perplexity

MMAPR Paper Summary: Teaching Guide with Definitions and Examples

Based on the research paper content provided, here's a comprehensive explanation of the key concepts, hyperparameters, and state-of-the-art systems mentioned:

Core Definitions from the Paper

Large Language Models (LLMs)

Definition from paper: "A large language model (LLM) can be viewed as a probability distribution over sequences of words. This distribution is learned using a deep neural network with a large number of parameters."

Example: Think of an LLM like a very sophisticated autocomplete system that predicts what word comes next in a sentence based on patterns it learned from millions of text documents.

Large Language Model trained on Code (LLMC)

Definition: "When the LLM has been trained on significant amounts of code, we refer to it as a large language model trained on code (LLMC)."

Example: OpenAI's Codex is an LLMC - it's like GPT-3 but specifically trained on over 50 million GitHub repositories to understand programming languages.

Hyperparameters Explained

Temperature

Definition from paper: "One important hyperparameter is temperature, which controls the extent to which we sample less likely completions."

What it does:

- **Low temperature (e.g., 0.1):** The model becomes very conservative and predictable, always choosing the most likely next word

- **High temperature (e.g., 1.0+):** The model becomes more creative and random, sometimes choosing less likely but potentially more interesting options

Example from paper: "We set the temperature to 0.8 based on preliminary experiments."

Practical Example:

```
# With temperature = 0.1 (conservative)
def add_numbers(a, b):
    return a + b  # Very predictable, standard solution

# With temperature = 0.8 (more creative)
def add_numbers(a, b):
    result = a + b
    return result  # Slightly more verbose but still correct
```

Learning Paradigms

Few-Shot Learning

Definition from paper: "LLMs have shown to be effective for few- and even zero-shot learning. This means that the LLM can perform tasks it was not explicitly trained for just by giving it a few examples of the task."

Example from paper:

```
# Few-shot prompt example:
[[ Shot Starts ]]
# Incorrect Program #
print (m+n)
# Correct Program #
print (m*n)
[Shot Ends]

[[ Buggy Program Starts ]]
### Buggy Program ###
sum = m
i = 0
while i < n:
    sum += 1
```

```
    i += 1
print (sum)
```

Zero-Shot Learning

Definition: The model performs a task without any examples, just based on instructions.

Example: Asking the model to "fix this Python code" without showing any examples of fixes.

Prompt-Based Learning

Definition from paper: "A prompt is a textual template that can be given as input to the LLM to obtain a sequence of iteratively predicted next tokens, called a generation."

Example from paper:

```
# Fix the syntax error of the program #
# Buggy program #
print("\")
```

State-of-the-Art Systems Mentioned

BIFI

Definition from paper: "BIFI is a state-of-the-art transformer-based syntax repair engine for Python"

What it does: Fixes syntax errors in Python code using transformer neural networks
Performance: 80.07% syntax repair rate on the benchmark

Refactory

Definition from paper: "Refactory is a state-of-the-art symbolic semantic repair engine for introductory Python assignments"

What it does: Uses symbolic reasoning to fix logical/semantic errors in student code
Limitation: Can only work on syntactically correct code

GenProg

Definition: A genetic programming-based automated program repair system

Approach: Uses evolutionary algorithms to generate and test program fixes

Prophet

Definition: A statistical learning approach for program repair

Approach: Learns patterns from existing code to suggest fixes

CodeT5

Definition: A pre-trained transformer model for code understanding and generation

Training: Pre-trained on code-text pairs for better code comprehension

MMAPR Architecture Components

Multi-Modal Prompts

Definition from paper: "Multi-modal prompts are those that incorporate different modalities of inputs, such as natural language, code, and data."

Example from paper:

```
[[ Buggy Program ]]
### Buggy Program ###
x=input()
y=int(x)
# ... buggy code

[[ Problem Description ]]
#Write a program to read a number (int) from the user...

[[ Test Suite ]]
#input: 43
#output: Reverse: 34
```

Program Chunking

Definition: A technique to extract relevant code sections containing syntax errors

Algorithm from paper:

1. Locate the error line
2. Find the indentation level
3. Extract adjacent code with same/higher indentation
4. Include encompassing control-flow statements

Purpose: Reduces spurious edits by limiting the code surface exposed to the LLMC

Evaluation Metrics

Token Edit Distance (TED)

Definition: Measures how many token-level changes are needed to transform one program into another

Why it matters: Smaller edit distances mean the repair is closer to the student's original thinking, making it easier to understand

Results from paper:

- MMAPR: 31.40 average TED
- Baseline (BIFI + Refactory): 42.50 average TED

Repair Rate

Definition: Percentage of programs that the system can successfully fix

Results from paper:

- MMAPR (without few-shot): 86.71%
- MMAPR (with few-shot): 96.50%
- Baseline: 67.13%

Key Performance Insights

Syntax vs Semantic Repair

Syntax Phase: Fixes structural errors (missing colons, indentation, etc.)

- MMAPR: 100% success rate
- BIFI: 80.07% success rate

Semantic Phase: Fixes logical errors (wrong algorithms, incorrect outputs)

- Uses test cases as validation
- Employs few-shot learning with peer programs

Iterative Querying Impact

Definition: Repeating the repair process multiple times to fix multiple errors

Result: Improved repair rate from 82.87% to 86.71% by allowing iteration

This comprehensive breakdown shows how MMAPR combines multiple state-of-the-art techniques with carefully tuned hyperparameters to achieve superior performance in automated program repair for educational settings.

⁂