# Nghiên cứu về Fast & Slow Thinking Systems trong LLMs

## 1. Tổng quan từ tài liệu gốc

Tài liệu gốc đề xuất một phương pháp để kết hợp hai hệ thống tư duy trong LLMs: - **Fast thinking (System 1)**: Tư duy nhanh, trực giác và tự động, dựa vào các mẫu đã học - **Slow thinking (System 2)**: Tư duy chậm, có phương pháp và phân tích, dựa vào suy luận logic từng bước

Kiến trúc đề xuất bao gồm: - Mô-đun nhận diện độ phức tạp (Complexity Detector) - Mô-đun điều khiển tư duy (Thinking Controller) - Mô-đun Fast Thinking - Mô-đun Slow Thinking - Mô-đun tích hợp kết quả (Result Integrator)

Thách thức chính: - Cân bằng tài nguyên tính toán - Thiết kế cơ chế chuyển đổi giữa hai chế độ tư duy - Tích hợp kết quả từ cả hai quá trình - Thiếu dữ liệu huấn luyện chất lượng cao - Khó khăn trong việc gán nhãn và đánh giá

## 2. Nghiên cứu từ bài báo "Fast-Slow-Thinking: Complex Task Solving with Large Language Models" (2025)

Bài báo này đề xuất phương pháp "Fast-Slow-Thinking" (FST) để giải quyết các nhiệm vụ phức tạp với LLMs. Phương pháp này gồm ba bước:

1. **Fast Thinking (FT)**:
2. Loại bỏ các ràng buộc phức tạp từ nhiệm vụ
3. Đơn giản hóa nhiệm vụ thành một nhiệm vụ tổng quát và ngắn gọn

4. Tập trung vào khía cạnh tổng quát của nhiệm vụ

5. **Slow Thinking (ST)**:

6. Xem xét lại các ràng buộc của nhiệm vụ gốc
7. Cải thiện giải pháp được tạo ra trong FT để đáp ứng các ràng buộc

8. Tập trung vào chi tiết của nhiệm vụ

9. **Output Inspection (OI)**:

10. Kiểm tra và sửa chữa các lỗi tiềm ẩn trong câu trả lời
11. Đảm bảo câu trả lời đáp ứng tất cả các yêu cầu

Phương pháp này đã được thử nghiệm trên nhiều loại nhiệm vụ khác nhau như suy luận toán học, trả lời nội dung dài, và tạo câu chuyện có ràng buộc, với các mô hình nền tảng như GPT-3.5-turbo, Llama-3.1-8B-Instruct, và Gemini-pro.

## 3. Nghiên cứu từ bài báo "Unlocking a New Rust Programming Experience: Fast and Slow Thinking with LLMs" (2025)

Bài báo này giới thiệu RustBrain, một framework tích hợp hai quá trình tư duy để giải quyết các Undefined Behaviors (UBs) trong Rust:

1. **Fast Thinking**:
2. Trích xuất thông tin mã nguồn
3. Xác định loại lỗi
4. Phân tích sử dụng cơ sở kiến thức rộng

5. Tạo ra nhiều giải pháp sửa chữa

6. **Slow Thinking**:

7. Phân tích sâu và xác thực các giải pháp sửa chữa
8. Dựa trên bốn nguyên tắc thiết kế: phân rã, xác thực, suy luận và tổng quát hóa

9. Sử dụng nhiều agent khác nhau để hỗ trợ phân rã và xác thực

10. **Cơ chế phản hồi**:

11. Kết nối hai quá trình tư duy
12. Tối ưu hóa liên tục việc tạo giải pháp
13. Tinh chỉnh quá trình ra quyết định

# 4. Nghiên cứu từ bài báo "Slow Thinking with LLMs: Lessons from Imitation, Exploration, and Self-Improvement" (2024)

Bài báo này đề xuất một framework ba giai đoạn để nâng cao khả năng suy luận trong LLMs:

1. **Imitation (Bắt chước)**:
2. Huấn luyện mô hình để tuân theo các định dạng cụ thể
3. Sử dụng dữ liệu tối thiểu để tạo ra suy luận và giải pháp
4. **Exploration (Khám phá)**:
5. Tập trung vào các vấn đề khó
6. Phát triển nhiều giải pháp
7. Cải thiện dựa trên câu trả lời đúng
8. Đặc biệt hiệu quả cho các nhiệm vụ đòi hỏi slow thinking
9. **Self-improvement (Tự cải thiện)**:
10. Sử dụng dữ liệu chất lượng cao
11. Áp dụng các kỹ thuật như supervised fine-tuning (SFT) và direct preference optimization (DPO)
12. Lọc dữ liệu chất lượng thấp dựa trên các chỉ số như độ dài và perplexity

Phương pháp này đã được đánh giá trên các benchmark thách thức như MATH-OAI, AIME2024, và GPQA, với Qwen2.5-32B-Instruct làm mô hình nền tảng.

# 5. Điểm chung và khác biệt giữa các phương pháp

### Điểm chung:

- Tất cả đều phân biệt rõ ràng giữa fast thinking và slow thinking
- Đều nhấn mạnh việc kết hợp hai hệ thống tư duy để giải quyết các vấn đề phức tạp
- Đều sử dụng cách tiếp cận từ tổng quát đến chi tiết (coarse-to-fine)
- Đều có cơ chế kiểm tra và cải thiện kết quả

### Điểm khác biệt:

- FST tập trung vào việc phân tách nhiệm vụ phức tạp

- RustBrain tập trung vào việc sửa lỗi trong mã nguồn Rust
- Phương pháp "Imitate, Explore, Self-improve" tập trung vào quá trình huấn luyện mô hình

# 6. Ý tưởng cải tiến cho khung khái niệm

1. **Kiến trúc mô hình tích hợp**:
2. Kết hợp mô-đun nhận diện độ phức tạp với cơ chế phân tách nhiệm vụ của FST
3. Tích hợp cơ chế phản hồi từ RustBrain để liên tục cải thiện quá trình ra quyết định

4. Áp dụng framework ba giai đoạn (bắt chước, khám phá, tự cải thiện) cho quá trình huấn luyện

5. **Cải tiến quy trình tạo dữ liệu**:

6. Tạo dữ liệu có cấu trúc rõ ràng cho cả fast thinking và slow thinking
7. Phân loại nhiệm vụ dựa trên độ phức tạp và loại tư duy phù hợp
8. Tạo dữ liệu cho cả ba loại tư duy: fast, slow, và hybrid

9. Áp dụng các kỹ thuật tăng cường dữ liệu như biến đổi dữ liệu, tạo dữ liệu đối nghịch, và tạo dữ liệu tự động

10. **Cải tiến quy trình huấn luyện**:

11. Áp dụng quy trình huấn luyện ba giai đoạn: pre-training, instruction tuning với dữ liệu phân loại, và fine-tuning với dữ liệu kết hợp fast-slow thinking
12. Sử dụng các kỹ thuật tối ưu hóa cho RTX 4090 như gradient checkpointing, mixed precision training, và efficient attention mechanisms

13. Áp dụng các hyperparameters phù hợp với cấu hình phần cứng

14. **Cải tiến framework triển khai**:

15. Xây dựng framework toàn diện bao gồm các thành phần tạo và quản lý dữ liệu, huấn luyện mô hình, đánh giá và phân tích, và triển khai và inference
16. Tối ưu hóa framework cho RTX 4090 với các kỹ thuật tối ưu hóa bộ nhớ, tốc độ, và hiệu suất