Final Project Submission

Learning a Code Language Model using Self-Supervised Learning (SSL) for Code

Team Number- 22

Team Members	**Roll Numbers**
Nikhil Singh	2024201067
Rajath Gadagkar	2024201035
Abhradeep Das	2024202018
Sanket Adlak	2024204005

Software System Development (CS6.302)

Master of Technology,

International Institute of Information Technology, Hyderabad

Monsoon 2024

Table of Content

1. About the Project

Learning a Code Language using Self-Supervised Learning (SSL) for Code is a project which is aimed at enhancing the productivity of software developers by providing a sophisticated code suggestion tool that integrates seamlessly with modern Integrated Development Environments (IDEs). By leveraging **Self-Supervised Learning (SSL)**, this system predicts the next segment or token of code based on the context of the user's input, enabling real-time, context-aware code completion.

Key Objectives

1. **Improved Developer Productivity:** Reduce the time and effort spent on writing repetitive or boilerplate code by offering intelligent, context-sensitive suggestions.
2. **Language Support:** It only supports programming languages - Python.
3. **Real-Time Performance:** Maintain low-latency interactions for a seamless user experience, even when working with large-scale projects.

Core Features

- **Real-Time Code Suggestions:**
    - As the developer types code in the editor, the system predicts the next tokens, helping to complete lines of code or suggest potential constructs.
    - Includes support for language-specific constructs, syntax, and patterns.
- **Wide Dataset Coverage:**
    - Utilizes open-source repositories to create a large, diverse training corpus, ensuring the model learns from various coding styles and best practices.

Project Scope

The project includes the design, development, and deployment of:

- A **frontend interface** for user interaction, providing real-time code completion suggestions.

- A **backend inference service** powered by SSL-based Transformer models for predictions.
- A modular training pipeline for easy experimentation with different model architectures, such as Transformer-XL, LSTM, and RNN.

Assumptions

1. **Open-Source Data Availability:** The availability of open-source repositories is sufficient for creating a robust training corpus.
2. **Real-Time Requirements:** Low-latency performance is critical for providing a seamless user experience.
3. **User Feedback as a Learning Loop:** Continuous feedback from developers can help refine the system over time, making it more accurate and context-aware.
4. **Model Extensibility:** The modular design of the system will allow for easy integration of additional languages or features in the future.

Relevance

- **Industry Impact:** Code completion tools are vital in software development, and this project aligns with the industry's demand for AI-assisted coding tools.
- **Research Contribution:** Combines SSL techniques with Transformer models, providing valuable insights into their application in real-world coding scenarios.
- **Developer Support:** Facilitates the adoption of better coding practices by offering intelligent and optimized code suggestions.

2. Technology Stack

Backend

- **Flask**: A lightweight Python web framework used to develop the REST API that handles autocompletion requests from the frontend (VS Code extension). It manages HTTP POST requests to return predictions generated by the model.
- **Python**: The primary language for backend development and model integration.
- **JSON**: Used for structured data exchange between the VS Code extension and the Flask backend, enabling efficient communication of autocompletion requests and responses.
- **labml**: A tool for experiment tracking and monitoring, used for logging and tracking the performance of the model during inference.
- **labml_nn**: A library used for building and training neural networks, including utilities for transformer models like Transformer-XL.
- **torch**: The core PyTorch library used for defining, training, and using neural network models.

Frontend

- **VS Code Extension:**
  - **TypeScript**: The language used to develop the VS Code extension, which integrates seamlessly with the editor and the backend services.
  - **VS Code API**: The extension leverages the VS Code API to provide autocomplete functionality within the IDE, enabling real-time code suggestions to the user.
  - **JavaScript**: Used in the VS Code extension to manage asynchronous communication with the backend and process model responses.
  - **Node.js**: Provides the runtime environment for executing the VS Code extension's code, handling the event-driven communication between the frontend and backend.

Model

- **Transformer-XL**: A transformer-based architecture that is designed to handle long-range dependencies in sequential data, ideal for tasks like code autocompletion. This model is key to predicting the next word based on the context provided.
- **Autoregressive Model**: An autoregressive approach is used by the model, where predictions are made one word at a time based on previous context, making it suitable for natural language and code generation tasks.

3. Control Flow and Data Flow

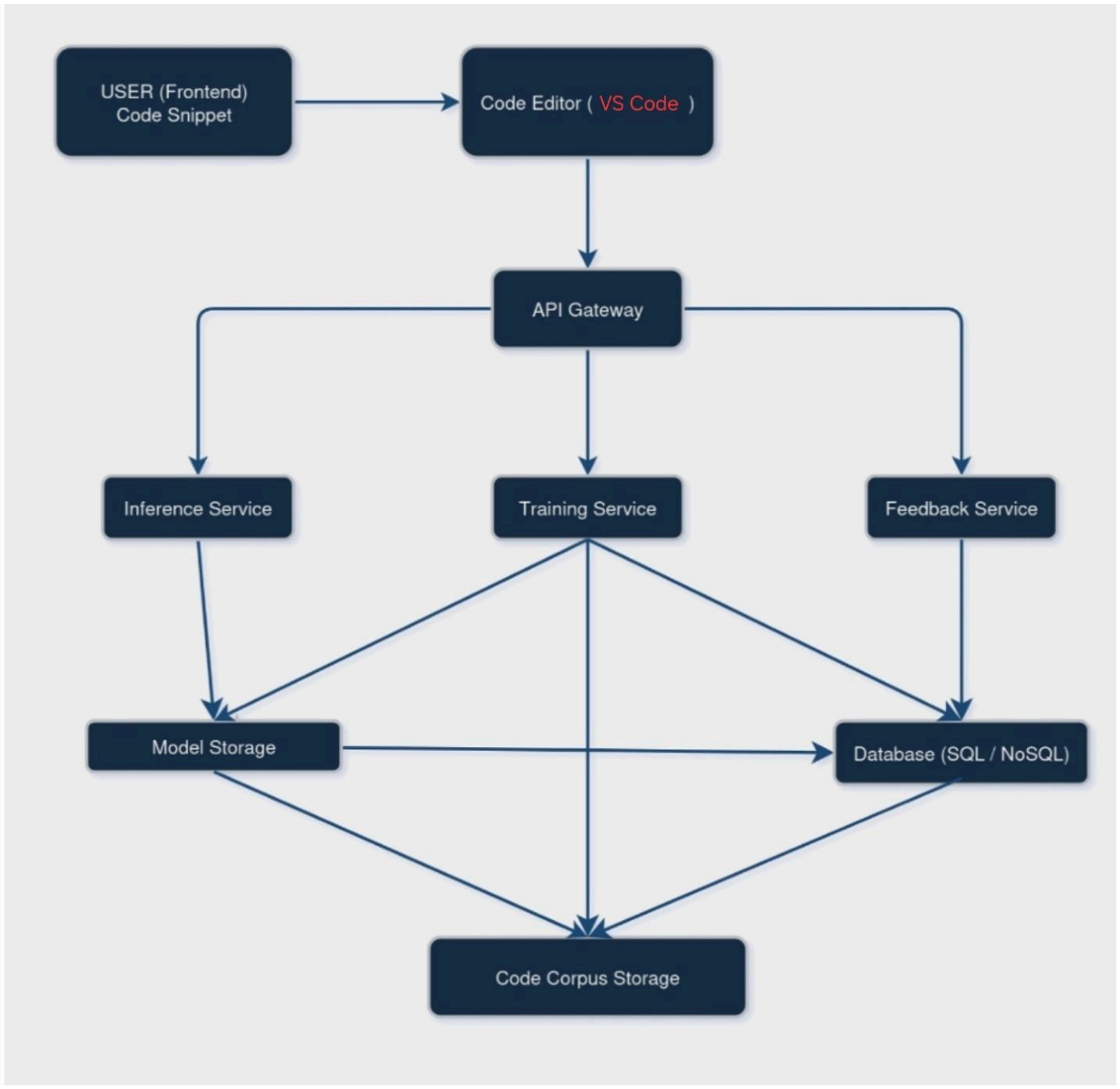


Fig. High-level architectural design

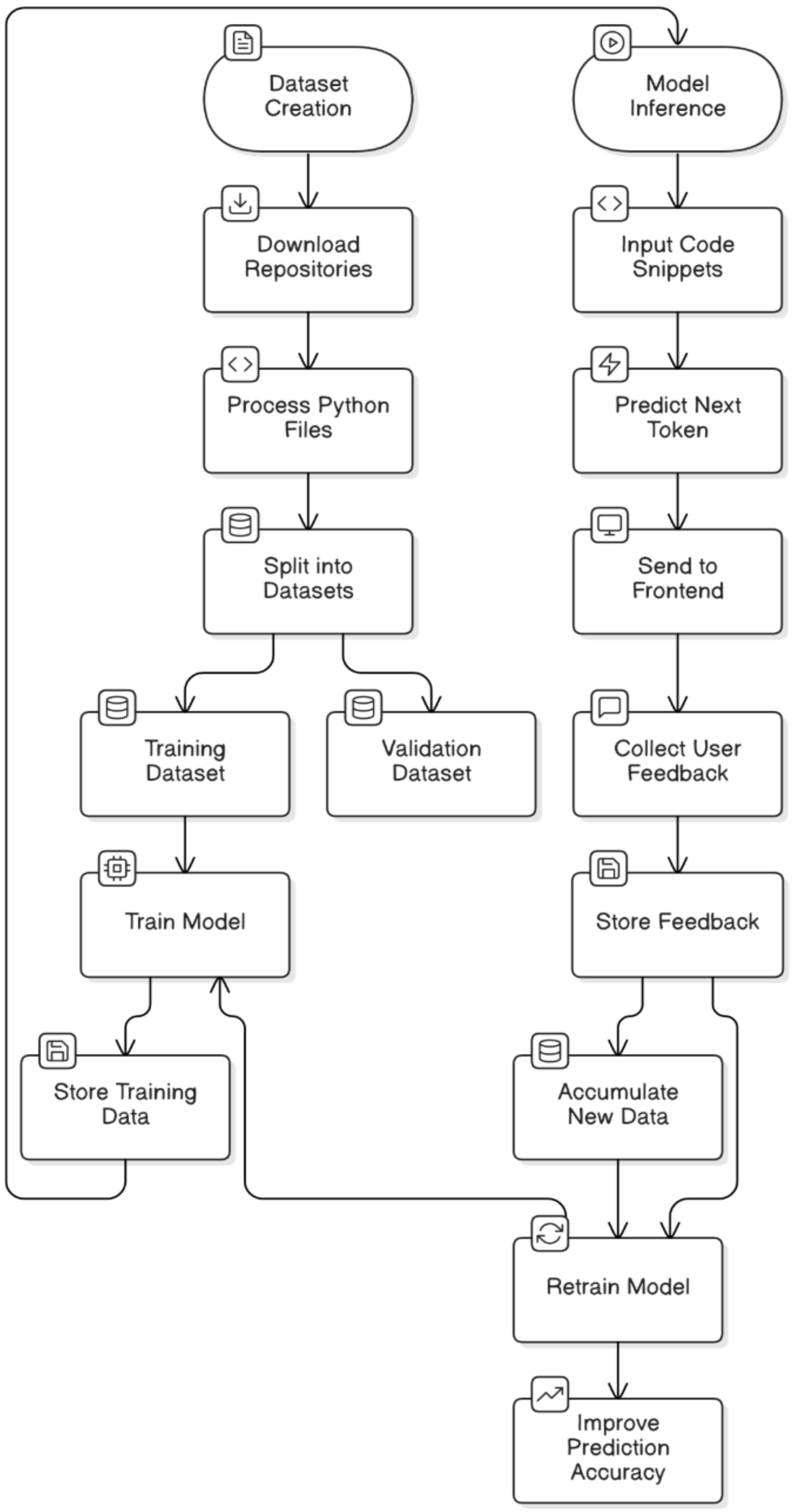


Fig. Data flow diagram

4. Code Directory Structure

This section represents a Python code autocomplete system using deep learning.

A. Training File:

```
train.py
```

- Main training script that defines the training configurations and model architectures
- Implements various model options like LSTM, RHN, Transformer models
- Contains training loop and evaluation logic

B. Model Implementations (./models/):

```
models/
├── __init__.py
├── highway.py
├── lstm.py
├── transformer.py
├── xl.py
```

- Contains different model architectures
- Each file implements a specific architecture (LSTM, Highway Networks, Transformer etc.)

C. Dataset Handling (./dataset/):

```
dataset/
├── __init__.py
├── bpe.py
├── break_words.py
├── dataset.py
```

- Dataset processing and loading utilities
- Implements tokenization, BPE (Byte Pair Encoding)
- Text preprocessing functionality

D. Evaluation (./evaluate/) :

```
evaluate/
├── __init__.py
├── anomalies.py
├── beam_search.py
├── beam_search_lengthy.py
├── eval_sample.py
├── factory.py
├── generate.py
```

- Contains evaluation and inference code
- Implements beam search for text generation
- Has anomaly detection and evaluation metrics

E. Serving (./server.py):

- Flask server implementation for model serving
- Provides REST API endpoints for predictions

F. Project Configuration files:

```
requirements.txt - Python dependencies
setup.py - Package installation configuration
Makefile - Build and deployment commands
.labml.yaml - Lab ML configuration
```

5. Source Code

- **GitHub**: SSL-For-Code

6. Video Link

- **OneDrive:** Learning a Code Language Model using Self-Supervised Learning (SSL) for Code

7. Project Evolution

The project evolved from system design and dataset preparation (Phase 1), to model training and UI development (Phase 2), culminating in a VSCode extension offering real-time code autocompletion. The final phase integrated model inference optimization, feedback loops, and extensive testing for a smooth, user-friendly experience.

Phase	Key Milestones	Key Differences
Phase 1	Research, system design, dataset selection, and architecture.	Focused on planning and research. No model training or real-time system integration.
Phase 2	Dataset preparation, basic model training, frontend UI design.	Model training pipeline setup, but no real-time integration with user interface.
Phase 3	VSCode extension creation, real-time code autocompletion.	Complete model integration into a working system (VSCode extension). Model optimization for real-time predictions.

8. Team Contributions:

1. **Nikhil Singh (2024201067)**
   - **Role**: Team Lead / Backend Developer
   - **Contributions**:
     - **Model Integration & Optimization**: Integrated and fine-tuned the autocompletion model (e.g., Transformer-XL), ensuring high-quality predictions with long-range context handling.
     - **API Development**: Developed the Flask-based API to serve the model's predictions to the VS Code extension, handling HTTP POST requests for autocompletion.
     - **Prediction Flow**: Contributed to the backend's logic for processing and sending code completions using NextWordPredictionComplete and get_predictor.
     - **Multithreading and Concurrency**: Implemented thread-safe operations with threading.Lock() to manage concurrent requests efficiently.
     - **Performance Monitoring**: Integrated performance tracking to optimize prediction speed and model accuracy.
     - **Model Training & Testing**: Conducted model training, experimentation, and evaluation to refine the autocompletion accuracy.
     - **API Optimization**: Improved server-side performance, reducing response times for real-time predictions.
     - **Backend Testing**: Performed unit and integration tests to ensure the reliability of the backend system.

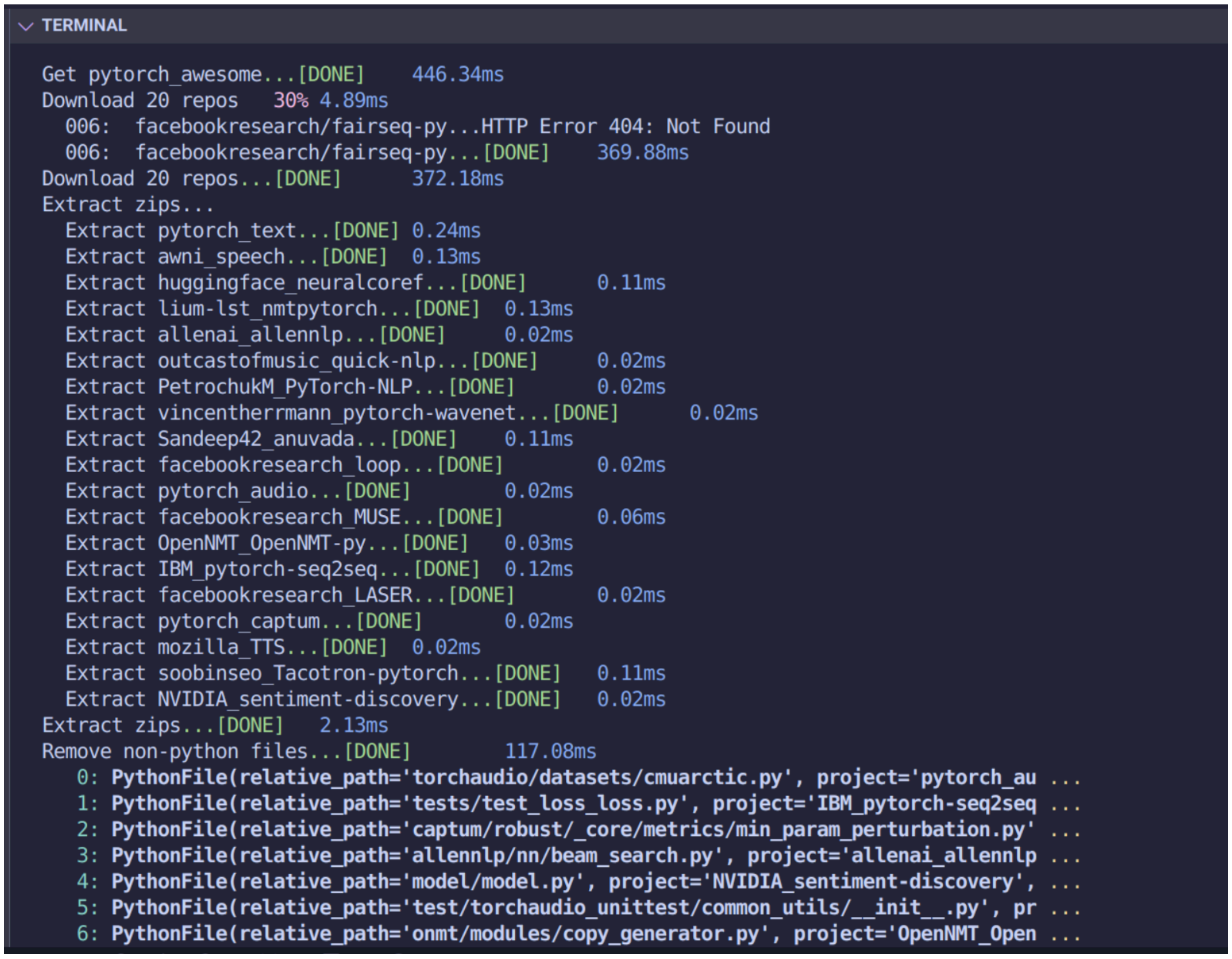
2. **Rajath Gadagkar (2024201035)**
   - **Contributions**:
     - **VS Code Extension**: Developed the VS Code extension in TypeScript, fetching autocompletion results from the backend API.
     - **Autocomplete UI**: Designed the autocomplete UI, ensuring proper display of code suggestions in the editor.
     - **Prediction Filtering**: Implemented logic to remove newlines, trim results, and filter out duplicates from the predictions.
     - **Error Handling**: Added error handling to manage API failures and ensure fallback mechanisms for the extension.
     - **User Interface Enhancements**: Refined the completion dropdown, optimizing the user experience.

3. **Abhradeep Das (2024202018)**
   - **Contributions**:
     - **Model Selection**: Researched and selected Transformer-XL as the autocompletion model, testing alternatives.
     - **Data Preprocessing**: Preprocessed the code corpus for training, ensuring clean and structured input.
     - **Model Evaluation**: Evaluated model performance, tuning hyperparameters for optimal code completion results.
     - **Training Support**: Assisted in training the model and analyzing its predictions.

4. **Sanket Adlak (2024204005)**
   - **Contributions**:
     - **Server Architecture**: Designed the Flask server to handle multiple requests efficiently, ensuring scalability.
     - **API Optimization**: Worked on API optimization for faster response times.
     - **Concurrency Management**: Enhanced thread management using locks to ensure the backend can handle simultaneous requests.
     - **Data Preprocessing**: Preprocessed the code corpus for training, ensuring clean and structured input.

9. Working Results:

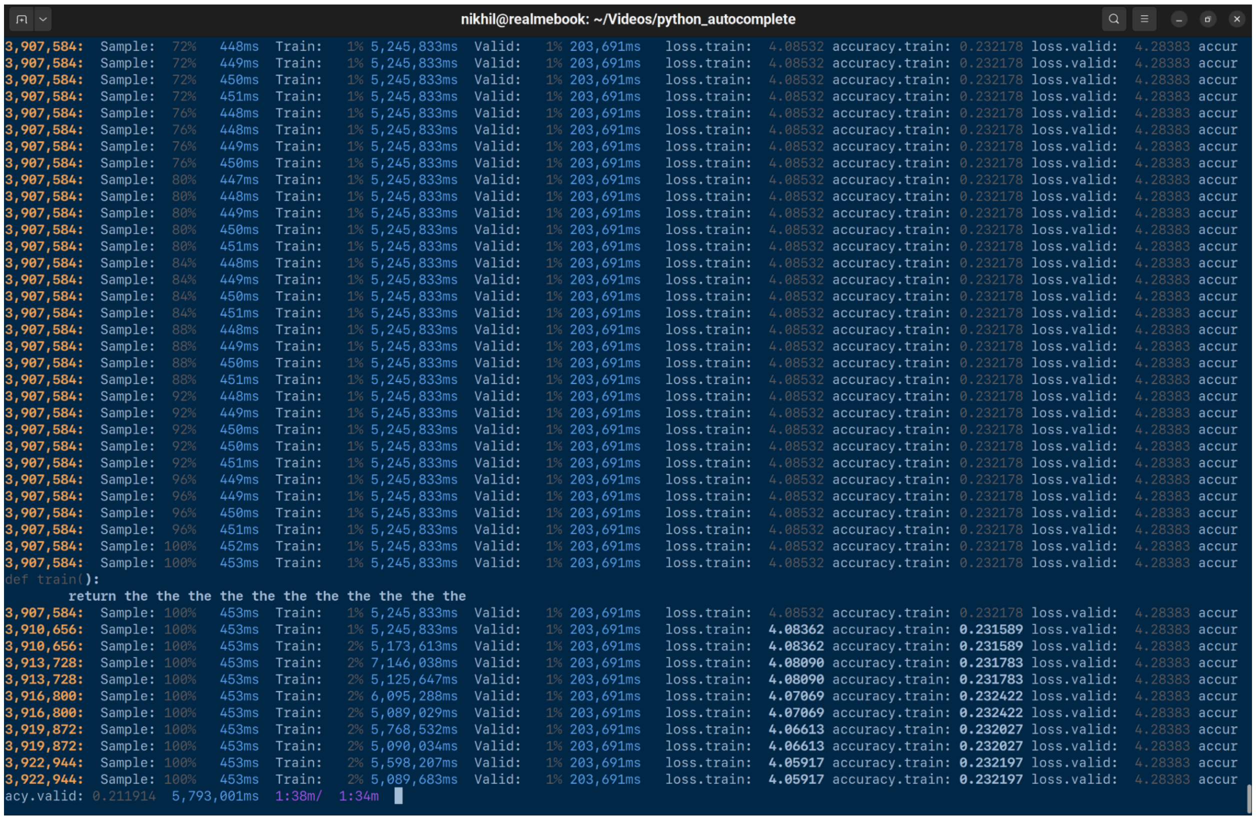
- **Preprocessing phase**



TERMINAL

```
Get pytorch_awesome...[DONE]    446.34ms
Download 20 repos   30% 4.89ms
  006:  facebookresearch/fairseq-py...HTTP Error 404: Not Found
  006:  facebookresearch/fairseq-py...[DONE]    369.88ms
Download 20 repos...[DONE]      372.18ms
Extract zips...
  Extract pytorch_text...[DONE] 0.24ms
  Extract awni_speech...[DONE]  0.13ms
  Extract huggingface_neuralcoref...[DONE]      0.11ms
  Extract lium-lst_nmtpytorch...[DONE]  0.13ms
  Extract allenai_allennlp...[DONE]     0.02ms
  Extract outcastofmusic_quick-nlp...[DONE]     0.02ms
  Extract PetrochukM_PyTorch-NLP...[DONE]       0.02ms
  Extract vincentherrmann_pytorch-wavenet...[DONE]      0.02ms
  Extract Sandeep42_anuvada...[DONE]    0.11ms
  Extract facebookresearch_loop...[DONE]        0.02ms
  Extract pytorch_audio...[DONE]        0.02ms
  Extract facebookresearch_MUSE...[DONE]        0.06ms
  Extract OpenNMT_OpenNMT-py...[DONE]   0.03ms
  Extract IBM_pytorch-seq2seq...[DONE]  0.12ms
  Extract facebookresearch_LASER...[DONE]       0.02ms
  Extract pytorch_captum...[DONE]       0.02ms
  Extract mozilla_TTS...[DONE]  0.02ms
  Extract soobinseo_Tacotron-pytorch...[DONE]   0.11ms
  Extract NVIDIA_sentiment-discovery...[DONE]   0.02ms
Extract zips...[DONE]   2.13ms
Remove non-python files...[DONE]        117.08ms
   0: PythonFile(relative_path='torchaudio/datasets/cmuarctic.py', project='pytorch_au ...
   1: PythonFile(relative_path='tests/test_loss_loss.py', project='IBM_pytorch-seq2seq ...
   2: PythonFile(relative_path='captum/robust/_core/metrics/min_param_perturbation.py' ...
   3: PythonFile(relative_path='allennlp/nn/beam_search.py', project='allenai_allennlp ...
   4: PythonFile(relative_path='model/model.py', project='NVIDIA_sentiment-discovery', ...
   5: PythonFile(relative_path='test/torchaudio_unittest/common_utils/__init__.py', pr ...
   6: PythonFile(relative_path='onmt/modules/copy_generator.py', project='OpenNMT_Open ...
```

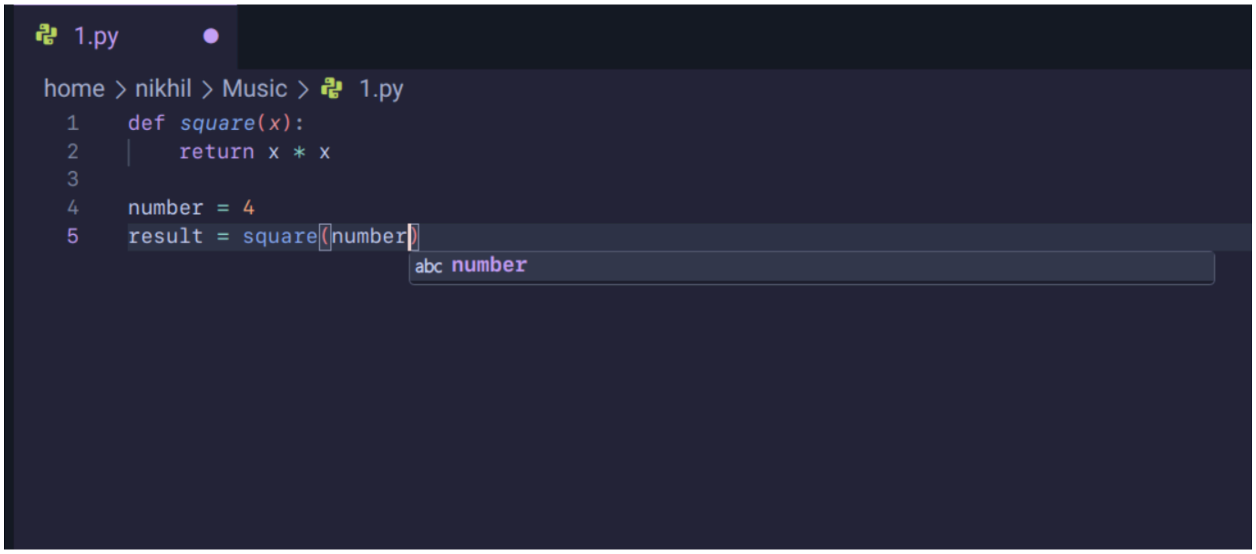
- Training phase



nikhil@realmebook: ~/Videos/python_autocomplete

```
3,907,584:  Sample:  72%  448ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  72%  449ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  72%  450ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  72%  451ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  76%  448ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  76%  448ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  76%  449ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  76%  450ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  80%  447ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  80%  448ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  80%  449ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  80%  450ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  80%  451ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  84%  448ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  84%  449ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  84%  450ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  84%  451ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  88%  448ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  88%  449ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  88%  450ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  88%  451ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  92%  448ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  92%  449ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  92%  450ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  92%  450ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  92%  451ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  96%  449ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  96%  449ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  96%  450ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample:  96%  451ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample: 100%  452ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,907,584:  Sample: 100%  453ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
def train():
        return the the the the the the the the the the the
3,907,584:  Sample: 100%  453ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08532 accuracy.train: 0.232178 loss.valid:  4.28383 accur
3,910,656:  Sample: 100%  453ms  Train:   1% 5,245,833ms  Valid:   1% 203,691ms   loss.train:  4.08362 accuracy.train: 0.231589 loss.valid:  4.28383 accur
3,910,656:  Sample: 100%  453ms  Train:   2% 5,173,613ms  Valid:   1% 203,691ms   loss.train:  4.08362 accuracy.train: 0.231589 loss.valid:  4.28383 accur
3,913,728:  Sample: 100%  453ms  Train:   2% 7,146,038ms  Valid:   1% 203,691ms   loss.train:  4.08090 accuracy.train: 0.231783 loss.valid:  4.28383 accur
3,913,728:  Sample: 100%  453ms  Train:   2% 5,125,647ms  Valid:   1% 203,691ms   loss.train:  4.08090 accuracy.train: 0.231783 loss.valid:  4.28383 accur
3,916,800:  Sample: 100%  453ms  Train:   2% 6,095,288ms  Valid:   1% 203,691ms   loss.train:  4.07069 accuracy.train: 0.232422 loss.valid:  4.28383 accur
3,916,800:  Sample: 100%  453ms  Train:   2% 5,089,029ms  Valid:   1% 203,691ms   loss.train:  4.07069 accuracy.train: 0.232422 loss.valid:  4.28383 accur
3,919,872:  Sample: 100%  453ms  Train:   2% 5,768,532ms  Valid:   1% 203,691ms   loss.train:  4.06613 accuracy.train: 0.232027 loss.valid:  4.28383 accur
3,919,872:  Sample: 100%  453ms  Train:   2% 5,090,034ms  Valid:   1% 203,691ms   loss.train:  4.06613 accuracy.train: 0.232027 loss.valid:  4.28383 accur
3,922,944:  Sample: 100%  453ms  Train:   2% 5,598,207ms  Valid:   1% 203,691ms   loss.train:  4.05917 accuracy.train: 0.232197 loss.valid:  4.28383 accur
3,922,944:  Sample: 100%  453ms  Train:   2% 5,089,683ms  Valid:   1% 203,691ms   loss.train:  4.05917 accuracy.train: 0.232197 loss.valid:  4.28383 accur
acy.valid: 0.211914  5,793,001ms  1:38m/  1:34m
```

- Prediction phase



```python
def square(x):
    return x * x

number = 4
result = square(number)
```

abc number

10. References

1. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., . . . Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1912.01703
2. Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-XL: Attentive Language models beyond a Fixed-Length context. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1901.02860
3. Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D Nonlinear Phenomena*, *404*, 132306. https://doi.org/10.1016/j.physd.2019.132306
4. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
5. Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Highway networks. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1505.00387