Replicating Research into Executable Knowledge

Da Ma
Institute for Engineering of Products and Systems (IEPS), Otto von Guericke University Magdeburg, da.ma@ovgu.de

Sebastian Lang
Institute for Engineering of Products and Systems (IEPS), Otto von Guericke University Magdeburg, Germany

Abstract
This study presents an agentic pipeline that converts engineering research papers into callable Model Context Protocol (MCP) tools. Existing approaches such as Retrieval-Augmented Generation (RAG) can retrieve knowledge from scientific literature and generate responses for humans, but they do not address the translation of such knowledge into tools usable by a Large Language Model (LLM), leaving a gap between static literature and executable knowledge that LLM agents require. The proposed pipeline integrates PDF-to-Markdown conversion via MinerU, tool coordination through LangGraph, and MCP code synthesis using the OpenAI Codex CLI. The most suitable executable knowledge artifact is identified from the full document, which may be an algorithm, or a mathematical model. Implementation-relevant information is then extracted through the detailed-reading steps. The extracted information is consolidated into a task specification that drives the OpenAI Codex CLI to generate FastMCP-compatible MCP tools. The prototype is validated on papers containing specific algorithms and mathematical models. The proof-of-concept confirms the feasibility of converting unstructured scientific literature into usable MCP tools, and highlights the wider potential for extending callable tool libraries that enrich the LLM ecosystem.

1. Introduction
Engineering research publications typically embed algorithmic and mathematical knowledge in forms accessible to expert readers but not directly usable for implementation. Procedures are typically distributed across natural language descriptions, pseudo-code, mathematical notation, and figures without a standardized interface that software agents can use to invoke them. Translating such content into runnable code manually is labor-intensive and error-prone, particularly where complex equations and non-standard notation are involved.

Recent progress in agentic systems based on a Large Language Model (LLM) has created concrete demand for callable tools that encapsulate domain knowledge. Model Context Protocol (MCP) has evolved into a structured interface standard built around a client-server architecture in which host applications connect to multiple lightweight servers, enabling LLM agents to discover and invoke external tools in a uniform way (Ferrag et al., 2025). The construction of such tools, however, still depends on manual implementation effort, which is a bottleneck when the source knowledge is dispersed across large corpora of scientific papers. Retrieval-Augmented Generation (RAG) systems can retrieve relevant knowledge for constructing MCP tools, but they overlook latent content in a broader context (Luo et al., 2025). Moreover, translating the retrieved content into executable code requires additional processing, and repeatedly retrieving and regenerating implementation code on each invocation is neither economical nor reliable. This leaves a practical gap: algorithmic knowledge is abundant in the literature, yet it remains inaccessible to autonomous agents as callable software.

The proposed agentic pipeline targets this gap by treating each source paper as a repository of knowledge rather than a document to be retrieved. After PDF-to-Markdown conversion via MinerU (Niu et al., 2025), a LangGraph ReAct agent performs a rough-reading pass over the entire Markdown document to identify the most suitable executable knowledge artifact, which may be an algorithm, a mathematical model, or any other well-specified computational procedure. A subsequent detailed-reading step and a multimodal figure-analysis step then extract the information needed for implementation. This information is assembled into a task specification that drives targeted code

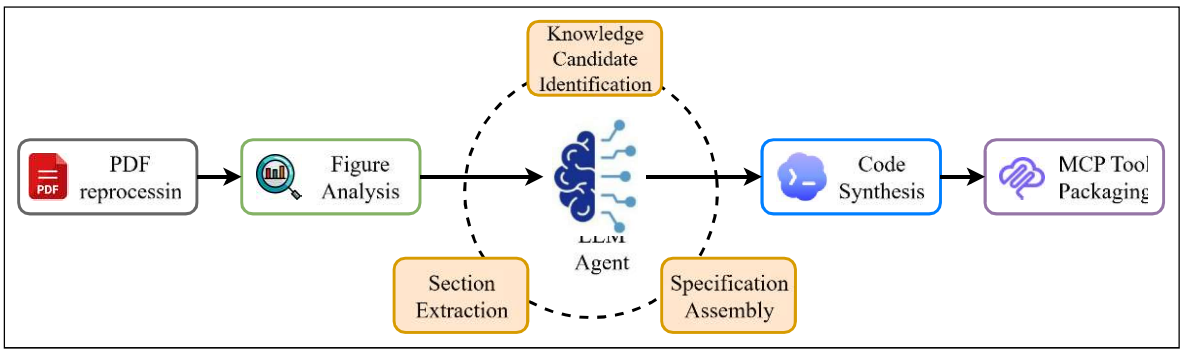

Figure 1: Overview of the Proposed Agentic Pipeline

synthesis by the OpenAI Codex CLI, producing a tested MCP tool built on the open-source library FastMCP (Alla, 2025).

A design constraint of the proposed pipeline is that it does not attempt full reproduction of each paper, but rather focuses on replicating the most critical knowledge. The pipeline is aimed at capturing the most algorithmically explicit content in each source paper, a scope that reduces hallucination risk from LLM-based code synthesis and ensures that generated code remains practically usable. The prototype is evaluated on two paper categories: papers containing specific algorithms and papers centered on mathematical models.

The broader motivation for this work extends beyond a single prototype. Systematic extraction of LLM-readable, callable knowledge from scientific literature is a step toward building an open-source tool ecosystem that Large Language Model Operation (LLMOps) pipelines can consume in the same way that human engineers rely on open-source libraries. As LLM agents take on increasingly complex technical tasks, the availability of structured, paper-grounded tools will become a practical prerequisite for reliable autonomous engineering workflows. The remainder of this paper is organized as follows. Section 2 describes the methodology and pipeline architecture. Section 3 presents implementation results. Section 4 discusses limitations and future directions.

2. Methodology

2.1. Pipeline Overview

The system is structured as a staged pipeline that separates deterministic preprocessing from LLM-mediated reasoning in handling complex tasks (Wang et al., 2024). As illustrated in Figure 1, the deterministic flow begins with PDF preprocessing and figure analysis, generating a Markdown file containing the results of the image analysis. Once the enriched Markdown file is available, the LangGraph ReAct agent takes over: it performs a rough-reading pass to identify the most suitable executable knowledge artifact, followed by detailed section extraction to gather implementation-relevant content. Strategic planning requires tools from multiple specialized roles (Belle et al., 2025). The ReAct mechanism is well suited to this phase because the agent can flexibly decide which sections to read and in what depth, adapting to the structure of each input paper rather than following a fixed sequence. The extracted knowledge is then consolidated into a per-paper task specification file. Code synthesis is delegated entirely to the OpenAI Codex CLI, whose strong task-following and code-generation capabilities make it suitable for the final implementation step.

Although proprietary models such as OpenAI Codex or Claude Code can also read a PDF and generate code directly, applying them in this way to academic papers carries two practical limitations. First, full papers are lengthy, and submitting an entire PDF to a frontier model on every invocation is token-intensive and costly. Decoupling document understanding from code synthesis allows a lighter or locally hosted model to handle the reading phase, so that only a concise, targeted specification is forwarded to the Codex CLI, which is more economical and also allows the specification to be manually inspected and corrected before synthesis. Second, the PDF parsing libraries used by the frontier models have limitations in handling complex mathematical expressions, which is a critical deficiency for knowledge replication that depends on correctly interpreting specialized notation and equations. Designing a scalable and cost-effective architecture is critical to the overall cost (Sinha et al., 2024). These considerations motivated the pipeline architecture described below. The following subsections will introduce each stage in the pipeline.

2.2. Tool Descriptions

2.2.1. PDF Preprocessing

Two approaches are commonly used to extract content from PDF documents. The first relies on open-source PDF parsing libraries that operate on the underlying file structure; these tools perform adequately on plain text but often produce garbled output when encountering equations, special symbols, or tables, and they cannot process scanned documents in which pages are stored as

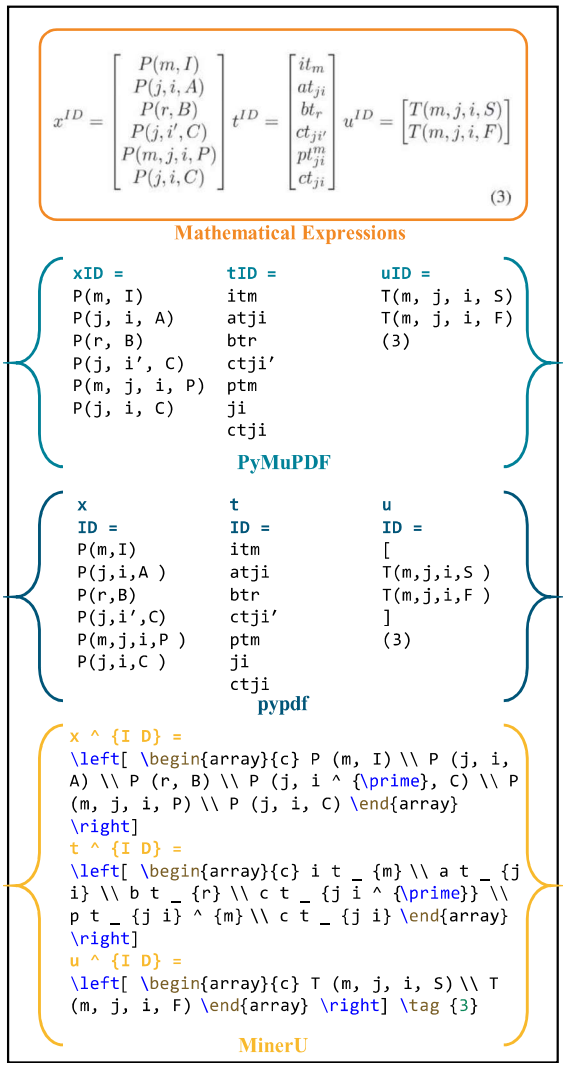

Figure 2: Results from PDF Processing Libraries

raster images with no embedded text layer. The second approach, Optical Character Recognition (OCR), can recover text from scanned documents but is generally unable to infer the heading-and-subheading hierarchy of the original document. MinerU addresses both limitations in PDF processing. It is based on a model fine-tuned from a visual LLM, enabling it not only to read text but also to segment and classify regions within each page, extract paragraphs in their original hierarchy as Markdown, isolate figure images as separate files, and most critically for this application, convert mathematical expressions into LaTeX notation. Figure 2 presents a comparison of processing results between popular libraries and MinerU, demonstrating that MinerU has a significant advantage in recognizing special structures and upper/lower indices. MinerU is invoked through a subprocess wrapper, and its output, a Markdown file containing heading hierarchy, mathematical expressions, and figure references, is stored in a workspace directory.

2.2.2. Multimodal Figure Analysis

Engineering papers can convey algorithmic structure, mathematical relationships, and parametric information through figures that are not fully described in the captions. To capture this information, a figure analysis step submits each extracted image to the vision interface of the locally hosted Ollama multimodal model. The model responses are treated as supplementary descriptions and written back into the enriched Markdown documents, co-locating visual and textual content before the knowledge-candidate identification during the rough reading step.

2.2.3. Knowledge Candidate Identification

Once the enriched Markdown document with figure annotations is available, the LangGraph ReAct agent conducts a rough-reading step over the full document. The purpose of this step is to assess the paper comprehensively and determine which executable knowledge artifact is the most suitable target for MCP conversion; the artifact may be an algorithm, a mathematical model, a scoring rule, or any other well-specified computational procedure. The LLM is required to generate a structured response that identifies the artifact type, provides an implementation-focused rationale, summarizes the paper, and lists the section headings relevant to implementation. This identification step is important because engineering papers vary considerably in how they organize executable knowledge, and constraining the agent to a fixed artifact type would reduce generalizability. Furthermore, some sections commonly found in engineering papers, such as literature reviews, experimental analyses, and reference lists, are not relevant to implementation; excluding these sections during the rough-reading pass ensures that subsequently extracted content remains focused.

2.2.4. Specification Assembly

Based on the relevant section headings identified during the rough-reading stage, a rule-based parser through regular expression retrieves the corresponding source passages directly from the enriched Markdown document. These extracted passages are then assembled together with the rough-reading output into a task specification, rendered using a templating mechanism as shown in the Listing 1. The specification is organized into a set of mandatory components: a concrete implementation instruction for the code synthesis stage, the original source passages extracted from the relevant sections, input-output specifications, domain constraints and assumptions, figure analysis results, FastMCP-compatible coding conventions and usage examples, and at least two concrete test cases.

2.2.5. Code Synthesis and MCP Tool Packaging

The final stage delegates implementation to the OpenAI Codex CLI, which receives the per-paper task specification and is instructed to produce three artifacts: a FastMCP-compatible Python implementation that exposes the target knowledge as a callable MCP tool; a structured tool manifest

```
# Task
{{ task_instruction }}

## Input / Output Specification
{{ io_spec }}

## Implementation Requirements
{{ implementation_requirements }}

## FastMCP Reference (fastmcp==3.x)
You **must** follow the pattern below exactly. Any deviation will produce a non-functional MCP server.
```python
from fastmcp import FastMCP

# 1. Create the server instance — use a kebab-case name
mcp = FastMCP(
    "{{ paper_slug }}",
    instructions="One-sentence description of what this server does.",
)

# 2. Register each tool using individual typed parameters.
#    The @mcp.tool() decorator keeps the function directly callable.
@mcp.tool(description="Clear description shown to the MCP client.")
def my_function(
    param1: int,
    param2: str,
    optional_param: float = 1.0,
) -> dict[str, object]:
    Args:
        param1: Description.
        param2: Description.
        optional_param: Description, defaults to 1.0.

    Returns:
        A JSON-serialisable dict with result fields.
    """
    # implementation
    return {"result": ..., "success": True}
```

Test Cases
{% for case in test_cases %}### Test Case {{ loop.index }}
Input: {{ case.input }}
Expected Output: {{ case.output }}
```

Listing 1: Template of Specification

documenting the tool's name, description, input-output schema, source paper reference, and a usage example; and a test suite containing at least two test cases. Codex is invoked as a subprocess within a dedicated Python environment. After Codex returns, the pipeline performs a series of validation checks on the generated artifacts: syntactic correctness of the implementation is verified programmatically, the manifest is checked for structural completeness and all required fields, and the presence of the test file is confirmed. If any check fails, Codex is retried once. The generated tool is consumable by any MCP-compatible LLM agent client, completing the pipeline from raw PDF input to callable software artifact.

3. Results and Discussion

3.1. Evaluation Strategy

Given that this study is a proof-of-concept system rather than a production-ready one, the evaluation is qualitative and focuses on demonstrating end-to-end feasibility. The primary criterion for a successful run is that the pipeline produces a syntactically valid, FastMCP-compatible MCP tool whose implementation can be traced back to specific content in the source paper. Quantitative metrics such as execution accuracy or semantic similarity to a ground-truth implementation are not reported at this stage; their systematic collection is left to future work. Instead, two representative paper categories are used to explore the range of knowledge artifacts the pipeline can handle: papers presenting specific algorithms in pseudocode form, and papers whose central contribution is a mathematical model. These two categories were chosen because they represent distinct extraction challenges and cover a large proportion of implementable knowledge artifacts found in engineering literature.

3.2. Case Study

Two papers are selected as case studies, representing an algorithm-centric and a model-centric contribution respectively. The pipeline was configured with *qwen3-vl:8b-instruct-q4_K_M* for local reasoning and *GPT-5.1 Codex mini* for code synthesis.

3.2.1. Case Study I: Algorithm Extraction

The first case study uses the paper by Sun et al. (Sun et al., 2021), which proposes an improved Dijkstra algorithm for AGV path planning in a grid-based warehouse environment. This paper is representative of a common structural pattern in engineering publications: it presents both a baseline method (the traditional four-directional Dijkstra algorithm) and a proposed improvement (the eight-angle search variant), where the distinction between the two is conveyed through figures rather than text alone. This structure tests two aspects of the pipeline simultaneously: whether the agent can correctly identify the proposed algorithm as the target artifact among multiple candidate methods, and whether the multimodal figure-analysis step can extract algorithm-relevant information from figures.

During the rough-reading pass, the LangGraph ReAct agent correctly selected Section 4 of the

paper as the primary MCP conversion target, identifying the improved eight-angle algorithm rather than the baseline Dijkstra description in Section 3. The multimodal figure-analysis step processed the figures from two algorithms and correctly understood the eight-directional connectivity structure. This figure-derived information supplemented the textual extraction and was incorporated into the task specification, demonstrating that the vision module contributes content when the paper relies on diagrams to communicate algorithmic structure.

The Codex CLI synthesized the implementation on the first invocation. The generated *improved_dijkstra* function accepts a binary occupancy grid, start and goal coordinates, and configurable movement costs, including the diagonal cost. Post-generation checks confirmed syntactic correctness and manifest validity. The two generated test cases cover diagonal path preference on an obstacle-free grid, verified against expected cost, and correct unreachable-goal handling when the target cell is enclosed by obstacles. One synthesis decision that goes slightly beyond the paper's explicit text is a corner-cutting prevention check for diagonal moves, which verifies that both orthogonally adjacent cells are free before a diagonal step is accepted. This is physically appropriate for AGV operation but is not stated in the source, indicating that Codex introduced a reasonable domain assumption during synthesis.

```
@mcp.tool(description="Compute the improved
Dijkstra eight-angle path for an AGV grid.")
def improved_dijkstra(
    grid: list[list[int]],
    start: list[int],
    goal: list[int],
    orthogonal_cost: float = 1.0,
    diagonal_cost: float = 1.4142135623730951,
    max_iterations: int = 10000,
) -> dict[str, object]:
    directions: list[tuple[int, int, float]] = [
        (1, 0, orthogonal_cost),
        (-1, 0, orthogonal_cost),
        (0, 1, orthogonal_cost),
        (0, -1, orthogonal_cost),
        (1, 1, diagonal_cost),
        (1, -1, diagonal_cost),
        (-1, 1, diagonal_cost),
        (-1, -1, diagonal_cost),
    ]
    ...
```

Listing 2: Code Snippet for the MCP Tool

The generated code as shown in Listing 2 above describes the tool as an improved Dijkstra AGV planner using eight-angle neighborhood expansion. The traditional Dijkstra algorithm usually uses four-angle search for node expansion in a grid map. The improved method searches the eight directions of the grid: west, southwest, north, northwest, east, northeast, south, and southeast.

The main limitation of this case is that the generated tool cannot reproduce the specific quantitative results reported in the paper, such as the path length of 27.331 m on the paper's test grid, because the section containing the particular obstacle layout was not recognized as directly relevant to the algorithm during the rough-reading stage. This is consistent with the pipeline's design scope: the agent extracted the algorithm logic faithfully, but the specific simulation environment parameters were not part of the extracted artifact. Excluding sections that appear peripheral during the rough-reading improves the focus and conciseness of the specification, but it carries the risk of discarding contextual information that, while not part of the core algorithm, is necessary to reproduce reported results. One direction for future work is to investigate context compacting techniques that condense rather than discard lower-priority sections, allowing the specification to remain concise while retaining a wider coverage of the source paper.

3.2.2. Case Study II: Mathematical Model Extraction

The second case study uses the paper by Mostafa and Eltawil (Mostafa & Eltawil, 2015), which proposes a generic mathematical formulation for the Production-Inventory-Distribution-Routing Problem (PIDRP). From the agent's perspective, this paper presents a substantially different extraction challenge from Case Study I. The majority of its content, comprising Sections I through III and Section V, consists of an extensive literature review, research gap analysis, and a conceptual framework, none of which contain directly executable content. The executable knowledge is limited to Section IV, where a fully defined model is introduced with a complete index set, parameter table, decision variable declaration, an objective function, and 21 constraint equations. The agent therefore needs to identify a relatively small target section buried within a large volume of source material.

During the rough-reading pass, the agent correctly classified the paper's primary executable artifact as a mathematical model rather than a procedural algorithm, and selected Section IV as the sole MCP conversion target. This classification decision is non-trivial because the paper contains substantial technical content in its review tables and discussion sections that could superficially appear relevant. The detailed-reading pass then extracted the full index sets (T, N, K, V), all parameters including setup costs s_k, unit operational costs o_k, production capacities c_k, inventory holding costs h_{0k} and h_{ik}, demand d_{ikt}, and vehicle capacities q_v, and all

eight decision variable types (Q_{kt}, P_{kt}, I_{0kt}, I_{ikt}, X_{iktv}, W_{it}, Y_{ijktv}, Z_{ijtv}). This demonstrates that the pipeline can narrow its focus to the executable section of a review-dominated paper without incorporating irrelevant content into the specification.

A central challenge of this case study is the density and complexity of the mathematical notation in Section IV. The model involves multi-indexed variables such as Y_{ijktv} and Z_{ijtv}, where each subscript carries a unique semantic role spanning facilities, products, time periods, and vehicles simultaneously. MinerU's LaTeX extraction capability was essential here: unlike standard PDF parsers that frequently garble subscript-heavy expressions, MinerU preserved the index structure of the constraint equations in machine-readable form, allowing the LLM reading stage to correctly interpret the scope of each constraint and map variable names to their definitions in the notation table. The generated function accepts 20 typed input parameters that directly mirror the model's index sets and parameter notation. The implementation covers all major constraints from the paper, from production capacity (constraints 2 and 3) and inventory flow balance (constraints 4 and 5) to flow conservation (constraints 11 and 13). This end-to-end handling of dense, multi-indexed mathematical notation, from PDF extraction through LLM interpretation to typed function parameters in the generated tool, is a key indicator that the pipeline is capable of handling complex mathematical expressions.

3.3. Discussion

The two case studies confirm that the pipeline can generate functional MCP tools from both algorithm-centric and model-centric papers. However, there are several limitations. First, the quality of extraction is bounded by the quality of the source document: papers with ambiguous notation, inconsistent variable naming, or results distributed across different sections will produce incomplete or incorrect specifications, and the pipeline has no mechanism to flag this automatically. Second, the current implementation evaluates only two paper instances; a more rigorous assessment would require a larger and more diverse paper set with independently verified reference implementations for comparison.

The pipeline's design also imposes a deliberate scope constraint: it targets the single most explicit knowledge unit per paper rather than attempting a complete reproduction of all reported experiments. This constraint is appropriate for a prototype and reduces hallucination risk, but it means that papers whose contribution is a combined system of multiple interacting components may yield only partial implementations. Addressing this would require either multi-target extraction across pipeline runs or a more sophisticated candidate-selection strategy that can represent composite knowledge artifacts. Finally, the dependence on the proprietary Codex CLI limits its portability. As the performance of open-source LLMs improves, future versions should explore using open-source LLMs and coding agents for code synthesis to lower the barrier to implementation.

4. Limitations and Conclusion

The prototype establishes the feasibility of converting engineering research papers into callable MCP tools through staged information discovery and specification-guided code synthesis. The two case studies confirm that the pipeline can correctly identify executable knowledge artifacts across different paper types, handle multimodal content including algorithm diagrams and dense mathematical notation, and produce correct implementations on the first synthesis attempt. The evidential chain from source paper to generated tool is preserved through systematic logging and intermediate artifact persistence, making the pipeline's decisions inspectable.

The current implementation has three notable limitations. The pipeline's output quality is bounded by the completeness of the source paper: publications that omit critical variables or constraints cannot be faithfully implemented regardless of extraction quality. The section-filtering strategy improves specification conciseness but may discard peripheral context that is still relevant to result reproduction. The evaluation also remains qualitative; broader empirical validation across diverse paper types and subject domains is needed to characterize the pipeline's generalizability.

At a broader level, the work contributes to an emerging agenda of building callable knowledge ecosystems for LLM agents. An increasing amount of engineering literature contains explicitly formulated algorithms and computational models that remain inaccessible to autonomous agents in their current form. Systematically converting this literature into standardized MCP tools would allow agents to invoke domain knowledge directly, rather than re-deriving it from raw text at each invocation. The present prototype demonstrates that this conversion is tractable with current tools and models, and provides a foundation for future work on scalable, auditable translation of scientific knowledge into reusable software artifacts.

References

Alla, M. (2025). Scalable MCP Server-Client Architecture with FastMCP in Microservices. Journal of Multidisciplinary, 5(7), 823–829. https://doi.org/10.5281/zenodo.16354765

Belle, N., Barnes, D., Amayuelas, A., Bercovich, I., Wang, X. E., & Wang, W. (2025). Agents of

Change: Self-Evolving LLM Agents for Strategic Planning (arXiv:2506.04651). arXiv. https://doi.org/10.48550/arXiv.2506.04651

Ferrag, M. A., Tihanyi, N., & Debbah, M. (2025). From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review (arXiv:2504.19678). arXiv. https://doi.org/10.48550/arXiv.2504.19678

Luo, Y., Yu, Z., Wang, X., Zhu, Y., Zhang, N., Wei, L., Du, L., Zheng, D., & Chen, H. (2025). Executable Knowledge Graphs for Replicating AI Research (arXiv:2510.17795). arXiv. https://doi.org/10.48550/arXiv.2510.17795

Mostafa, N. A., & Eltawil, A. B. (2015). The production-inventory-distribution-routing problem: An integrated formulation and solution framework. 2015 International Conference on Industrial Engineering and Operations Management (IEOM), 1–10. https://doi.org/10.1109/IEOM.2015.7093751

Niu, J., Liu, Z., Gu, Z., Wang, B., Ouyang, L., Zhao, Z., Chu, T., He, T., Wu, F., Zhang, Q., Jin, Z., Liang, G., Zhang, R., Zhang, W., Qu, Y., Ren, Z., Sun, Y., Zheng, Y., Ma, D., … He, C. (2025). MinerU2.5: A Decoupled Vision-Language Model for Efficient High-Resolution Document Parsing (arXiv:2509.22186). arXiv. https://doi.org/10.48550/arXiv.2509.22186

Sinha, M., Menon, S., & Sagar, R. (2024). LLMOps: Definitions, Framework and Best Practices. 2024 International Conference on Electrical, Computer and Energy Technologies (ICECET), 1–6. https://doi.org/10.1109/ICECET61485.2024.10698359

Sun, Y., Fang, M., & Su, Y. (2021). AGV Path Planning based on Improved Dijkstra Algorithm. Journal of Physics: Conference Series, 1746(1), 012052. https://doi.org/10.1088/1742-6596/1746/1/012052

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J. (2024). A survey on large language model based autonomous agents. Frontiers of Computer Science, 18(6), 186345. https://doi.org/10.1007/s11704-024-40231-1