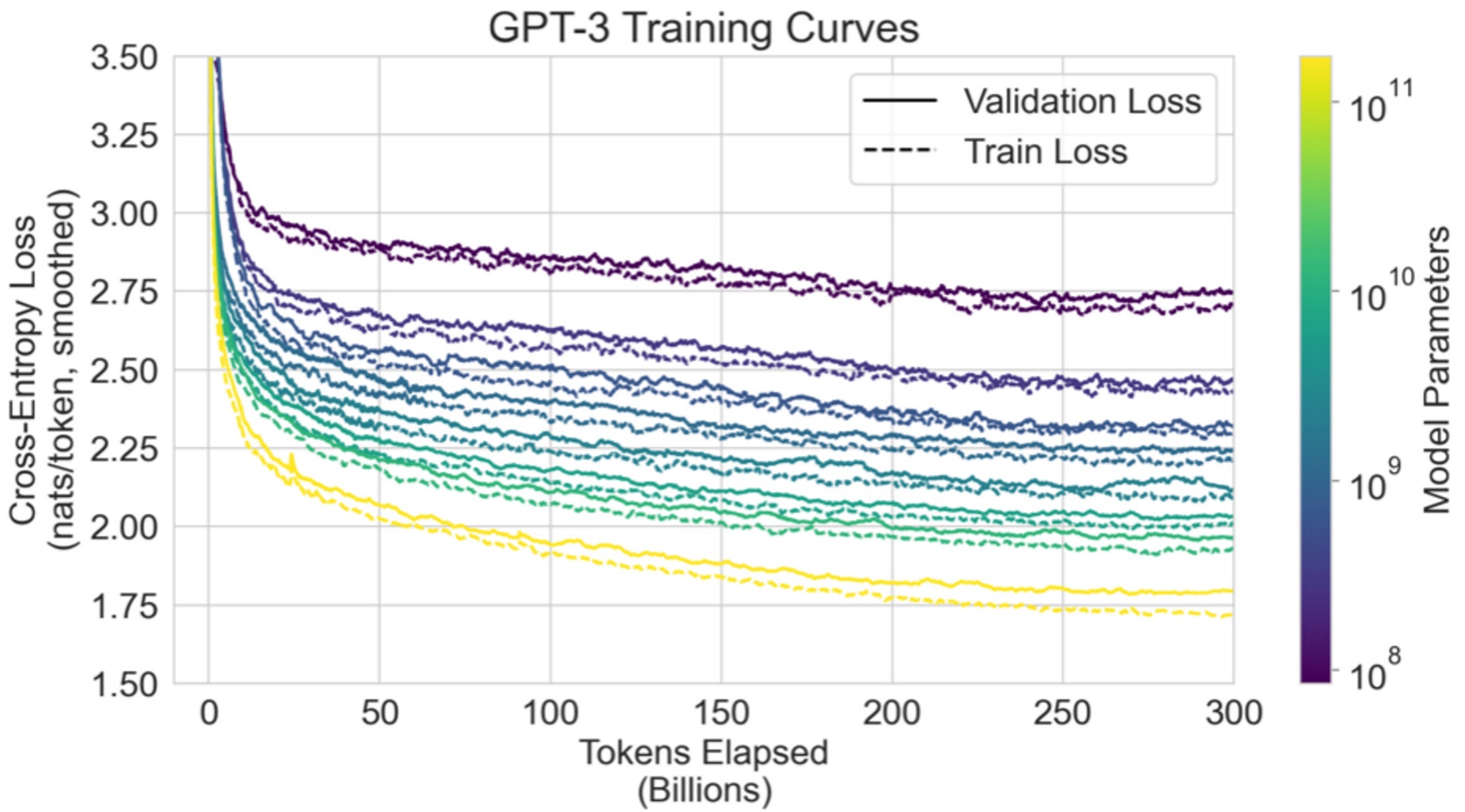
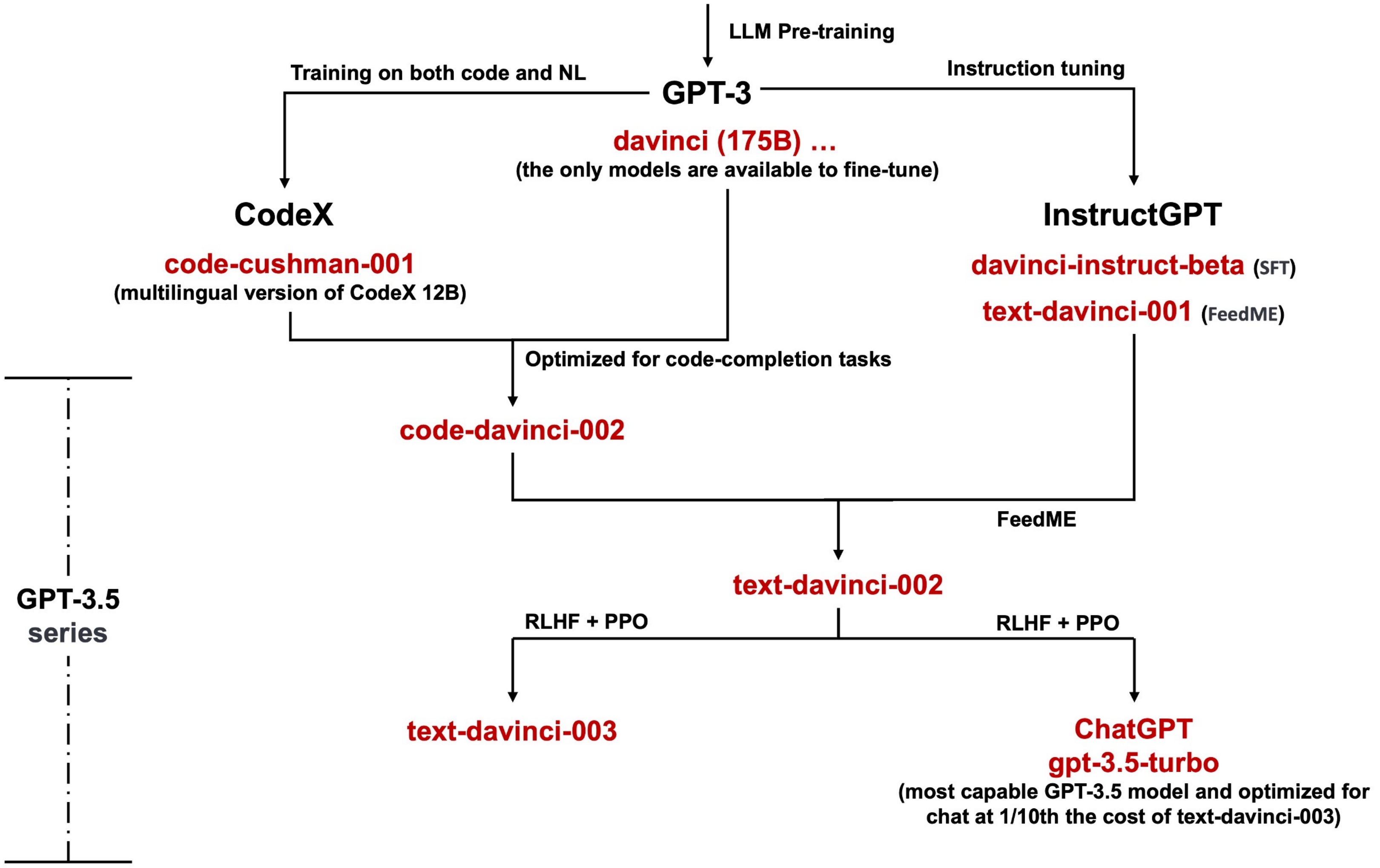


Figure 4.1: GPT-3 Training Curves We measure model performance during training on a deduplicated validation split of our training distribution. Though there is some gap between training and validation performance, the gap grows only minimally with model size and training time, suggesting that most of the gap comes from a difference in difficulty rather than overfitting.

问题3：为什么代码对LLM的训练有帮助？



SFT Supervised fine-tuning on human demonstrations
FeedME Supervised fine-tuning on human-written demonstrations and on model samples rated 7/7 by human labelers on an overall quality score
RLHF Reinforcement Learning from Human Feedback
PPO Reinforcement learning with reward models trained from comparisons by humans

为什么代码对LLM的训练有帮助？

（1）程序语言具有自然性

- 程序语言具有与自然语言相似的统计特性，所以，可以利用语言模型进行建模，只是这种建模方式并不高效。

Naturalness： Software is a form of human communication; software corpora have similar statistical properties to natural language corpora; and these properties can be exploited to build better software engineering tools.

为什么代码对LLM的训练有帮助？

（2）代码与自然语言之间存在对应性

- 例如，可以把代码模块映射为自然语言描述，也可以利用自然语言描述的思维链辅助代码的生成。

```
def convert\_time\_unit(hours, minutes):
 '''
 Write a function to convert time unit.
 '''
 seconds = hours \* 60 \* 60
 seconds = minutes \* 60
 return seconds

# Generate some comments for above code

def convert\_time\_unit(hours, minutes):
 '''
 Write a function to convert time unit.
 '''
 # Convert the time unit from hours to seconds.
 seconds = hours \* 60 \* 60
 # Convert the time unit from minutes to seconds.
 seconds = minutes \* 60
 # Return the number of seconds.
 return seconds
```

```
Task Id: 608
#
# 1. Define a function "bell_Number"
def bell_Number(n):
 # 2. Initialize a list "bell" with n+1 rows and n+1 columns.
 bell = [[0 for i in range(n+1)] for j in range(n+1)]

 # 3. Set the first element of the list "bell" to 1.
 bell[0][0] = 1
 # 4. Fill the first column of the list "bell" with the first n+1 Bell numbers.
 for i in range(1, n+1):
  bell[i][0] = bell[i-1][i-1]
 # 5. Fill the rest of the list "bell" with the next n+1 Bell numbers.
 for i in range(1, n+1):
  for j in range(1, i+1):
   bell[i][j] = bell[i-1][j-1] + bell[i][j-1]

 # 6. Return the first element of the list "bell".
 return bell[n][0]
```

为什么代码对LLM的训练有帮助？

（3）程序语言结构性、顺序性、层次性强

- 代码的语义是对现实世界某个解决方案的描述，而且这种描述通常具有更加清晰的顺序性、结构性和层次性，有助于现实世界知识的学习。

```
def convert\_time\_unit(hours, minutes):
 '''
 Write a function to convert time unit.
 '''
 seconds = hours \* 60 \* 60
 seconds = minutes \* 60
 return seconds

# Generate some comments for above code

def convert\_time\_unit(hours, minutes):
 '''
 Write a function to convert time unit.
 '''
 # Convert the time unit from hours to seconds.
 seconds = hours \* 60 \* 60
 # Convert the time unit from minutes to seconds.
 seconds = minutes \* 60
 # Return the number of seconds.
 return seconds
```

```
Task Id: 608
#
# 1. Define a function "bell_Number"
def bell_Number(n):
 # 2. Initialize a list "bell" with n+1 rows and n+1 columns.
 bell = [[0 for i in range(n+1)] for j in range(n+1)]

 # 3. Set the first element of the list "bell" to 1.
 bell[0][0] = 1
 # 4. Fill the first column of the list "bell" with the first n+1 Bell numbers.
 for i in range(1, n+1):
  bell[i][0] = bell[i-1][i-1]
 # 5. Fill the rest of the list "bell" with the next n+1 Bell numbers.
 for i in range(1, n+1):
  for j in range(1, i+1):
   bell[i][j] = bell[i-1][j-1] + bell[i][j-1]

 # 6. Return the first element of the list "bell".
 return bell[n][0]
```

Self-planning Code Generation with Large Language Model

Xue Jiang, Yihong Dong, Lecheng Wang, Qiwei Shang, Ge Li*

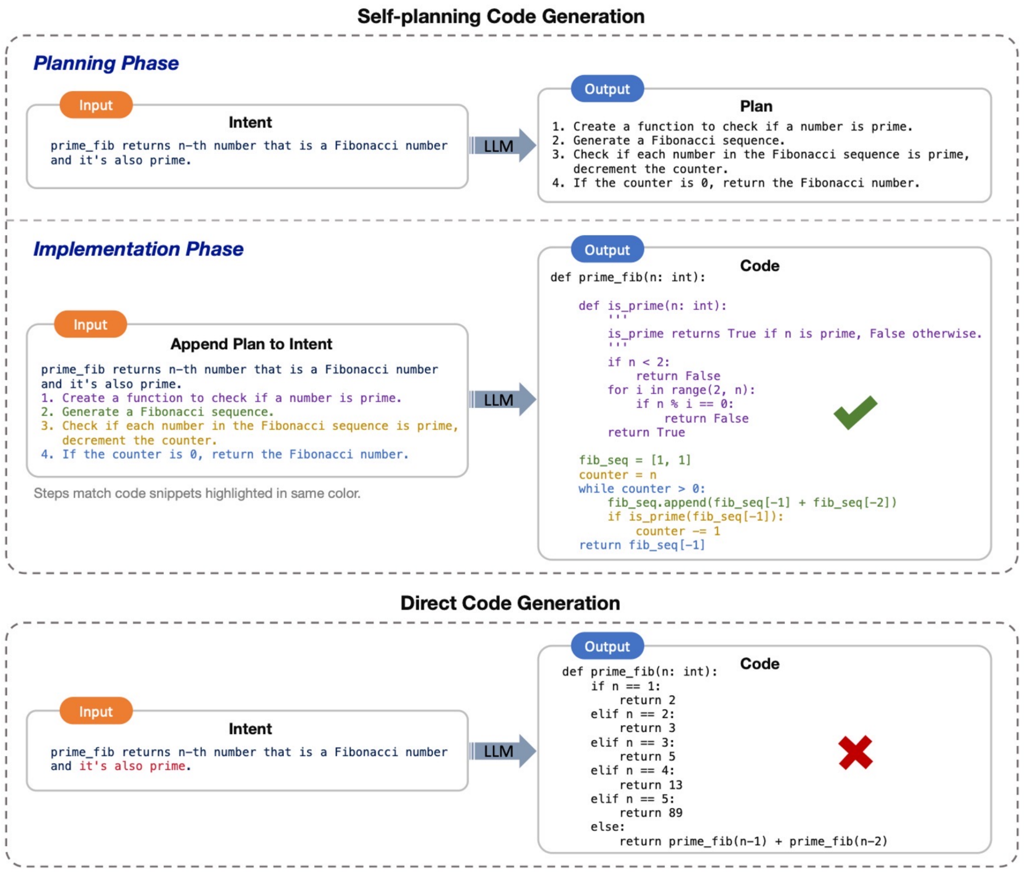


Figure 1: Self-planning code generation is carried out in two phases (i.e., planning phase and implementation phase): 1) In planning phase, LLM decomposes an intent into a set of easily solvable sub-problems and devises a plan for executing the solution steps; 2) In implementation phase, LLM generates code following the intent and plan, which assists self-planning code generation to be capable of handling more difficult problems than direct code generation.

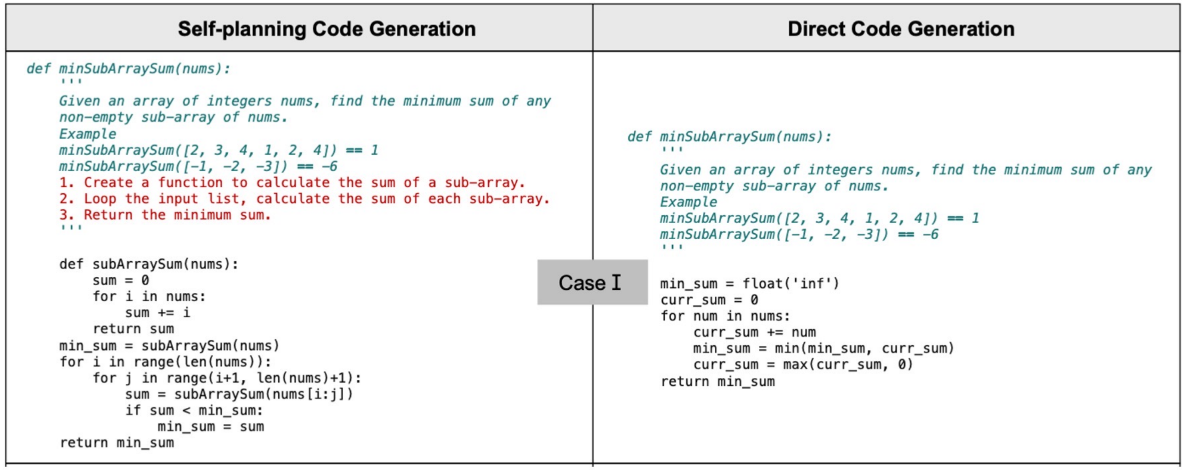


Figure 2: Three real cases from HumanEval with Self-planning and direct code generation. The input, generated plan, and code are highlighted in green, red, and black respectively.

Table 1: Comparison of self-planning and other methods.

Method	Pass@1	CodeBLEU	Pass@1
	HumanEval		HumanEval-ET
Direct	47.6	24.0	37.2
CoT	53.9	30.4	45.5
Self-planning	**60.3** (↑ 26.7%)	28.6	**46.2** (↑ 24.1%)
Ground Truth Planning	74.4	41.0	57.7
	MBPP-sanitized		MBPP-ET
Direct	49.9	25.6	37.7
CoT	54.7	26.4	39.6
Self-planning	**55.3** (↑ 10.9%)	24.9	**41.9** (↑ 11.2%)
Ground-truth Planning	65.1	33.7	50.7

AceCoder : Improving In-Context Learning for Code Generation

Jia Li, Kechi Zhang, Ge Li

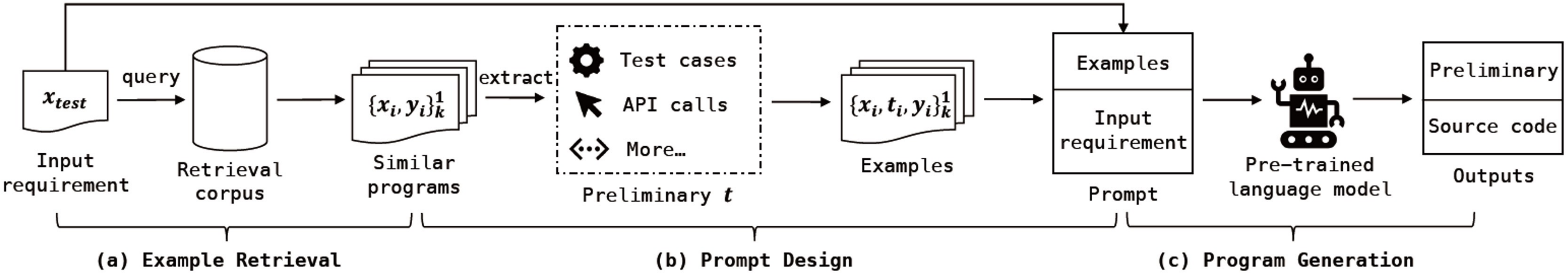


Table 2: The results of AceCoder and ICL-based baselines on three datasets. The values in parentheses are the relative improvements compared to standard ICL.

Base model	Approach	MBPP Pass@1 (%)	MBPP Pass@3 (%)	MBPP Pass@5 (%)	MBJP Pass@1 (%)	MBJP Pass@3 (%)	MBJP Pass@5 (%)	MBJSP Pass@1 (%)	MBJSP Pass@3 (%)	MBJSP Pass@5 (%)
Codex-175B	Zero-shot	25.00	44.20	51.40	14.00	29.61	38.54	24.95	44.42	49.13
	Standard ICL	37.20	54.40	60.60	43.00	58.82	64.91	31.23	46.04	50.71
	AceCoder	**47.40 (↑ 27.4%)**	**62.20 (↑ 14.3%)**	**66.40 (↑ 9.6%)**	**49.90 (↑ 16%)**	**61.26 (↑ 4.1%)**	**66.51 (↑ 2.5%)**	**41.38 (↑ 32.5%)**	**53.75 (↑ 16.8%)**	**58.42 (↑ 15.2%)**
CodeGeeX-13B	Zero-shot	5.20	13.80	19.40	4.46	11.97	18.26	0.20	0.20	0.41
	Standard ICL	20.40	30.60	36.00	16.63	26.17	34.48	11.16	19.88	25.56
	AceCoder	**25.40 (↑ 24.5%)**	**35.20 (↑ 15%)**	**40.00 (↑ 11.1%)**	**27.18 (↑ 63.4%)**	**35.90 (↑ 37.2%)**	**40.57 (↑ 17.7%)**	**20.06 (↑ 79.7%)**	**30.83 (↑ 55.1%)**	**35.70 (↑ 39.7%)**
CodeGen-6B	Zero-shot	10.40	19.40	24.40	14.81	25.76	31.44	8.72	19.67	22.92
	Standard ICL	14.60	24.00	30.20	18.25	30.02	34.68	9.94	19.88	23.12
	AceCoder	**21.60 (↑ 47.9%)**	**33.60 (↑ 40%)**	**39.00 (↑ 29.1%)**	**21.55 (↑ 18.1%)**	**33.48 (↑ 11.5%)**	**40.16 (↑ 15.8%)**	**15.64 (↑ 57.3%)**	**26.57 (↑ 33.7%)**	**31.42 (↑ 35.9%)**
InCoder-6B	Zero-shot	4.20	11.40	16.20	2.23	5.88	9.13	3.65	5.88	8.11
	Standard ICL	12.80	22.80	28.20	10.95	23.53	26.17	12.78	22.52	27.79
	AceCoder	**19.40 (↑ 51.6%)**	**30.40 (↑ 33.3%)**	**33.80 (↑ 19.9%)**	**15.82 (↑ 44.5%)**	**29.01 (↑ 23.3%)**	**34.08 (↑ 30.2%)**	**15.42 (↑ 20.7%)**	**26.77 (↑ 18.9%)**	**30.22 (↑ 8.7%)**

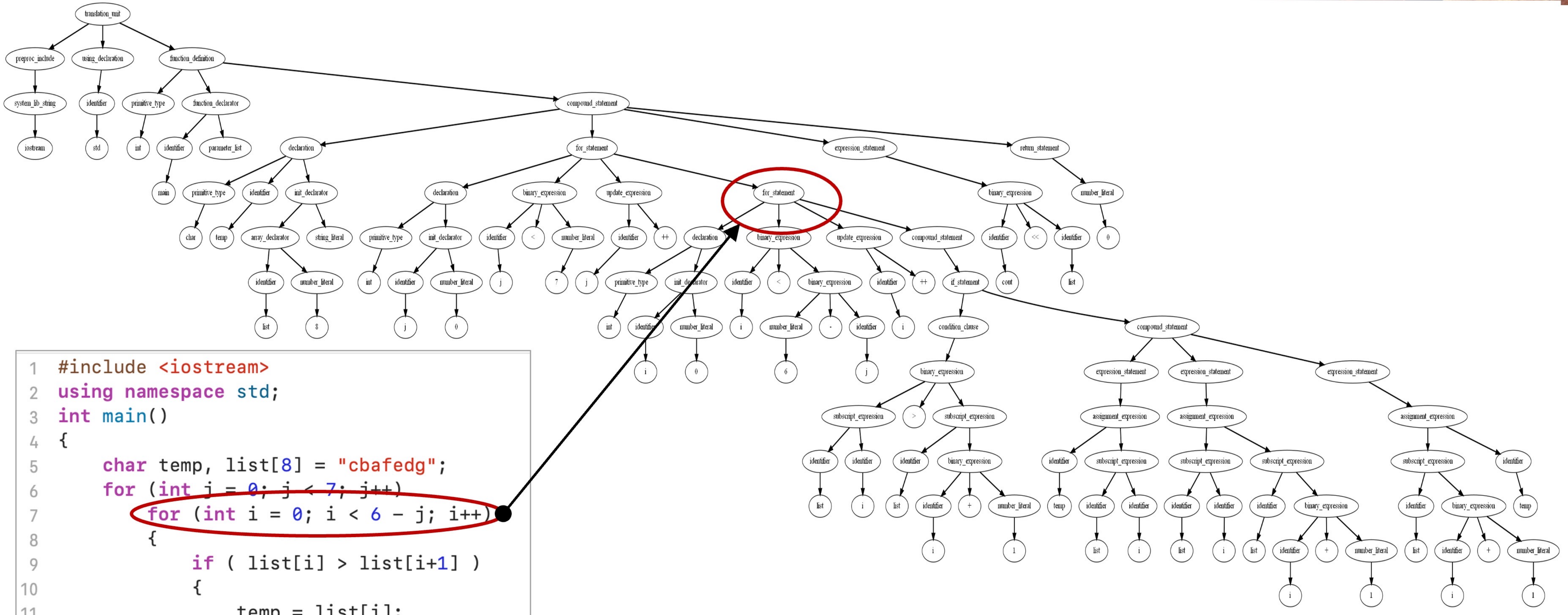
问题4：代码与自然语言有什么不同？

However,

Currently, people process code like they do with natural languages.

```
private static Object[] covertArray(Class<?> clazz, String vals[], int from, int to)
throws NoSuchMethodException, IllegalAccessException, InvocationTargetException
{ int start; int end; if (from > to) { start = to; end = from; } else { start = from;
end = to; } Object result[] = (Object[]) Array.newInstance(clazz, to - from); Method
valueOfMethod = clazz.getMethod("valueOf", new Class[] { String.class }); boolean
accessible = valueOfMethod.isAccessible(); valueOfMethod.setAccessible(true); if
(valueOfMethod != null) { for (int i = start; i < end; i++) { Object val =
valueOfMethod.invoke(clazz, new Object[] { vals[i] }); result[i - start] = val; } }
valueOfMethod.setAccessible(accessible); return result; }
```

（1）代码的结构性强于自然语言



```cpp
#include <iostream>
using namespace std;
int main()
{
    char temp, list[8] = "cbafedg";
    for (int j = 0; j < 7; j++)
        for (int i = 0; i < 6 - j; i++)
        {
            if ( list[i] > list[i+1] )
            {
                temp = list[i];
                list[i] = list[i+1];
                list[i+1] = temp;
            }
        }
    cout<<list;
    return 0;
}
```

- **代码可以映射为抽象语法树或具体语法树，具有严格的结构化特性，而当前的大模型通常依赖于代码的自然性进行建模。**

（2）代码的自然性弱于自然语言

- 表现为编程语言可以利用语法产生式进行严格定义，编写的代码的过程是在一个更加受限的语法空间中描述程序语义。

```python
def FordFulkerson(graph, source, sink):
    # This array is filled by BFS and to store path
    parent = [-1] * (len(graph))
    max_flow = 0
    while BFS(graph, source, sink, parent):
        path_flow = float("Inf")
        s = sink

        while s != source:
            # Find the minimum value in select path
            path_flow = min(path_flow, graph[parent[s]][s])
            s = parent[s]

        max_flow += path_flow
        v = sink

        while v != source:
            u = parent[v]
            graph[u][v] -= path_flow
            graph[v][u] += path_flow
            v = parent[v]
    return max_flow
```

is defined by

```
single_input: NEWLINE | simple_stmt | compound_stmt NEWLINE
file_input: (NEWLINE | stmt)* ENDMARKER
eval_input: testlist NEWLINE* ENDMARKER
decorator: '@' dotted_name [ '(' [arglist] ')' ] NEWLINE
decorators: decorator+
decorated: decorators (classdef | funcdef | async_funcdef)
async_funcdef: 'async' funcdef
funcdef: 'def' NAME parameters ['->' test] ':' suite
parameters: '(' [typedargslist] ')'
typedargslist: (tfpdef ['=' test] (',' tfpdef ['=' test])* [',' ['*' [tfpdef] (',' tfpdef ['=' test])* [',' ['**' tfpdef [',']]] | '**' tfpdef [',']]] | '*' [tfpdef] (',' tfpdef ['=' test])* [',' ['**' tfpdef [',']]] | '**' tfpdef [','])
tfpdef: NAME [':' test]
varargslist: (vfpdef ['=' test] (',' vfpdef ['=' test])* [',' ['*' [vfpdef] (',' vfpdef ['=' test])* [',' ['**' vfpdef [',']]] | '**' vfpdef [',']]] | '*' [vfpdef] (',' vfpdef ['=' test])* [',' ['**' vfpdef [',']]] | '**' vfpdef [','])
vfpdef: NAME
stmt: simple_stmt | compound_stmt
simple_stmt: small_stmt (';' small_stmt)* [';'] NEWLINE
small_stmt: (expr_stmt | del_stmt | pass_stmt | flow_stmt |import_stmt | global_stmt | nonlocal_stmt | assert_stmt)
expr_stmt: testlist_star_expr (annassign | augassign (yield_expr|testlist) | ('=' (yield_expr|testlist_star_expr))*)
aannaaien: ':' test ['=' test]
testlist_star_expr: (test|star_expr) (',' (test|star_expr))* [',']
augassign: ('+=' | '-=' | '*=' | '@=' | '/=' | '%=' | '&=' | '|=' | '^=' |'<<=' | '>>=' | '**=' | '//=')
del_stmt: 'del' exprlist
pass_stmt: 'pass'
flow_stmt: break_stmt | continue_stmt | return_stmt | raise_stmt | yield_stmt
break_stmt: 'break'
continue_stmt: 'continue'
return_stmt: 'return' [testlist]
yield_stmt: yield_expr
raise_stmt: 'raise' [test ['from' test]]
import_stmt: import_name | import_from
import_name: 'import' dotted_as_names
import_from: ('from' (('.' | '...')* dotted_name | ('.' | '...')+) 'import' ('*' | '(' import_as_names ')' | import_as_names))
import_as_name: NAME ['as' NAME]
dotted_as_name: dotted_name ['as' NAME]
import_as_names: import_as_name (',' import_as_name)* [',']
dotted_as_names: dotted_as_name (',' dotted_as_name)*
dotted_name: NAME ('.' NAME)*
global_stmt: 'global' NAME (',' NAME)*
nonlocal_stmt: 'nonlocal' NAME (',' NAME)*
assert_stmt: 'assert' test [',' test]
compound_stmt: if_stmt | while_stmt | for_stmt | try_stmt | with_stmt | funcdef | classdef | decorated | async_stmt
async_stmt: 'async' (funcdef | with_stmt | for_stmt)
if_stmt: 'if' test ':' suite ('elif' test ':' suite)* ['else' ':' suite]
while_stmt: 'while' test ':' suite ['else' ':' suite]
for_stmt: 'for' exprlist 'in' testlist ':' suite ['else' ':' suite]
try_stmt: ('try' ':' suite ((except_clause ':' suite)+ ['else' ':' suite] ['finally' ':' suite] |'finally' ':' suite))
with_stmt: 'with' with_item (',' with_item)*  ':' suite
with_item: test ['as' expr]
except_clause: 'except' [test ['as' NAME]]
suite: simple_stmt | NEWLINE INDENT stmt+ DEDENT
test: or_test ['if' or_test 'else' test] | lambdef
test_nocond: or_test | lambdef_nocond
lambdef: 'lambda' [varargslist] ':' test
lambdef_nocond: 'lambda' [varargslist] ':' test_nocond
or_test: and_test ('or' and_test)*
and_test: not_test ('and' not_test)*
not_test: 'not' not_test | comparison
comparison: expr (comp_op expr)*
comp_op: '<'|'>'|'=='|'>='|'<='|'<>'|'!='|'in'|'not' 'in'|'is'|'is' 'not'
star_expr: '*' expr
expr: xor_expr ('|' xor_expr)*
xor_expr: and_expr ('^' and_expr)*
and_expr: shift_expr ('&' shift_expr)*
shift_expr: arith_expr (('<<'|'>>') arith_expr)*
arith_expr: term (('+'|'-') term)*
term: factor (('*'|'@'|'/'|'%'|'//') factor)*
factor: ('+'|'-'|'~') factor | power
power: atom_expr ['**' factor]
atom_expr: ['await'] atom trailer*
atom: ('(' [yield_expr|testlist_comp] ')' |'[' [testlist_comp] ']' |'{' [dictorsetmaker] '}' |NAME | NUMBER | STRING+ | '...' | 'None' | 'True' | 'False')
testlist_comp: (test|star_expr) ( comp_for | (',' (test|star_expr))* [','] )
trailer: '(' [arglist] ')' | '[' subscriptlist ']' | '.' NAME
subscriptlist: subscript (',' subscript)* [',']
subscript: test | [test] ':' [test] [sliceop]
sliceop: ':' [test]
exprlist: (expr|star_expr) (',' (expr|star_expr))* [',']
testlist: test (',' test)* [',']
dictorsetmaker: ( ((test ':' test | '**' expr) (comp_for | (',' (test ':' test | '**' expr))* [','])) | ((test | star_expr) (comp_for | (',' (test | star_expr))* [','])) )
classdef: 'class' NAME ['(' [arglist] ')'] ':' suite
arglist: argument (',' argument)*  [',']
argument: ( test [comp_for] |test '=' test |'**' test |'*' test )
comp_iter: comp_for | comp_if
sync_comp_for: 'for' exprlist 'in' or_test [comp_iter]
comp_for: ['async'] sync_comp_for
comp_if: 'if' test_nocond [comp_iter]
encoding_decl: NAME
yield_expr: 'yield' [yield_arg]
yield_arg: 'from' test | testlist
```

（3）代码的局部性强于自然语言

- 表现为程序代码中的诸多成分的语义，重度依赖于所在的上下文环境，仅在该受限的上下文环境中才具备特定语义。

```
def FordFulkerson(graph, source, sink):
    # This array is filled by BFS and to store path
    parent = [-1] * (len(graph))
    max_flow = 0
    while BFS(graph, source, sink, parent):
        path_flow = float("Inf")
        s = sink

        while s != source:
            # Find the minimum value in select path
            path_flow = min(path_flow, graph[parent[s]][s])
            s = parent[s]

        max_flow += path_flow
        v = sink

        while v != source:
            u = parent[v]
            graph[u][v] -= path_flow
            graph[v][u] += path_flow
            v = parent[v]
    return max_flow
```

parent ?

graph ?

s ? v ? u ?

source ?

sink ?

n ?

temp ?

path_flow ?

（4）代码的环境依赖性强于自然语言

- 表现为代码多数依附于已有的软件架构进行编写，代码的语义通常是在特定的软件框架环境中的表达。

```python
def multilayer_perceptron(x):
    # Hidden fully connected layer with 256 neurons
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    # Hidden fully connected layer with 256 neurons
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    # Output fully connected layer with a neuron for each class
    out_layer = tf.matmul(layer_2, weights['out']) + biases['out']
    return out_layer

# Construct model
logits = multilayer_perceptron(X)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)
# Initializing the variables
init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
```

```python
tf.add(tf.matmul(x, weights['h1']), biases['b1'])
```
?

```python
tf.matmul(layer_2, weights['out']) + biases['out']
```
?

```python
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
```
?

```python
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)
```
?

GPT-3.5

GPT-3.5 models can understand and generate natural language or code. Our most capable and cost effective model in the GPT-3.5 family is `gpt-3.5-turbo` which has been optimized for chat but works well for traditional completions tasks as well.

LATEST MODEL	DESCRIPTION	MAX TOKENS	TRAINING DATA
`gpt-3.5-turbo`	Most capable GPT-3.5 model and optimized for chat at 1/10th the cost of `text-davinci-003`. Will be updated with our latest model iteration.	4,096 tokens	Up to Sep 2021
`gpt-3.5-turbo-0301`	Snapshot of `gpt-3.5-turbo` from March 1st 2023. Unlike `gpt-3.5-turbo`, this model will not receive updates, and will be deprecated 3 months after a new version is released.	4,096 tokens	Up to Sep 2021
`text-davinci-003`	Can do any language task with better quality, longer output, and consistent instruction-following than the curie, babbage, or ada models. Also supports inserting completions within text.	4,097 tokens	Up to Jun 2021
`text-davinci-002`	Similar capabilities to `text-davinci-003` but trained with supervised fine-tuning instead of reinforcement learning	4,097 tokens	Up to Jun 2021
`code-davinci-002`	Optimized for code-completion tasks	8,001 tokens	Up to Jun 2021

问题5：代码是否也应该用语言模型建模？

Program can be

- a Token Sequence.
- a Tree Structure.
- a Graph.
  - Data flow Graph
  - Control flow Graph
  - Call Graph
  - ……
- an Operation sequence.
- an Production sequence.

问题5+：现有程序表示技术能否用于大模型？

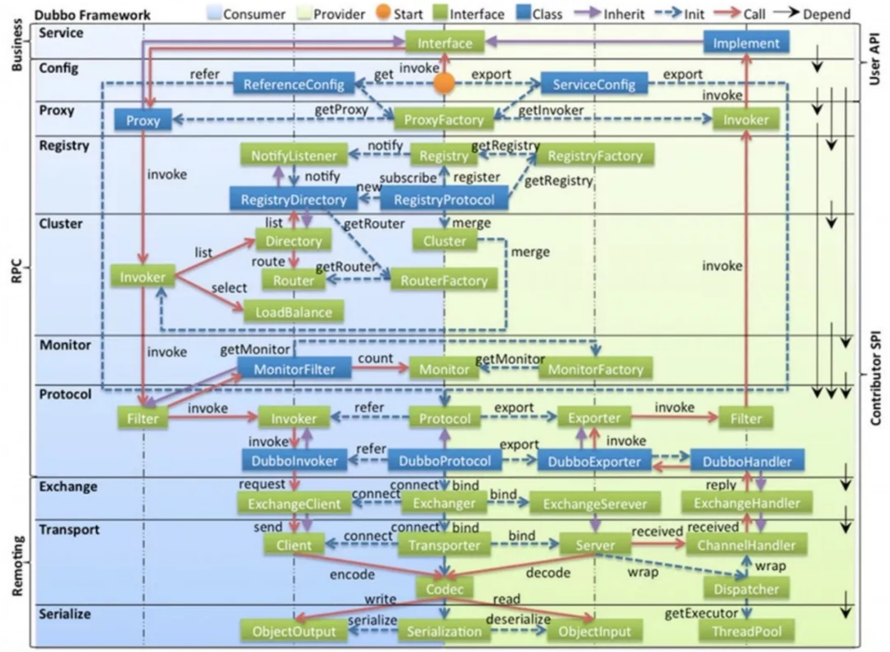
Can we make use of different representations of programs to build BIG MODELS?

—— Our research has confirmed that these approaches are effective in small models.


问题6：当前大模型在软件开发中的应用存在哪些问题？

（1）大模型与领域知识如何结合？

Big Model **+**

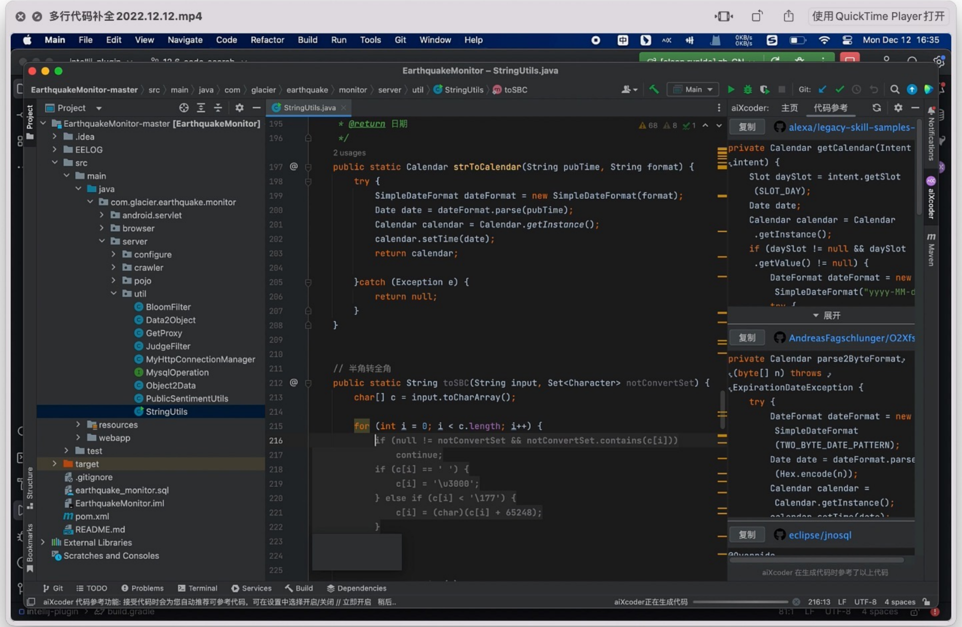


—— Software development usually depends on personalized private software development framework.

How can we let the big model learn domain knowledge?

（2）复杂上下文如何与大模型交互？

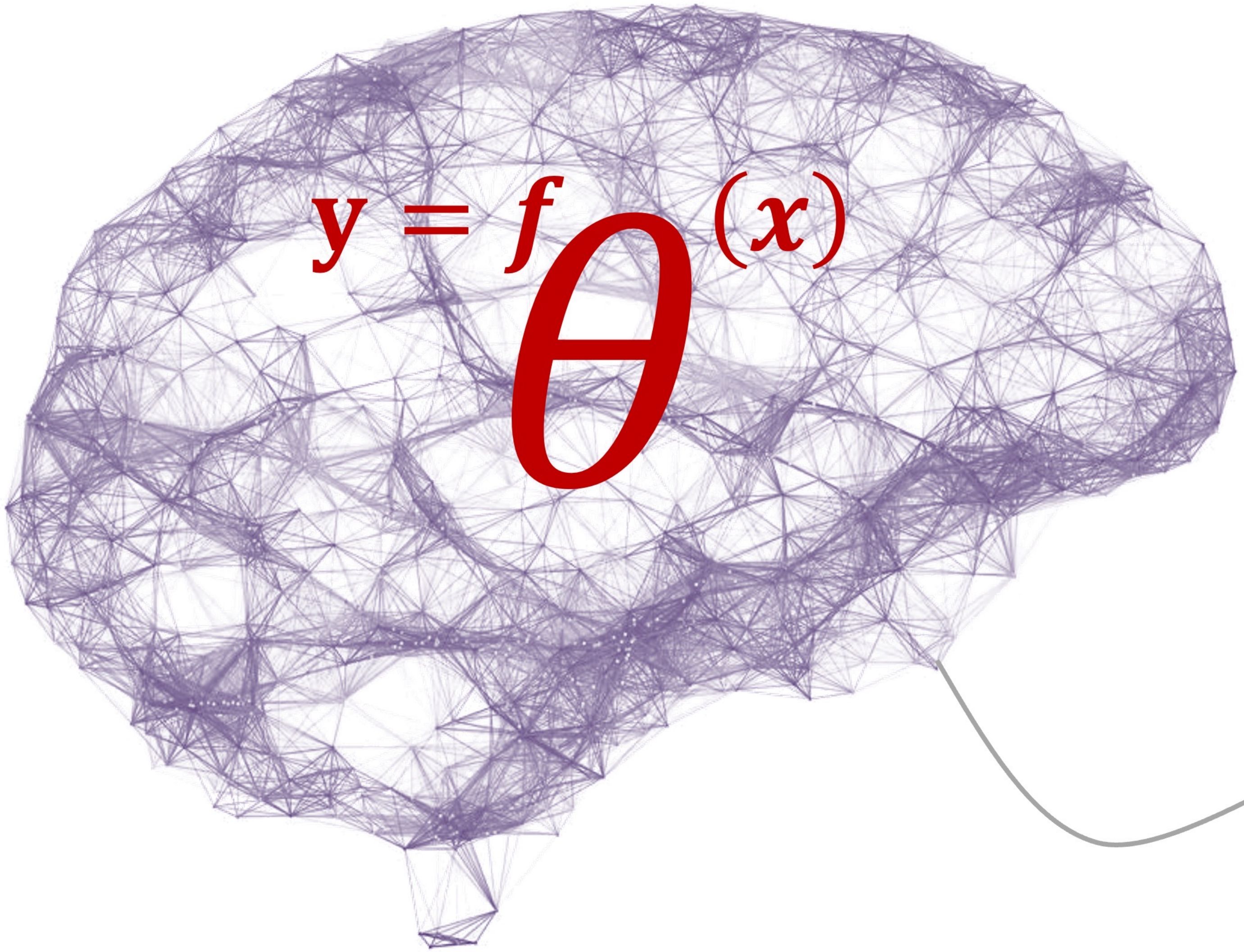
Big Model +



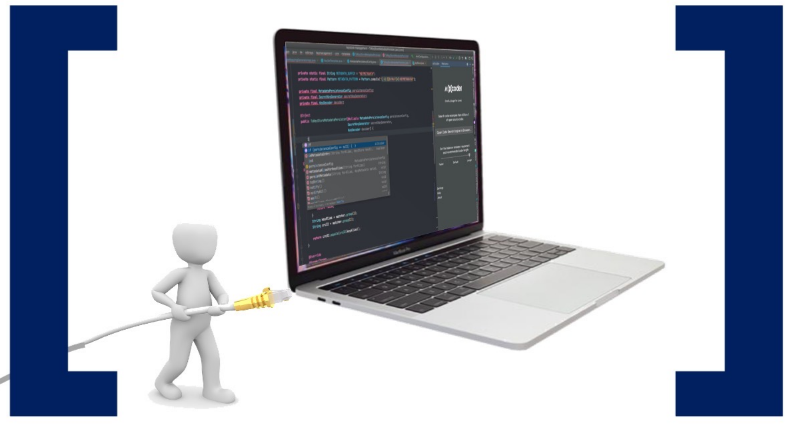
—— Software development happens in a complex environment, and it is difficult to transfer all the things to a big model.

—— What we can do is pull the big model into this complex environment to play its role.

（3）如何保护代码的知识产权？



$$y = f_{\theta}(x)$$



问题7：大模型对软件开发有何影响？

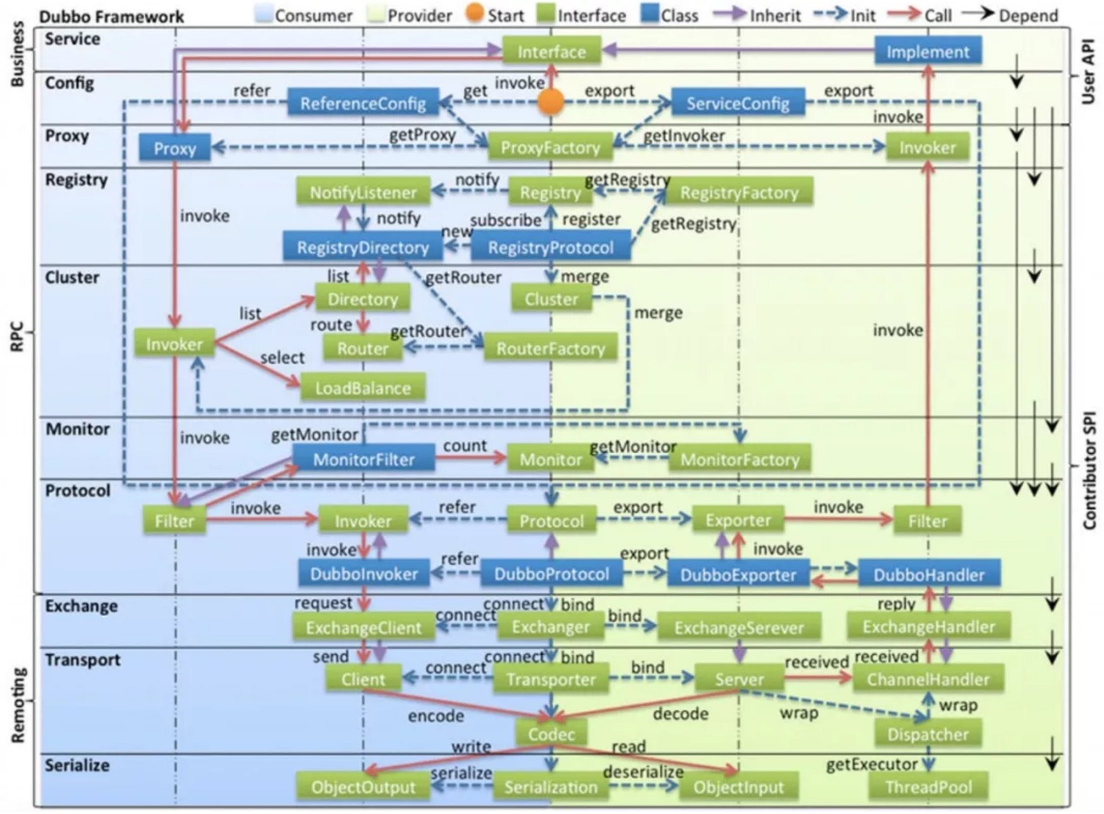
编程 ≠ 软件开发

```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

```python
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

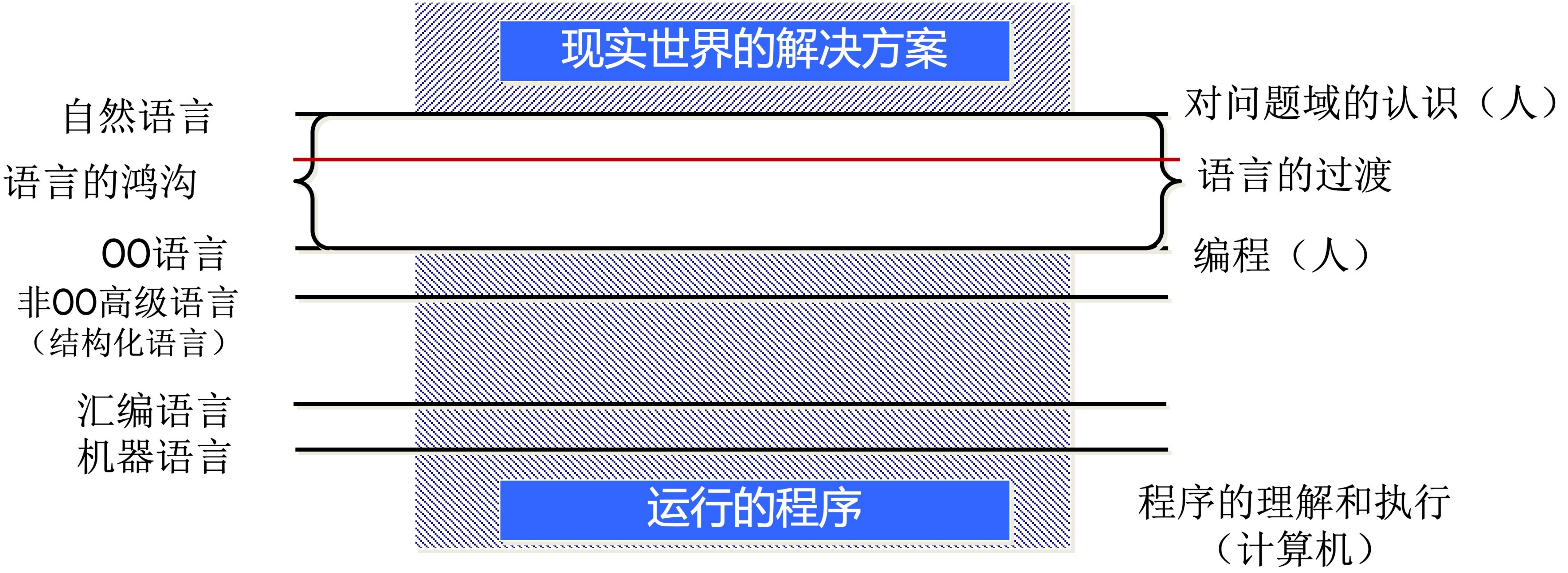
```python
def encode_cyclic(s: str):
    """
    returns encoded string by cycling groups of three characters.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group. Unless group has fewer elements than 3.
    groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups]
    return "".join(groups)

def decode_cyclic(s: str):
    """
    takes as input string encoded with encode_cyclic function. Returns decoded string.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group.
    groups = [(group[-1] + group[:-1]) if len(group) == 3 else group for group in groups]
    return "".join(groups)
```

vs.



问题8：大模型对软件开发有何影响？

Maybe, the advanced programming language will become today's assembly language.



And, what will happen if programs can be generated automatically?

——

Automated requirement analysis will be required.
Automated testing will be required.
Automatic repair will be required.
Automated defect detection will be required.
……

in the end

The big model has shown us the dawn of software development automation, but we are also convinced that there is still a long way to go in the future.

THANKS