AIPA's International Journal on Artificial Intelligence: Bridging Technology, Society and Policy
ISSN: 3062-097X
Published: 07 December 2024

Large Language Model-Based Autonomous Agents: Trends and Directions

Levent Dinçkal[1*]

[1]**Artificial Intelligence Policies Association (AIPA) Member, ORCID: 0009-0007-7938-8368**

ORIGINAL RESEARCH PAPER

Abstract
This paper explores the vibrant field of autonomous agents based on large language models. In recent years transformer-based large language models (LLMs) have advanced state of the art considerably in a wide range of natural language tasks and demonstrated almost human-like reasoning capabilities and world knowledge. Since autonomous agents rely on such properties, advances in LLMs have accelerated the progress in autonomous agents. This paper reviews the literature by briefly describing how LLMs work and how they can be leveraged in the overall architecture of an autonomous agent to produce significantly more capable and robust agents. Planning, memory, and action components of the autonomous agent are examined separately and a discussion of trends and future directions follows.

autonomous agents, large language models, artificial intelligence, artificial general intelligence

Keywords: autonomous agents, large language models, artificial intelligence, artificial general intelligence

1 Introduction

Large language models (LLMs) based on the Transformer architecture [1] have been extraordinarily successful in recent years. Many of these models [2] [3] [4] [5], developed and released by research and industry groups in quick succession, have rapidly pushed forward the state of the art in natural language tasks in few-shot and zero-shot settings [6] [7]. LLMs excel in solving a broad range of natural tasks, exhibiting unprecedented emergent properties, extensive world knowledge, and advanced reasoning abilities [8]. Autonomous agents [9] stand to benefit significantly from these new capabilities, enhancing their ability to understand instructions in natural language as well as to plan and act with common sense and a robust world model.

This paper explores the rapidly evolving trends in LLM-based autonomous agents and describes selected important examples from the field. We provide a brief overview of LLMs and the transformer architecture in section 2 and in section 3, we examine the specifics of LLM-based autonomous agents, focusing on essential components such as planning, memory, and action. Finally, section 4 identifies and discusses prevalent patterns in current research, offering conclusions and future directions in this swiftly developing area.

OPEN ACCESS
AIPAJ Vol:1, Issue:1

Corresponding author
leventdinckal@gmail.com

Submitted *15 June 2024*

Accepted *28 October 2024*

Citation
Dinçkal, L. (2024). Large Language Model-Based Autonomous Agents: Trends and Directions. In AIPA's International Journal on Artificial Intelligence: Bridging Technology, Society and Policy (Vol. 1, Number 1, pp. 13–24). DOI: 10.5281/zenodo.14293261

2 An Overview of the Decoder-Only Transformer Architecture

We begin with a brief overview of the large language models as the LLM-based autonomous agents are, by their very nature, tightly coupled with this transformer-based architecture. The fundamental structure of transformers was put forth in the groundbreaking "Attention is All You Need" paper [1] and has remained mostly similar to this day, with the more recent models adding tweaks to this architecture for incremental benefits. Instead of documenting each possible modification to the architecture, we will present a stylized version of a possible current model and note that LLMs from different publishers might utilize slight variations in their structure.

At a fundamental level, the large language models create a conditional probability distribution on tokens, which are words or word-like bits of text described in more detail in subsection 2.1, given the tokens previously seen in the text. The models we consider in this paper are auto-regressive, which means they operate in a sequential manner to predict the next token based only on the previous tokens. LLMs are trained with massive corpora of text and the training aims to maximize the probability of the correct token at each location in the text. In concrete terms, given a sequence of tokens $\mathbf{t} = \{t_1, \cdots, t_N\}$ in the training corpus, the training objective is to maximize the likelihood of the correct token or, equivalently, minimize

$$L(\mathbf{t}) = -\sum_{i=1}^{N} \log[P(t_i|t_{i-1}, \cdots, t_{i-k}; \Omega)] \tag{1}$$

where $L(\mathbf{t})$ is the cross-entropy loss over the given sequence of tokens, k is the number of previous tokens we consider (sometimes called the context window length), and Ω is the set of trainable parameters of the model. This objective function will force the randomly initialized model to mimic its input texts after some training as the Kullback-Leibler divergence between the distributions of the training texts and predicted tokens will be minimized.

The architecture of the large language model we will consider can be seen in Figure 1. Note that this is a decoder-only model rather than the fundamental encoder-decoder structure proposed in [1] as the agents we are interested in will use decoder-only models. We now briefly review the components of this architecture separately in the order of their appearance in the forward pass of the model.

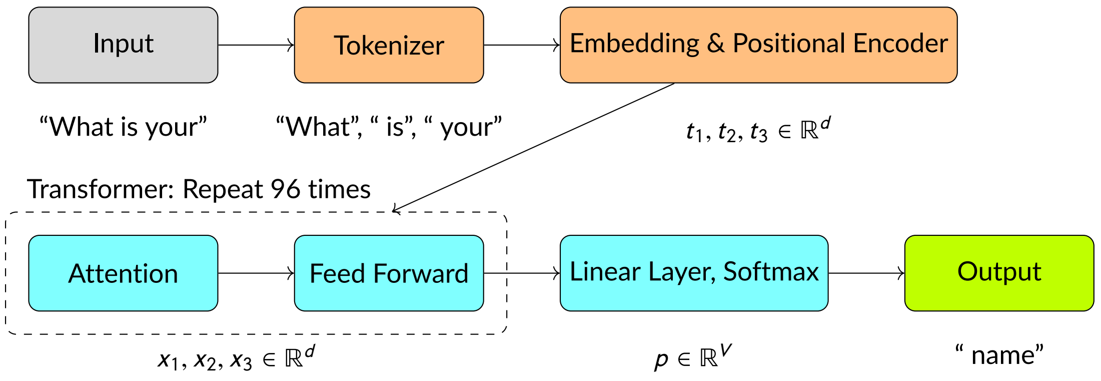

Figure 1. Stylized Structure of a Large Language Model.

2.1 Tokenization and Embedding

Sequences of text need to be tokenized and embedded into a vector space before they can be consumed by language models. A tokenizer is a deterministic program that uses a heuristic rule to break down text into chunks called tokens. These tokens, usually words and subwords, are the inputs into the model and the final output is a conditional probability distribution on the tokens. Two of the most commonly used tokenizers in the currently state-of-the-art models are Wordpiece [10] and Byte Pair Encoding [11]. They have a similar working principle in that they both break down the text into characters and then progressively merge contiguous chunks. The difference is that Byte Pair Encoding (BPE) merges chunks depending on the frequency of the merged text in the corpus, whereas Wordpiece, in addition to BPE's criterion, gives priority to merging chunks that are infrequent on their own. The examples below illustrate the tokenization of two random phrases by the BPE tokenizer of OpenAI's GPT-4 [5] as implemented by the tiktoken library [12].

Example Go to the store and buy groceries

Example high-rise buildings and backfilled excavation sites

The tokenizer indexes each possible token in the created vocabulary with an integer value and for any given text, this sequence of integers are sent into the embedding layer. Unlike the tokenizer, the embedding layer is not deterministic and usually needs to be trained along with the rest of the model. Embedding in NLP refers to the process of projecting each discrete token-indexing integer into a high-dimensional vector space. Some of the first papers on embedding in NLP were Word2Vec [13] and GloVe [13], which showed how to embed words into high-dimensional vector spaces in such a way that their distances and linear transformations in that space are semantically sensible. While these earlier algorithms considered vector spaces of dimension sizes in the hundreds, the current breed of models tend to embed tokens to much higher dimensional spaces in order to capture richer meaning, with LLaMA-2 [4], for instance, using an embedding dimension of 4096.

While the token vector embeddings capture the meanings of the individual words, the positions of tokens also hold semantic and syntactic value. The attention block itself has no way to account for the ordering of the tokens. Therefore, we need to feed information about the order to the rest of the model with another type of embedding called position encoding. [1] uses different frequencies of sine and cosine functions to derive positional encodings. A popular alternative is Rotary Position Embedding (RoPe) as proposed by [14] and used in prominent models such as LLaMA-2. Regardless of the method used, the positional encoding is usually a vector that is the same size as the semantic embedding vector described previously, which allows the two vectors to be simply summed up to create one vector per token as an input to the attention block.

2.2 The Transformer Block

The attention mechanism described in [1] is still commonly used in LLMs today and it is arguably the most important part of the model as it can take credit for most of the impressive features of large language models. The type of attention that we consider in this paper falls into the category of self-attention, which means it attends only to its input sequence as opposed to other attention types that can also attend to their output sequences.

Before launching into a description of the core of the transformer block, we note that most models include a normalization step at the end or, more commonly in the recent models, at the beginning of the attention mechanism. One popular normalization method is Layer Normalization [15], which scales and shifts its input to match a distribution easier to process. These normalization components provide numerical stability during training and help avoid the usual problems of vanishing or exploding gradients.

$$\begin{tikzcd}
 & & q_1 & & & \\
\text{“What”} & t_1 \in \mathbb{R}^d \arrow[ur, "Q"] \arrow[r, "K"] \arrow[dr, "V"'] & k_1 \arrow[r, "q_3"] & \alpha_{1,3} \arrow[r, "v_1"] & x_{1,3} \arrow[dddr] & \\
 & & v_1 & & & \\
 & & q_2 & & & \\
\text{“ is”} & t_2 \in \mathbb{R}^d \arrow[ur, "Q"] \arrow[r, "K"] \arrow[dr, "V"'] & k_2 \arrow[r, "q_3"] & \alpha_{2,3} \arrow[r, "v_2"] & x_{2,3} \arrow[r] & x_3 \in \mathbb{R}^d \\
 & & v_2 & & & \\
 & & q_3 & & & \\
\text{“ your”} & t_3 \in \mathbb{R}^d \arrow[ur, "Q"] \arrow[r, "K"] \arrow[dr, "V"'] & k_3 \arrow[r, "q_3"] & \alpha_{3,3} \arrow[r, "v_3"] & x_{3,3} \arrow[uuur] & \\
 & & v_3 & & &
\end{tikzcd}$$

Figure 2. Inner Working of the Attention Mechanism.

The fundamental function of the attention layer is to calculate the weight which a given token will assign to the information content of every other token. Figure 2 gives an overview of the mechanism. Given the token embedding dimension size d and assuming the attention mechanism also uses dimension d, the model learns the weight matrices $Q \in \mathbb{R}^{d\times d}$, $K \in \mathbb{R}^{d\times d}$, and $V \in \mathbb{R}^{d\times d}$, which are used to project the incoming token embedding vectors into query, key, value spaces respectively. Thus, at every prediction step of the model, we calculate for every token $t_i \in \mathbb{R}^d$ in the context window

$$\begin{aligned} q_i &= Qt_i \\ k_i &= Kt_i \\ v_i &= Vt_i \end{aligned} \tag{2}$$

which are query, key, and value vectors for the token respectively. Query vectors are used to query the relevance of the token to every other token. The key vector is the answer to queries from other tokens to determine similarity and the value vector holds the information content of the token. In practice, we keep the key and value vectors in a cache instead of re-calculating them at every step and further optimizing the memory footprint of that sizable cache has been a persistent research target recently as in [16]. At the end of the attention layer, we want to calculate a weighted sum of all value vectors for every token in the context window, where the weights are the attention weights given to the corresponding tokens. For a given token t_i, we first calculate attention scores a_{ij} between t_i and all other tokens t_j in the context window:

$$a_{ij} = \frac{q_i \cdot k_j}{\sqrt{d}} \tag{3}$$

The inner product is used here as a measure of similarity as two vectors will have a greater inner product if they are more closely aligned in the vector space. Thus, the attention score will be greater if the query vector of one token is similar to the key vector of the other. The scaling expression in the denominator is helpful because the inner product yields very large values as the dimension size increases and large values yield very small gradients during training. Once we have the attention scores, we obtain the attention weights with the softmax function:

$$\alpha_{ij} = \mathrm{softmax}(a_{ij}) \tag{4}$$

and the output vector x_i for the token is simply

$$x_i = \sum_j \alpha_{ij} v_j \tag{5}$$

The process described here is applicable in the case of a single attention head. Many models make use of multi-head attention, which refers to multiple such attention mechanisms working as described but independently of each other. It is common for a multi-head attention model with h heads to use Q, K, V matrices in $\mathbb{R}^{d_k \times d}$ such that $d_k = d/h$, in which case the output vector from each head can be simply concatenated to produce a d-dimensional output vector.

The second part in the transformer block is a relatively straightforward feed-forward network (FFN), which has two linear transformations separated by a non-linear activation function such as ReLU:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{6}$$

where W_1 and W_2 are trainable weight matrices and b_1 and b_2 are trainable bias terms. It is common to regularize the FFN using the dropout method [17]. Many models set the dimensions of W_1 and W_2 such that the input vector is mapped to a higher dimension by the first transformation and then reduced to the original dimension by the second one. This allows a richer semantic representation inside the fully connected component. Intuitively, the attention mechanism retrieves all the relevant information from the context and the feed-forward network analyzes the retrieved information in depth with the higher dimension mapping and non-linearity enhancing its expressive power.

Most models also make use of residual connections in between the attention and FFN components as well as after the FFN. These connections add the input of a layer to the output of the same layer. Residual connections provide a more direct connection from the trainable parameters appearing earlier in the model to the parts closer to the final prediction. This provides better gradient updates to the parameters near the beginning during training and prevents situations in which these parameters are not learned adequately due to extremely small gradients.

Large language models run multiple transformer blocks in succession of each other in order to process data more thoroughly. For instance, the LLaMA-2 model has versions with 32, 40, and 80 transformer layers [4] and one of the larger GPT-3 [2] versions has 96 layers.

2.3 Token Prediction

The last step in the large language model comes after the series of transformer blocks. With the last transformer block yielding one enriched context vector x_i for every token t_i in the input sequence, this last step maps the vectors back into the vocabulary space and uses the softmax function to create a distribution of probabilities. Thus, for i-th position, the predicted probability vector is

$$p = \mathrm{softmax}(W_0.x_i + b_0) \tag{7}$$

where $x_i \in \mathbb{R}^d$ is the input context vector for the token from the last transformer block, $W_O \in \mathbb{R}^{V \times d}$ and $b_O \in \mathbb{R}^V$ are trainable parameters and V is the vocabulary size. The most obvious and common way to pick a token using the output probability distribution is to pick the token corresponding to the highest predicted probability. Alternative methods exist such as sampling from the multinomial distribution implied by p after filtering down only to the top k most likely tokens, called top-k sampling. [18] give a good overview of the sampling alternatives.

3 LLM-Based Autonomous Agents

Large language models have become remarkably popular in recent years in some part because they opened up the state of the art in artificial intelligence to a general audience in the user-friendly form of chatbots. There is a growing body of research which shows that, in addition to their usefulness as chatbots, the large language models can enable breakthroughs in the field of autonomous agents due to their extraordinarily strong reasoning abilities. For the purposes of this paper, an autonomous agent is defined as a system that interacts with its environment to solve a given task. The task can be quite complicated with multiple steps and the autonomous agent has to make its planning and pursue its agenda without any assistance from humans. The older autonomous agents relied on relatively simple rules and heuristics in small sandbox environments, often borrowing from traditional control theory. [9] provides a good overview of what they refer to as intelligent agents. Later on, deep learning opened up new possibilities and techniques using deep reinforcement learning, in particular, such as [19] and [20] made marked improvements on earlier methods even though they still suffered from the problem of being confined to relatively small world models. The newer LLM-based agents have innovated heavily on traditional methods and show better promise in handling complex real-world tasks and possibly growing into AGI systems in the future.

3.1 Examples of LLM-Based Autonomous Agents

AutoGPT [21] and BabyAGI [22] were two of the first LLM-based autonomous agents that were released shortly after ChatGPT. Both of these systems implemented the idea of running OpenAI's GPT model [2] multiple times in order to break down complex tasks into parts and work on the parts in an iterative fashion. While AutoGPT focuses on handling practical tasks and has the ability to navigate the Internet and perform various external actions, BabyAGI focuses more on learning, memory, sequential decision-making, and efficient task management in an effort to put forth a blueprint for a possible future AGI. A similarly early agent is HuggingGPT [23], which uses ChatGPT for planning and matching user requests with models available on the Hugging Face platform based on model descriptions in order to combine capabilities across a diversity of expertise areas and modalities such as text, vision, and speech. ViperGPT [24] is another take on multimodality, where the agent, given an image and a related question, generates Python code leveraging vision models to answer the question.

Agents focused on the software engineering process have particularly attracted a lot of attention. GPT-Engineer [25] is an agent that takes a brief description of the desired software product, asks a few clarifying questions, and builds a complete project consisting of multiple files possibly written in different languages. It also has the ability to take a sketch of the desired UI as an input image and recreate the UI in code. ChatDev [26] and MetaGPT [27] make use of multi-agent frameworks to create virtual models of complete software companies. A multi-agent framework contains multiple autonomous agents, each customized for a particular role with different priorities, capabilities, and targets. ChatDev and MetaGPT create agents with the roles of developers, QA engineers, designers, architects, and project managers, whose interactive collaboration produces the desired software program.

Games provide a relatively cheap playground in which to experiment with autonomous agents. Generative Agents [28] is a very interesting simulation of human interaction in the form of a small community of human-like agents living in a simulation environment similar to The Sims. The agents in the community have different identities and memories. They are able to remember persistent information about themselves and others, remember what happened in the past in the simulation, keep to their schedules, plan, and interact with other agents in a way that causes emergent social structure. Another example is Meta's Cicero agent [29] that plays Diplomacy, which is a turn-based Risk-like game of conquest that involves strategic calculations and delicate negotiations with other players in natural language. Cicero models what every other player is likely planning, computes its own optimal plan and uses large language models to bargain with, or manipulate, other players. Thus, it serves as an excellent example of combining other machine learning capabilities with natural language models.

Embodied agents, which are autonomous agents attached to either a physical or simulated body, have also benefited from the recent wave of integration with large language models. The DEPS framework [30] works on the problem of multi-step reasoning for long-term tasks in the Minecraft world by introducing a refinement of planning with descriptions and iterative feedback on errors. Voyager [31], on the other hand, creates embodied agents in the Minecraft world that emphasize long-term learning, development, maximization of exploration, and, notably, storage of complex behavior in libraries of executable code. SayCan [32] is an application of LLM-based agents in the world of robotics, in which rich world knowledge and semantic capabilities of LLMs are combined with the low-level physical skills of a robotic system, enabling a robotic arm to follow long-term human instructions specified in natural language. In summary, transition from simulated agents to the physical world is likely to enable a huge spectrum of potential uses for artificial intelligence systems in the near future. Especially the autonomous agents that utilize the growing capabilities of multimodal models that integrate language capabilities with other modalities such as vision and sound hold great potential for robotics.

3.2 Overall Architecture

The examples described in subsection 3.1 come with significantly varying designs, but a general architecture for a representative LLM-based autonomous agent can be seen in Figure 3. The indicated modules for Planner, Memory, and Action are the capabilities that the agent is able to draw upon when performing its tasks. The planner module is particularly important as it is the entry point for the high-level task set by a human and specified in natural language. This module decides how the task will be handled and which of the other modules to call on if needed. This module benefits significantly from the real world information and reasoning capability embedded in large language models. The memory module holds information, either persistent or produced during the agent's run, that is not contained in the LLM used by the planner. The action module serves as an actuator, acting on the environment, possibly using external tools, and returning the result of the action to the planner. It can be observed that the architecture of the autonomous agent is somewhat reminiscent of the human brain and the specialization of modules for different functionality in pursuit of efficiency is akin to different sectors of the brain having the same biological material but specializing in different tasks.

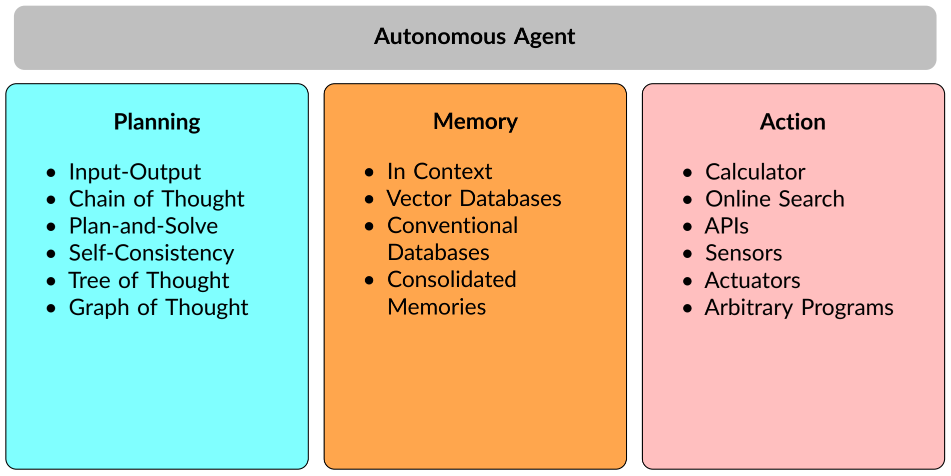

Figure 3. Components of a Representative Autonomous Agent.

In the remainder of this section, we discuss the major components of the architecture separately in greater detail.

3.3 Planning

Planning is a fundamental task for autonomous agents and it relies heavily on a human-like reasoning capability to navigate the complex landscape of real-world tasks. This aspect of the agent has significantly benefited from the advent of large language models. LLMs excel in ingesting instructions specified in natural language, modeling rich world scenarios, and exhibiting advanced reasoning capabilities, making them ideally suited for planning tasks for an autonomous agent. The primary function of the planning module is to manage the agent by understanding human-supplied instructions, decomposing the high-level goal into smaller subtasks, and utilizing other modules of the agent as necessary. Depending on the complexity of the assigned tasks, this module can range from simple in structure to quite sophisticated.

The simplest possible way of planning is the Input-Output method, which is merely posing the question to an LLM and getting an answer immediately without any intermediate steps. This might be viewed as a normal usage of an LLM rather than an autonomous agent. A marked improvement on this approach is the Chain of Thought (CoT)[6], which significantly increases the correctness of answers to complex queries and provides the breakdown of complex queries that is useful for an autonomous agent. This method works by adding to the prompt a directive to break down the task into smaller subtasks such as “Let’s think step by step.” and a few worked-out examples of a complex query being decomposed into intermediate steps and solved sequentially. An intuitive understanding of why this helps the LLM might come from the observation that LLMs are designed to produce each token with the same amount of computation. An easier question can be answered with smaller amounts of computation, but a complex query requires the larger computational space that CoT provides.

Some notable variations on the CoT idea are zero-shot CoT [7], which omits the worked-out examples from the prompt with no large penalty, and Plan-and-Solve prompting [33], which records gains in math and common sense questions by requiring the LLM to make the plan first before answering. A more sophisticated approach is Self-Consistency [34] (CoT-SC). This method uses the temperature parameter of the LLM to produce multiple different plans and then chooses steps that are supported by the most plans in a majority vote setting. This kind of planning ends up being more creative and takes into account the fact that multiple plans might all have valid insights into the problem.

A major improvement over the Chain of Thought was the Tree of Thoughts (ToT) [35]. The ToT method builds a tree in which the root node is problem statement and each branching node is a "thought" relating to its parent node. The generated tree is then pruned by a process which might judge a particular subtree as unpromising due to its children nodes diverging too far away from the desired result. This gives the ToT the crucial ability of backtracking from wrong paths without losing good progress already made. As a point of comparison, the Self-Consistency method also produces multiple plans but might discard plans entirely if their later stages go down a wrong path. This more iterative approach enables significant gains for ToT in problem-solving abilities and real-world applications such as robotics. A further generalization over Tree of Thought is the Graph of Thought (GoT) [36]. GoT is very similar to ToT, but it has a few key improvements, such as the inclusion of a self-refining process at every step and modeling thoughts as a general graph rather than a tree. A node in a graph might have multiple parents and GoT takes particular care to merge branches in different parts of the graph. The intuitive idea behind this method is that multiple paths of thought might lead to the same conclusions or further paths of thought and this reusability helps in efficiency and correctness. An even further step in this direction is the Hypergraph of Thoughts [37]. A hypergraph is a generalization of the concept of graphs, in which an edge can join an arbitrary number of vertices. Such a structure can potentially become useful especially in a multimodal context. Selected planning methods are summarized in Figure 4.

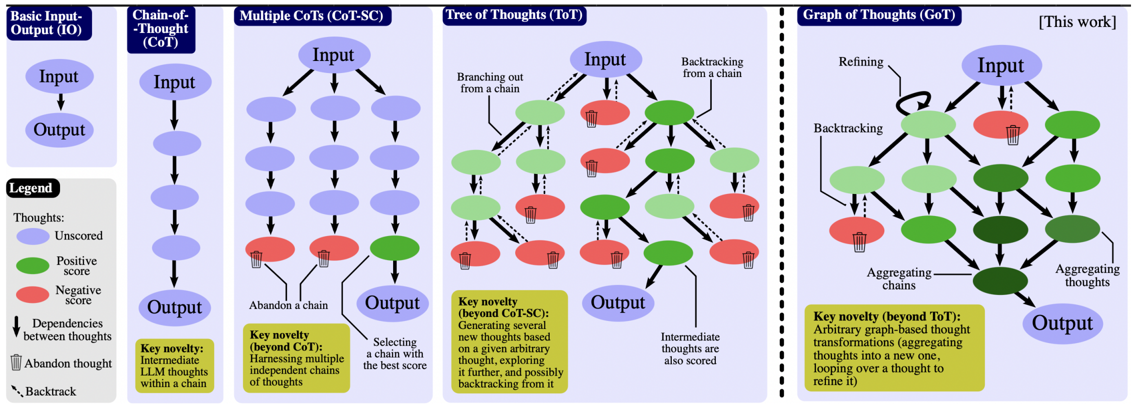
Basic Input-Output (IO)
Input
Output
Legend
Thoughts:
Unscored
Positive score
Negative score
Dependencies between thoughts
Abandon thought
Backtrack
Chain-of--Thought (CoT)
Input
Output
Key novelty: Intermediate LLM thoughts within a chain
Multiple CoTs (CoT-SC)
Input
Abandon a chain
Output
Selecting a chain with the best score
Key novelty (beyond CoT): Harnessing multiple independent chains of thoughts
Tree of Thoughts (ToT)
Input
Branching out from a chain
Backtracking from a chain
Output
Intermediate thoughts are also scored
Key novelty (beyond CoT-SC): Generating several new thoughts based on a given arbitrary thought, exploring it further, and possibly backtracking from it
Graph of Thoughts (GoT)
[This work]
Refining
Input
Backtracking
Aggregating chains
Aggregating thoughts
Output
Key novelty (beyond ToT): Arbitrary graph-based thought transformations (aggregating thoughts into a new one, looping over a thought to refine it)

Figure 4. Different Planning Frameworks. Image reproduced with permission from Besta, Blach et al. (2024) [36].

A crucial aspect of autonomous agents is their ability to use external tools as portrayed in Figure 3. The planner of the autonomous agent must, therefore, be aware of the existence and purpose of the available tools and be ready to use them if appropriate. One popular way of integrating external tools with a CoT agent is the ReAct framework [38]. ReAct, which stands for reason and action, interleaves reasoning and acting steps, thereby giving the agent ability to evaluate the results of actions and plan further at each step. This method uses directives and examples in the prompt to make LLM output Thought - Action - Observation steps at each stage. The Thought part reasons and identifies an action to take. Action executes the action and Observation is an evaluation of the result of the action taken. Thought and Observation steps are generated by the LLM whereas Action is executed externally. In this framework the agent chooses external tools based on the context in the prompt, which includes a brief description of every tool as well as instructions for utilization. This iterative method of using tools and making new plans based on results provides a significant capability boost to autonomous agents.

3.4 Memory

The memory component may contain records of external information or recollections of the history of the agent and, therefore, is an important part of the autonomous agent. Efficiently organized memory structures allow an agent to “remember” past observations and actions as well as any type of knowledge not already included in the base large language model. This section reviews the different ways of storing and accessing memories.

An easy way of integrating memory is to inject what needs to be remembered into the prompt of the LLM. This can be viewed as a short-term memory as the information in the prompt is immediately available to the model and is not persisted in another storage tool. While convenient, this method is restricted by the relatively narrow context window of the currently available large language models. Continuing the metaphor of human memory, we can consider as long-term memory the use of external storage. Some examples of storage types might be conventional and NoSQL databases or vector databases that hold embedding vectors for bits of text. The use of vector databases to hold information is sometimes called Retrieval Augmented Generation (RAG) [39]. In this framework, the agent has to find the memory most relevant to the task at hand and retrieve it as natural language to be added to the prompt of the LLM in the next step. Finding the most relevant memory is a problem remarkably similar to the task of attention mechanism described in subsection 2.2, which computes how much attention to pay each token in the context based on the input. The solution is likewise similar. For an input query q and a memory set M, all produced with the same embedder, the most relevant memory for the given query is

$$m^* = \arg\max_{m \in M} q \cdot m \tag{8}$$

where $\cdot$ is the inner product operator. Most vector databases use efficient Maximum Inner Product Search algorithms that run such optimizations with reasonable algorithmic complexity.

It is often useful to reflect on memories and store the result of the reflection rather than raw memories themselves. One advantage of such reflection is that the summarized information can be much smaller, making storage and later retrieval much cheaper. Another advantage is that the model could glean valuable high-level insights from raw observations that could not be observed by themselves. One such memory structure is demonstrated in Generative Agents [28], described in subsection 3.1, which creates a community of individual agents living in a virtual town, all with their own personality and memory. Every agent in this world takes time to reflect on what it observes and synthesizes a consolidated higher-level thought, called a reflection. To give a concrete example, if agent A observes agent B reading books about architecture, agent A might reflect that “Agent B is interested in architecture.” Such consolidation creates more useful information for the agent as well as reducing the cost of storage significantly. Another noteworthy innovation is the Reflexion framework [40], which uses self-reflection to gain an understanding of its own past errors and improvement opportunities and can store its conclusions in a long-term database. Such input allows it to make better plans in the future, enabling steady long-term improvements for the agent.

3.5 Action

The ability to act on or get information from the external world is one of the most useful aspects of autonomous agents in real world settings. The set of available actions might be baked into the construction of the agent, such as a robotic arm, or it might include the use of arbitrary computer programs as tools. Just as the usage of tools is extremely valuable for humans, it also unlocks lots of new capabilities for an autonomous agent. The action module is usually called once the planner of the agent decides to perform an action or use a tool with a process such as ReAct [38], described in subsection 3.3, and forwards the context to the action component. In addition to performing the action, this module is also responsible for the generation of action inputs, which might be commands or parameters that most tools require. For example, given a decision to use an Internet search engine and the context, the agent needs to generate the search query to be passed to the search engine.

The set of available actions is usually given to the agent as part of the initial prompt and can include a wide variety of prepared functionality. One action might be to simply use an LLM, which can be either the same one as the planner or another one specialized for certain areas of expertise such as law or medicine. HuggingGPT [23], described in subsection 3.1, is an example in which the agent chooses a model to use from the online repository of Hugging Face. Another popular external tool family is mathematical software. Since mathematical operations are not the strong suite of large language models by their nature [41], an agent can, for instance, write down a complex arithmetic expression and pass it to a calculator. TPTU [42] builds on this idea and creates a framework in which the agent can compose and execute Python code to compute answers to mathematical problems. Another family of tools that have seen widespread usage are APIs, or Application Programming Interfaces. These APIs can have arbitrary functionality, including supply of information or acting in various ways on the world. Toolformer [41] and API-Bank [43] are two noteworthy systems that use LLMs to pick which APIs to use from a large set of available APIs and formulate valid queries to the chosen API. Since the action module is responsible for most of the external effects of an autonomous agent, care should be taken in picking which functionality is made automatically available to the agent without human oversight. While getting weather information or modifying a database can be relatively safe, it is still a touch too early to give an agent the ability to launch nuclear weapons.

4 Discussion and Conclusion

The previous sections described large language models and trends in autonomous agents in some detail. This section will conclude by pointing out some prevalent themes in the research and discussing some possible future trends in the field.

A major theme in the research so far has been the great value of utilizing LLMs relatively frequently during the operation of an autonomous agent. The use of an LLM is analogous to thinking for the agents and if the agent "thinks" at each step of the plan, after the output of an action, and frequently at critical points, the end result seems to improve significantly. Such frequent use of LLMs might also be an antidote to the serious problem of error propagation, which refers to the agent making an error at a certain stage and having its subsequent steps steadily diverge from useful paths [41]. The obvious downside to using LLMs at every opportunity is that they are still relatively expensive to run. Error propagation and various other modes of failure during the operation of the agent, such as problems interfacing with external tools due to wrong parameters or invalid JSON, also depend on the quality and correctness of LLMs. As better and cheaper are created over time, autonomous agents will become much more robust and new avenues of planning and executing will be unlocked.

Another theme with most deep learning research so far and especially prominent in autonomous agents has been the obvious analogies with the human brain. The fundamental deep learning building blocks such as artificial neural networks, the activation functions therein, and attention are inspired by biological constructs. Additionally, the overall architecture of the agent has a lot of similarities with a model of the human brain. Planning, for example, can be seen as standing for the prefrontal cortex and long term memory reminds one of the hippocampus. The analogy will likely persist and future innovations will draw heavily from the human brain functionality that is so far missing from AI systems. Some candidates are creativity, habit formation, spatial reasoning, and sensory information. Multimodal agents, in particular, seem like one of the nearest innovations and they will enable systems that can more easily interact with the environment with the help of a broad range of sensors.

As autonomous agents have evolved from experimental constructs to systems with real value to users, we are likely to see a widespread use in some commercial areas such as customer support. This will create opportunities for agents to reflect on and learn from previous experience, using frameworks like Reflexion [40]. Such real-world feedback might accelerate the improvement of agents considerably. Even more complex scenarios might arise when cooperation with humans and other autonomous agents with different specializations is allowed. Naturally, such working models would also necessitate guarding against risks as autonomous agents might interact with each other in unpredictable ways and humans can provide adversarial input. Particular attention should be paid to the use of external tools as this part of the autonomous agents has the potential to cause most harm. Human oversight is likely to remain necessary for any action that has the potential to cause undesirable effects.

This paper described large language models and how autonomous agents leverage these models to undertake complex tasks with enhanced capability and robustness. The analogy between the autonomous agents described and the human brain is apparent and many applications are on the road to approach human-like performance even if in limited settings. Thus, LLM-based autonomous agents constitute a seemingly viable path to AI systems with high real-world value and, eventually, artificial general intelligence.

References

[1] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. Advances in neural information processing systems. 2017;30.

[2] Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, et al. Language models are few-shot learners. Advances in neural information processing systems. 2020;33:1877-901.

[3] Chowdhery A, Narang S, Devlin J, Bosma M, Mishra G, Roberts A, et al. Palm: Scaling language modeling with pathways. Journal of Machine Learning Research. 2023;24(240):1-113.

[4] Touvron H, Martin L, Stone K, Albert P, Almahairi A, Babaei Y, et al. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:230709288. 2023.

[5] Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, Aleman FL, et al. Gpt-4 technical report. arXiv preprint arXiv:230308774. 2023.

[6] Wei J, Wang X, Schuurmans D, Bosma M, Xia F, Chi E, et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems. 2022;35:24824-37.

[7] Kojima T, Gu SS, Reid M, Matsuo Y, Iwasawa Y. Large language models are zero-shot reasoners. Advances in neural information processing systems. 2022;35:22199-213.

[8] Wei J, Tay Y, Bommasani R, Raffel C, Zoph B, Borgeaud S, et al. Emergent abilities of large language models. arXiv preprint arXiv:220607682. 2022.

[9] Russell SJ, Norvig P. Artificial Intelligence: A Modern Approach. 3rd ed. Prentice Hall; 2009.

[10] Schuster M, Nakajima K. Japanese and Korean Voice Search. In: International Conference on Acoustics, Speech and Signal Processing; 2012. p. 5149-52.

[11] Sennrich R, Haddow B, Birch A. Neural machine translation of rare words with subword units. arXiv preprint arXiv:150807909. 2015.

[12] OpenAI. tiktoken is a fast BPE tokeniser for use with OpenAI's models; 2024. Version 0.7.0. https://pypi.org/project/tiktoken/.

[13] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. arXiv preprint arXiv:13013781. 2013.

[14] Su J, Ahmed M, Lu Y, Pan S, Bo W, Liu Y. Roformer: Enhanced transformer with rotary position embedding. Neurocomputing. 2024;568:127063.

[15] Ba JL, Kiros JR, Hinton GE. Layer normalization. arXiv preprint arXiv:160706450. 2016.

[16] Brandon W, Mishra M, Nrusimha A, Panda R, Kelly JR. Reducing Transformer Key-Value Cache Size with Cross-Layer Attention. arXiv preprint arXiv:240512981. 2024.

[17] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 2014;15(56):1929-58. Available from: http://jmlr.org/papers/v15/srivastava14a.html.

[18] Nadeem M, He T, Cho K, Glass J. A systematic characterization of sampling algorithms for open-ended language generation. arXiv preprint arXiv:200907243. 2020.

[19] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature. 2015;518:529-33. Available from: https://api.semanticscholar.org/CorpusID:205242740.

[20] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, et al. Continuous control with deep reinforcement learning. arXiv preprint arXiv:150902971. 2015.

[21] Gravitas S. AutoGPT; 2024. Accessed: 2024-06-15. https://github.com/Significant-Gravitas/AutoGPT. Available from: https://agpt.co.

[22] Nakajima Y. BabyAGI; 2024. Accessed: 2024-06-15. https://github.com/yoheinakajima/babyagi.

[23] Shen Y, Song K, Tan X, Li D, Lu W, Zhuang Y. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. Advances in Neural Information Processing Systems. 2024;36.

[24] Surís D, Menon S, Vondrick C. Vipergpt: Visual inference via python execution for reasoning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision; 2023. p. 11888-98.

[25] Osika A. gpt-engineer; 2023. Accessed: 2024-06-15. https://github.com/gpt-engineer-org/gpt-engineer. Available from: https://gpt-engineer.readthedocs.io.

[26] Qian C, Cong X, Yang C, Chen W, Su Y, Xu J, et al. Communicative agents for software development. arXiv preprint arXiv:230707924. 2023.

[27] Hong S, Zhuge M, Chen J, Zheng X, Cheng Y, Zhang C, et al.. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework; 2023.

[28] Park JS, O'Brien J, Cai CJ, Morris MR, Liang P, Bernstein MS. Generative agents: Interactive simulacra of human behavior. In: Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology; 2023. p. 1-22.

[29] (FAIR)† MFARDT, Bakhtin A, Brown N, Dinan E, Farina G, Flaherty C, et al. Human-level play in the game of Diplomacy by combining language models with strategic reasoning. Science. 2022;378(6624):1067-74. Available from: https://www.science.org/doi/abs/10.1126/science.ade9097.

[30] Wang Z, Cai S, Chen G, Liu A, Ma X, Liang Y. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. arXiv preprint arXiv:230201560. 2023.

[31] Wang G, Xie Y, Jiang Y, Mandlekar A, Xiao C, Zhu Y, et al. Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv preprint arXiv: Arxiv-230516291. 2023.

[32] Ahn M, Brohan A, Brown N, Chebotar Y, Cortes O, David B, et al. Do As I Can and Not As I Say: Grounding Language in Robotic Affordances. In: arXiv preprint arXiv:2204.01691; 2022. .

[33] Wang L, Xu W, Lan Y, Hu Z, Lan Y, Lee RKW, et al. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. arXiv preprint arXiv:230504091. 2023.

[34] Wang X, Wei J, Schuurmans D, Le Q, Chi E, Narang S, et al. Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:220311171. 2022.

[35] Yao S, Yu D, Zhao J, Shafran I, Griffiths T, Cao Y, et al. Tree of thoughts: Deliberate problem solving with large language models. Advances in Neural Information Processing Systems. 2024;36.

[36] Besta M, Blach N, Kubicek A, Gerstenberger R, Podstawski M, Gianinazzi L, et al. Graph of thoughts: Solving elaborate problems with large language models. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 38; 2024. p. 17682-90.

[37] Yao F, Tian C, Liu J, Zhang Z, Liu Q, Jin L, et al. Thinking like an expert: Multimodal hypergraph-of-thought (hot) reasoning to boost foundation modals. arXiv preprint arXiv:230806207. 2023.

[38] Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, et al. React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:221003629. 2022.

[39] Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems. 2020;33:9459-74.

[40] Shinn N, Labash B, Gopinath A. Reflexion: an autonomous agent with dynamic memory and self-reflection. arXiv preprint arXiv:230311366. 2023.

[41] Schick T, Dwivedi-Yu J, Dessì R, Raileanu R, Lomeli M, Hambro E, et al. Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems. 2024;36.

[42] Ruan J, Chen Y, Zhang B, Xu Z, Bao T, Du G, et al. Tptu: Task planning and tool usage of large language model-based ai agents. arXiv preprint arXiv:230803427. 2023.

[43] Li M, Song F, Yu B, Yu H, Li Z, Huang F, et al. Api-bank: A benchmark for tool-augmented llms. arXiv preprint arXiv:230408244. 2023.