Literature review :

In the NLP domain, models based on Transformer architectures have shown state-of-the-art performance on a broad set of NLP tasks. These models can be categorized into three groups: encoder-only models such as BERT, RoBERTa, and ELECTRA, decoder-only models like GPT, and encoder-decoder models such as MASS, BART, and T5.

On the other hand, pre-training on the programming language is a nascent field where recent work attempts to extend the NLP pre-training methods to source code. Two pioneering models in this field are CuBERT and CodeBERT. CuBERT employs BERT's powerful masked language modeling objective to derive generic code-specific representation, and CodeBERT further adds a replaced token detection task to learn NL-PL cross-modal representation.

Despite these advancements, there are still challenges in the field of autonomous code summarization. One of the main challenges is the complexity of code and the difficulty of understanding the semantics of different programming constructs.

Approaches for Autonomous Code Summarization:

Encoder-Decoder Framework: As described earlier, this is a common approach where the code is encoded into a hidden space and then decoded into natural language space. However, this approach may suffer from the limitation of considering only the sequential content of the code, ignoring the tree structure which is also critical for code summarization.

Graph Neural Networks (GNN): Some studies propose using Graph Neural Networks to capture the structural information of the Abstract Syntax Tree (AST) of the code. This approach can handle the hierarchical nature of the code and might perform better in capturing the semantic meaning of the code.

Tree-LSTM: Another approach is to use Tree-LSTM, which can handle the hierarchical structure of the code more effectively. This approach treats each node in the AST as a time step in the LSTM, allowing the model to understand the code in a more structured manner.

Neural Attention Models: Some models utilize neural attention mechanisms to focus on important parts of the code when generating the summary.

This approach is based on using an attention mechanism to assign weights to different parts of the code based on their importance.

Implementation Strategy (using CodeT5) :

For a given programming language generate its AST.
Now we need to pass the root of AST tree to a summarize_function
We initialise CodeT5 model and build a pipeline for our summarization task.
The AST is traversed and functions are extracted using a parser.
The CodeT5 summarization pipeline is used to generate a summary for each function text.

Future Challenges/Solutions :

Future challenges in the field of autonomous code summarization are as follows:

1. Handling Diverse Coding Styles: Different programmers have different coding styles, which can make the code difficult to read and summarise. Future models should be able to handle code written in various styles.
2. Increasing code complexity: Code can become increasingly complex as software development practices evolve. Future models will need to be able to understand and summarise complex code structures including recursion, and advanced data structures.

Possible solutions to these challenges could include:

1. Larger Datasets: Training models on larger datasets could help them better understand and handle diverse coding styles.
2. Advancements in Deep Learning and evolution of LLMs: With evolving techniques like MoE (Mixture of experts) and further advancements in deep learning and computing power, the ability to understand and summarise code could increase.
3. Feedback Mechanism (RHLF): Reinforcement Learning with Human Feedback (RLHF) is a promising approach to tackle the challenges in autonomous code summarization. It combines the power of reinforcement learning with human feedback to improve the quality of generated summaries.

Credits:
CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation.
A Transformer-based Approach for Source Code Summarization