A Model Predictive Control & Deep Q Learning Approach to Wayfinding

Jad Soucar (jadsoucar@gmail.com)
USC Viterbi Daniel J. Epstein Department of Industrial
Systems Engineering 3715 McClintock Avenue, GER 240
Los Angeles, CA 90089-0193

November 29, 2024

Mentoring Organization: Red Hen Lab
Keywords: Reinforcement Learning, Agent Modeling, Control Theory, Wayfinding

1 Summary

For the last 200 years research into human cognition and decision-making has revolved around rational choice theory, which assumes that actors are utility maximizers capable of searching and finding rational and optimal decisions. However human behavioral data doesn't support the assumptions that we have infinite time nor infinite cognitive energy to search the full space of candidate choices to produce optimal decisions. Take for example a hiker, who has an infinite number of paths to choose from. It's irrational to assume that the hiker is capable of fully evaluating each of the infinite paths available to them to then choose the most rational/ optimal path. And yet that is the operating assumption for most modern cognitive models. To remedy this problem, I propose three axiomatic principles of energy-efficient decision making, along with an optimal-control based model and Deep Q learning model that capture those principles. I hope that a formalization of my theory of energy-efficient decision making along with the open-source python code I develop can be used by researchers in private and academic settings to develop more accurate computational models of human decision making, that can be used to model agent behavior in a number of fields ranging from market behavior to crowd dynamics.

Beyond classical applications to human and animal way finding, I believe that MPC and Deep Q learning based way finding models I will develop throughout this proposal have potential applications to the development of large language models. Currently LLM's operate using auto-regressive frameworks that are capable of searching the possibility space of the "next word" or token. The model then chooses the most likely next token based on the context embedded in the previous sequence of tokens. However the MPC and Deep Q frameworks Ill develop bellow have the potential capability to cost effectively search the space of the next N tokens to more efficiently plan the sequence of tokens being generated. While applications to LLMs and other generative AI algorithms may seem particularly interesting to the reader, I've chosen to formalize our theory of decision making by analyzing the classical way finding problem of foraging, since foraging is an especially intuitive example. Before ending with an explanation as to how the two models can be generalized to LLMs.

Proposal Structure

I will begin by proposing 3 principles of energy-efficient decision before introducing some relevant mathematical background for their computational implementation. I end by exploring two possible models of agent-based wayfinding that incorporate the 3 principles referenced above along with their applications.

2 Proposed Principles of Energy-Efficient Decision Making

Actors engaged in wayfinding will oftentimes miss the optimal trajectory towards their target, whether that be food, shelter or a mate, due to cognitive energy constraints. In order to construct mathematical models around this idea, I will first break down the concept of costly cognition, along with its natural consequences.

1. **Costly Cognition**: Recent literature suggests that cognition requires 20% of the body's total energy consumption. It's these energy constraints that limit an actor's ability to fully flush out a set of candidate decisions before adopting an action. Instead, a finite cognitive energy budget forces actors to develop rough sketches of various candidate decisions. Each of these sketches define a subspace of more detailed decisions that fall within the category of the decision sketch. In other words, cognitive energy constraints improve the efficiency of the decision-making process by constraining the space of possible decision, but can also guide actors towards energy-efficient choices by eliminating an optimal decision space during the initial screening process.

2. **The Subspace Selection Problem**: The costly cognition principal forces actors to develop decision subspaces in order to reduce cognitive energy load. The natural question is "How is that subspace selected?". I propose that the decision subspaces are evaluated using fuzzy and incomplete heuristics. The internal system then weighs the probability of success relative to the cognitive energy associated with exploring each of those subspaces. Only after this process is complete will an actor elect to conduct a more thorough search of a particular subspace to come to a final decision.

3. **Decision as Future Projections**: I've discussed how costly cognition forces actors to rely on rough decision sketches to choose a decision space to thoroughly explore. I propose that a human decision maker evaluates those sketches by constantly projecting themselves into possible futures based on those decision sketches. Each projection is made onto a temporal window chosen by the actor. For example, a decision made in the pursuit of a goal which the actor heavily values will encourage a projection further into the future than one of minimal importance to the actor. Regardless of the future projections size, an actor must consistently re-generate decision sketches at every time step, to adjust to new internal and external conditions. This process of constantly projecting oneself into the future to create a continuous stream of decisions produces the effect of a receding horizon. This conceptualization is in line with our intuition of decision making as a process that becomes increasingly fuzzy and uncertain the further forward in time an actor looks.

3 Mathematical Background

In this section I provide relevant mathematical foundation of deep reinforcement learning and model predictive control. In future sections I will use those two frameworks to construct two candidate models of cognitively constrained wayfinding that incorporates the 3 core principles of decision making outlined above

3.1 Deep Reinforcement Learning

3.1.1 General Reinforcement Learning

Before I begin my discussion of Q learning, I will start with an exposition of the two primary categories of RL: model-based and model-free. Model-based RL systems create a model of the environment, which includes the dynamics of how actions lead to subsequent states and rewards. Essentially, it predicts the future states and rewards for actions taken from a given state. Take for example a gambler who is given a probability function $P(s',s,a)$ that provides explicit probabilities as to whether the actor will win or lose a gamble. In other words, the gambler queries the environment for its transition dynamics. The second approach is model-free, which is where a system learns a policy or value function directly from interactions with the environment without constructing an explicit model of the environment's dynamics. The system learns what to do by trial and error, adjusting its actions based on the rewards received.

3.1.2 Q Learning

With the general background of RL out of the way, its important to note that Q learning is a model-free training scheme. That means that Q learning does not rely on an environment that can produce exact transition probabilities and must instead use trial and error of near-optimal actions in order to learn the optimal policy. In order to formalize this type of training scheme I first define the quality function $Q(s,a)$, which tells you the joint value/quality of taking an action a given a current state s.

To formally define $Q(s,a)$ I first introduce some notation. Let $R(s',s,a)$ be the reward of transitioning from state s to s' through action a, γ be the discount factor of future reward, and $V(s')$ be the expected value over all possible future rewards given a current state s'. With that I define $Q(s,a)$ as

$$Q(s,a) = \mathbb{E}\left(R(s',s,a) + \gamma V(s')\right) = \sum_{s'} P(s'|s,a)(R(s',s,a) + \gamma V(s')) \tag{1}$$

In other words, Q is the expected sum of the instantaneous reward of the state-action pair (s,a) along with the discounted future rewards of being at a new state s' brought on by (s,a). Using the quality function $Q(s,a)$ we can define an optimal policy π and value function $V(s)$, that considers which action a is optimal and what the expected reward from taking that action is.

$$V(s) = \max_a Q(s,a) \tag{2}$$

$$\pi(s) = \arg\max_a Q(s,a) \tag{3}$$

We've defined what a q value is and how to construct the quality function $Q(s,a)$, but now we define a recursive equation to update $Q(s,a)$ as the agent learns through trial and error.

$$Q^{\text{new}}(s_k,a_k) = Q^{\text{old}}(s_k,a_k) + \alpha\left(r_k + \gamma \max_a Q^{\text{old}}(s_{k+1},a) - Q^{\text{old}}(s_k,a_k)\right) \tag{4}$$

Let's dissect this update equation. As we engage in trial and error learning, we update our $Q(s,a)$ by slowly nudging our q values up or down by a factor of the difference between the actualized current reward r_k in addition to the best possible future reward $r_k + \gamma \max_a Q(s_{k+1},a)$ (TD-Target) and our predicted reward $Q^{\text{old}}(s_k,a_k)$). This difference is sometimes referred to as the "Temporal Difference (TD) error". We also note that α in this case is the learning rate.

$$\underbrace{\overbrace{r_k + \gamma \max_a Q(s_{k+1},a;\theta^-)}^{\text{TD Target}} - Q^{\text{old}}(s_k,a_k;\theta)}_{\text{TD Error}} \tag{5}$$

We also note that when calculating the TD error, the agent calculates the future reward by looking one step into the future. Naturally, the degree to which the agent looks into the future can be modified using the following updated scheme which looks n steps into the future. This process is known as TD-N learning.

$$Q^{\text{new}}(s_k,a_k) = Q^{\text{old}}(s_k,a) + \alpha(r_k + R_\Sigma^{(n)} - Q^{\text{old}}(s_k,a_k)) \tag{6}$$

$$R_\Sigma^{(n)} = (\sum_{j=1}^{n} \gamma^j r_{k+j}) \tag{7}$$

where r_{k+j} is the reward derived from future applications of the agent's chosen policy π.

3.1.3 Deep Q-Learning

Now we discuss the process of recasting Q Learning into training schemes that incorporate deep learning methods like neural networks. This is especially useful in settings where the state space is too large to reasonably store all q values in a Q table. Here, a deep learning approach seeks to parameterize $Q(s,a)$ as dependant on some weights θ such that

$$Q(s,a) \approx Q(s,a,\theta) \tag{8}$$

In order to optimize for the parameters θ within a Q Learning framework we use a loss function that is eerily similar to the TD error defined in equation (4).

$$\mathcal{L} = \mathbb{E}[(r_k + \gamma \max_a Q(s_{k+1},a,\theta) - Q(s_k,a_k,\theta))^2] \tag{9}$$

We can also choose to recast n-step temporal difference learning framework (TD-N) described above into the following neural network loss function.

$$\mathcal{L} = \mathbb{E}[(r_k + R_\Sigma^{(n)} - Q(s_k,a_k))^2] \tag{10}$$

With the loss function properly defined, we can move on to the neural network architecture, and how it allows us to approximate the Q function. In practice, we find that the architecture of the network varies based on the use case. For example a team of DeepMind engineers in 2013 coupled the loss function described above with several convolutions layers and fully connected layers. The inputs were several consecutive frames for the game, which represented the agent's state. And the output was one of several possible action that the agent could take. Regardless of the architecture used the value contained in each output node approximates the value $Q(s,a)$ for the associated (s,a) which corresponds to the network's $(\text{Input}, \text{output})$ pair.

3.1.4 Policy

In practice, we follow the ε-greedy policy to catalyze off-policy actions by introducing randomness. Formally ε-greedy is implemented by replacing the $\max_a Q(s_{k+1},a)$ term in equation 5 with $Q(s_{k+1},\tilde{a}_{k+1})$ where

$$\tilde{a}_{k+1} = \begin{cases} \text{random action from } A(s_t), & \text{with probability } \varepsilon \\ \arg\max_a Q(s_t,a), & \text{with probability } 1-\varepsilon \end{cases} \tag{11}$$

As training continues we slowly take $\varepsilon \to 0$ using a simulated annealing strategy. The reason the ε-greedy strategy can be beneficial is that it allows the agent to explore the space of possible actions freely at the beginning of the training process while also emphasizing exploitation of the agent's accumulated knowledge toward the end of training. Please note that for the purpose of feeding the state into the deep Q network, we must first normalize by applying a soft max function to each column of the matrix s_t and then flatten the matrix.

3.1.5 Loss Function

We've defined the DQN loss function in equation (9) and (10) but in practice the loss is implemented using various approximations. First the expectation in the Q Learning loss function is approximated using experience replay by sampling a mini-batch of experiences from the replay buffer. Given a replay buffer D that contains experiences (s,a,r,s'), a mini-batch of N experiences $B = \{(s_i,a_i,r_i,s_i')\}_{i=1}^N$ is sampled uniformly at random from D. Using this strategy we can approximate the loss function in equation (9) as follows.

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\left(r_i + \gamma \max_{a'} Q(s_i',a';\theta) - Q(s_i,a_i;\theta)\right)^2\right] \tag{12}$$

The next adjustment practitioners use is the integration of the the Huber loss. This is designed to make our loss function quadratic for small values of the error and linear for large values of the error. The parameter δ effectively determines the sensitivity of the loss function to outliers. Previously we defined TD Error in

equation (5), which for the sake of integrating the Huber loss we'll equate to δ. This final improvement yields the following loss function.

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} L(\delta) \tag{13}$$

where

$$\delta = \left(r + \gamma \max_{a'} Q(s', a')\right) - Q(s, a) \tag{14}$$

$$L(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases} \tag{15}$$

3.1.6 Optimizer

Lastly, in practice the Adam optimizer is used to minimize the loss function. This is an adaptive optimization algorithm used in training our neural networks. It incorporates momentum to accelerate convergence in relevant directions and introduces bias correction to improve parameter update accuracy. By computing adaptive moments of gradients, Adam scales updates effectively and ensures numerical stability with a small constant in the denominator. These combined features make Adam a robust and efficient choice for optimizing our neural networks.

3.2 Model Predictive Control

The general formulation of MPC involves solving an optimization problem at each control step. The objective is to find the control inputs that minimize the objective function subject to the model dynamics and constraints.The general setup for an MPC problem is as follows.

1. **Choice Functional**:

$$J(U_t^{t+\Delta}, x_t) = \sum_{k=t}^{t+\Delta} l(x(k), u(k)) + F(x(T)) \tag{16}$$

   Where:

   - J: The choice function to minimize.
   - $u(t)$: The discrete function of control inputs
   - $U_t^{t+\Delta}$: A sequence of future control inputs $u(t), u(t+1), \ldots, u(t+\Delta)$. For the sake of simplicity the subscript of U denotes when the sequence begins, and the superscript denotes when it ends.
   - x_t: The current state of the system.
   - l: The stage cost function, evaluating the cost of each predicted state and control action.
   - F: The terminal cost function, evaluating the cost of the final state.
   - T: Terminal Time End of Episode.
   - Δ: Predictive Horizon

2. **Subject to**:

   - System dynamics: $x(k+1) = f(x(k), u(k))$
   - Initial condition: $x(0) = x_0$
   - Control and state constraints: $u(k) \in A, x(k) \in S$, where A is the action space and S is the state space.

3. **Iterative Algorithm:** The user then iteratively implements the following algorithm to trace the evolution of an agent's state from a set of initial conditions to a desired terminal state.

(a) Initialize the agent at state x_0

(b) Optimize a choice functional over the predictive horizon. In other words find $\tilde{u} = \arg\min_U J(U_0^\Delta), x_0)$ to determine the next "best course of action" U_0^Δ

(c) Apply the system dynamics to determine the next state $x_1 = f(x_0, \tilde{u}(0))$. *Note that despite finding a sequence of controls from* 0 *to* Δ*, the agent only applies the first control in the sequence.*

(d) Reinitialize the agent at x_1, and optimize the choice functional $J(U_1^{1+\Delta}, x_1)$ over the time horizon.

(e) Iteratively apply steps (a)-(c) until the terminal time N is reached.

4 Candidate Wayfinding Models

In this section we propose two wayfinding models. One model will be built using a model predictive control (MPC) framework, while the other builds upon principles from Reinforcement Learning (RL). The two models take slightly different approaches to the wayfinding problem. The core difference being that the RL system focuses on single priority agents with future projections of variable size, while the MPC model focuses on multi-priority agents with future projections of fixed size. Before we discuss the architecture of the two candidate models, we'll begin with some core assumptions.

4.1 Core Assumptions

We begin by outlining several core operating assumption that will be used for both the MPC and RL models.

First we assume that an agent is capable of traversing a $d = 2,3$ dimensional rectangle. We may choose to define a topology G within $\mathcal{D}$. G may be mountainous, flat, or another terrain the user chooses.

$$\mathcal{D} = [-\delta, \delta]x...x[-\delta, \delta] \in \mathbb{R}^d \tag{17}$$

where food sources F is a finite set of coordinate in $\mathcal{D}$ such that every $f \in F$ is picked from D according to a uniform distribution $f \sim U(\mathcal{D})$. We also assume that the space $\mathcal{D}$ is populated with finite environmental information. We define a set of environmental stimuli where i_n is an environmental stimulus like scent, auditory noise, temperature, or a binary flag b for whether food exists at a specific coordinate, and x is that information's coordinate position in $\mathbb{R}^d$.

$$\forall i \in I, i = \{i_1, ..., i_n, b, x \in \mathbb{R}^d\} \tag{18}$$

Next we assume that the agent is equipped with energy that can be spent on taking an action or on the cognitive cost of deciding which action to take. So we let the agent's energy at time t be E_t.

$$E_t \in [0, 1] \tag{19}$$

Next we assume that an agent is capable of collecting information from its environment. However we only allow the agent to collect information from within a radius r around its current location. This is a realistic restriction since no agent can reasonably be assumed to have complete and perfect knowledge of its environment. So the information that an agent can access at any given time is defined as follows, where $s_{t,x}$ is the agents current position in $\mathbb{R}^d$

$$I_{r,t} = \{i = \{i_1, ..., i_2, x \in \mathbb{R}^d\} \in I \text{ if } |x - s_{t,x}| \leq r\} \tag{20}$$

Using the assumptions outlined above we can define the agents state s_t as a tuple of the agent's energy, current information, and current position in space at a given time t. The state space S contains all possible s_t.

$$s_t = \{E_t, I_{r,t}, s_{t,x}\} \tag{21}$$

The action space A will be defined as follows, where ϕ is the magnitude of the agent's step and θ is the direction of the step. For the purposes of our simulation, both θ and ϕ are discretized.

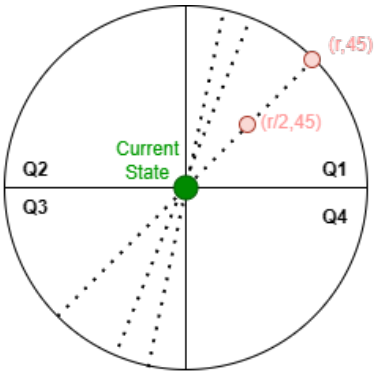

Figure 1: Action Space Visualization

$$\forall a \in A, a = \{\theta, \phi\} \tag{22}$$

If an agent takes an action $a = \{\theta, \phi\}$, we assume that the energy expended on taking the action is directly tied to the terrain G, since walking up a hill or mountain would naturally require more energy then walking downhill or on flat ground. We define this change in energy using the following dynamics:

$$c(s_{t,x}, a_t) = -\beta \nabla G(s_{t,x} + a_t) \tag{23}$$

Finally the reward function will be a piecewise function that represents how close an agent is to its goal. Note that the agent receives a significantly larger reward once it actually attains its goal, which we define as the agent reaching the source within a small margin ε. Conversely, if the energy is depleted below some threshold T_E, the agent receives a negative reward, to disincentives a zero energy state, which constitutes death.

$$R(s_t) = \begin{cases} 1 & \text{if } \exists f \in F \text{ s.t } |f - s_{t,x}| < \varepsilon, \\ -\beta_1 s_{t,E} & \text{if } s_{t,E} < T_E, \\ \beta_2 |f - s_{t,x}| & \text{otherwise.} \end{cases} \tag{24}$$

4.2 Deep Q Learning Wayfinding Model

We now transition to a possible application of the Deep-Q Learning Framework to the wayfinding problem. For our purposes we use Q-Learning since its model-free nature more accurately reflects an actor's uncertainty regarding the exact outcomes of its actions when engaging in the wayfinding process. Using the state space, action space, environment, and reward function described above in section 4.1 we construct the following deep Q learning architecture.

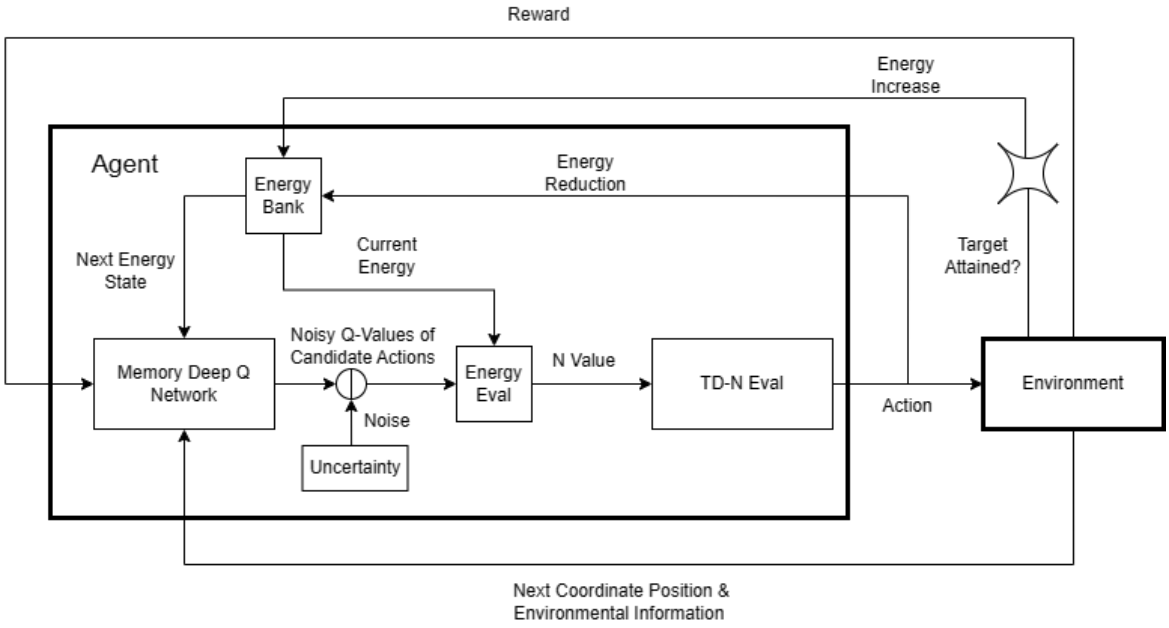

Figure 2: RL Framework

1. We begin be giving the agent prior knowledge of it environment through a fuzzy memory. This fuzzy memory functionally serves as the agent's intuition. To do this we train a Deep Q Network using the TD-N loss function defined in equation (10), along with the experience replay and Huber loss defined in section 3.1.5. We note that for training the memory deep Q network we fix N. We also use an epsilon greedy policy with an epsilon that converges to 0.33 towards the end of the training epoch. Intuitively the reason for such a high epsilon is to ensure that the agent's memory deep q network is populated with a wide breadth of experiences. The algorithm in trained using a the classical vanilla deep Q network architecture diagrammed bellow.

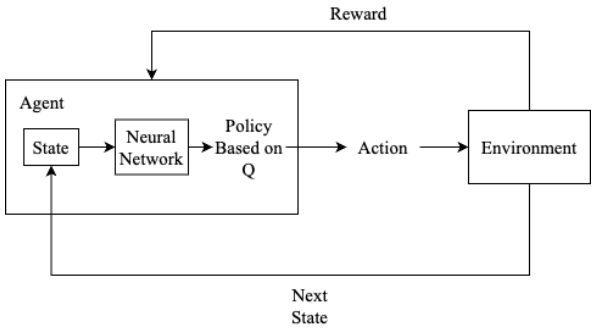

Figure 3: V-DQN Framework

We define the neural network unit in the diagram above to have the convolutional architecture outlined in figure 4. The reason why we choose a convolutional architecture is because the environmental information along with the agent's current coordinate position is inherently spatial and ought to be represented using a tensor in $\mathbb{R}^d$ instead of a flattened vector. We also exclude the energy E_t from the initial input, and instead only input $s_{t,x}$ and $I_{t,x}$. Because the energy cannot be spatially represented its concatenated to the flattened vector labeled bellow.

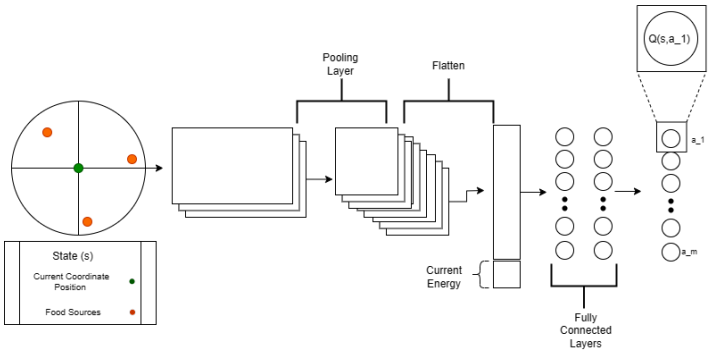

Figure 4: V-DQN Framework

Note that the memory deep network module is simply an approximation of a the Q function which takes an input of a state and outputs a estimated value of that state given each action in the action space. However because memory is often fuzzy and realistic agents don't have perfect recall we add slight Gaussian noise to the output. We define the Q function approximated by the Memory Deep Q Network as Q^{MB}

$$Q^{\text{MB}}(s) = \{(s,a,q(s,a_1)),(s,a,q(s,a_2)),...,(s,a_m,q(s,a_2))\} + \mathcal{N}(0,\sigma) \tag{25}$$

2. Next we initialize our agent with an initial coordinate position in $\mathbb{R}^d$, and a complete energy of $E_0 = 1$. Given that the agent is initialized with a random initial coordinate position, the agent will consequently gain access to all information within a radius r around its initial position. In other words the agent begins the simulation with a initial state of s_0 as defined above.

3. The agent then queries its memory bank for noisy q-values of all possible actions given its state. Those values are then passed to the energy evaluation unit. Within the energy evaluation unit the agent evaluates the average possible reward of each quadrant. This done by calculating quadrant-wise q-value averages. Then based on the magnitude of the possible reward the agent assigns an N value for each quadrant. Next the energy evaluation unit measures whether the cognitive cost of projecting forward N steps in each quadrant is less then the projected reward. If the cognitive cost is larger, the agent ignores that quadrant. Furthermore, if the amount of energy E_t is too low to afford exploring all promising quadrants, only the top K quadrants are chosen. Intuitively what that means is that the agent is only dedicating cognitive energy to create future projections of itself in decision subspaces that the agent can afford and whose potential reward outweigh the expected cognitive costs.

4. The agent then picks the actions in each quadrants selected which provide the largest q-value. Using those selected actions the TD-N evaluation unit finds the TD-N Target defined in equation (5) for each quadrant given the N value that was recommended by the energy evaluation unit. Note that the TD-N calculation is created using the policy recommended by the Memory Deep Q Network. The TD-N evaluation unit then chooses the course of action with the highest TD-N Target. The agent then applies that action to produce its next coordinate position. The energy cost of the action defined in equation (23) along with the cognitive energy used during the TD-N evaluation process (which is positively correlated to the amount of quadrants explore and the magnitude of the N values for each quadrant) is subtracted from the total energy E_t stored in the Energy bank.

5. Now that the agent has applied an action which produces its next coordinate position, it can interact with the environment to update its state with the new information $I_{t,r}$ it has access to and determine whether its reward. The way we've defined the reward function in equation (24) implies that the agent will receive a higher reward the closer it is to the food source. If a food source is attained, this triggers an energy increase for the agent. If the energy is depleted below some threshold T_E, the agent receives a negative reward. The last step involves storing the state, action pair that the agent chose along with the next state

and reward that state action pair generated in a memory bank that is used to retrain the Memory Deep Q Network.

6. Periodically, the Memory Deep Q Network Q^{MB} is retrained using the new experiences it has stored in its memory bank using the mini-batching technique outlined in section 3.1.5 along with the same architecture outlined in Step 1. Intuitively these periodic retraining session can be interpreted as an agent sleeping, since recent research indicates that sleep and memory consolidation are related processes that involve integrating new information into long-term memory.

7. Note that the only terminal state for the agent is when energy reaches 0 or if the simulation runs until the max iterations set by the user.

4.3 Model Predictive Control Wayfinding Model

We now transition to a possible application of the Model Predictive Control (MPC) to the wayfinding problem. Due to the flexibility of the MPC model framework, we allow our agent to have multiple priorities. This seems like a natural extension since most wayfinders are forced to balance multiple priorities. Take for example a mammal who must balance its priorities of finding food, shelter and a mate. For the purposes of this model we impose that the agent's set of priorities F is finite.

$$F = \{P_1, P_2, ..., P_m\} \tag{26}$$

In order to account for a multi-priority agent we make several small alterations to how we define the environment, environmental information, and reward function in the section 4.1. First we adjust the environment we create m goal sets F_i for each priority each of which is a finite set of coordinates in $\mathcal{D}$ such that every $f_j \in F_i$ is picked from D according to a uniform distribution $f_j \sim U(\mathcal{D})$. For example if priority P_k is finding shelter, then there would be an associated finite set of coordinates F_k that represents where in $\mathcal{D}$ shelter exists. This means that the elements in the set of environmental stimuli I would also include binary flag b_i for whether a goal associated with each priority exists at a specific coordinate.

$$\forall i \in I, i = \{i_1, ..., i_n, b_1, b_2, ..., b_m, x \in \mathbb{R}^d\} \tag{27}$$

This adjustment means we must also add a new reward function for each priority. So our new generalized definition of the reward function defined in equation (24), would be as follows.

$$R_i(s_t) = \begin{cases} 1 & \text{if } \exists f \in F_i \text{ s.t } |f - s_{t,x}| < \varepsilon, \\ -\beta_1 s_{t,E} & \text{if } s_{t,E} < T_E, \\ \beta_2 |f - s_{t,x}| & \text{otherwise.} \end{cases} \tag{28}$$

With those core adjustments defined, we can draw our attention to the central question for multi-priority agents; How does the agent choose which priority to focus on. To answer this question we propose an activation function a_n, which measures to what extent an agent should focus on its n^{th} priority.

$$a_n(s_t) = \frac{R_n(s_t)\mathbb{P}_n(s_t)}{D(P_n)\prod_0^{n-1} R_n(s_t)\mathbb{P}_n(s_t)\prod_{n+1}^{m} R_n(s_t)\mathbb{P}_n(s_t)} \tag{29}$$

Let break down how this activation function works. The activation function is directly related to the the reward of being in a current state with respect to the n^{th} priority, weighted by the probability of that reward being attained. The probability measure $\mathbb{P}_n$ incorporates both the agent's intuitive and experiential understanding of the environment signals i_n to produce a probability of success. In other words the higher the expected reward of pursuing the n^{th} priority, the more likely that the agent will choose to pursue that priority. However, the activation function is inversely related to $D(P_n)$, which represents the cognitive complexity of attaining the goals of the n^{th} priority. This is fairly intuitive since an agent will tend to steer away from complex priorities due to their large cognitive costs unless the expected reward is sufficiently large. Lastly the activation function

of the n^{th} priority is inversely related to the expected rewards of the other priorities. In this way, the activation function also models the opportunity cost of pursuing one priority over another.

Next we construct a choice functional that must be maximized at every given time step of the simulation, similar to that defined in equation (16). Where $U_t^{t+\Delta}$ is the sequence of actions $u(k) \in A$ from t to $t+\Delta$.

$$J(U_t^{t+\Delta}, s_{t,xx}) = \sum_{k=t}^{t+\Delta} \sum_{j=1}^{m} a_j(s_t) R_j(s_{k,x}, u(k)) \tag{30}$$

Through this choice function the decision to name the a_n the activation function becomes more clear, since a_n quite literally switches certain priorities on and off. Next we define the agent's dynamics. In order to adhere to the costly cognition principle, the agent's dynamics must be directly linked to its cognitive energy consumption. In other words if the agent chooses to explore a subspace that is relatively complex, its energy state E_t must be reduced, and if a goal like finding food or shelter is reached, its energy state must be reverted to the maximum energy state of 1. We formalize the system's dynamics below.

$$\tilde{d} = \sum_{j=0}^{m} a_j(s_t) D(P_j) \tag{31}$$

$$E_{t+1} = H(s_t, a_t) \tag{32}$$

$$H(s_t, a_t) = \begin{cases} E_t - \beta_1 \tilde{d} - c(s_{t,x}, a_t) & \text{if } \sum_{j=0}^{m} R_m(s_t) <= 1 \\ 1 & \text{otherwise.} \end{cases} \tag{33}$$

$\tilde{d}$ is the weighted complexity of the subspace chosen. $\tilde{c}(a)$ is the energy cost of taking an action and $\tilde{c}(s_{t,x}, a_t)$ is defined in equation (23). With the choice functional and system dynamics defined we implement the same iterative algorithm outlined in section 3.2. By design this scheme captures that core principles of energy-efficient decision making since at every time step the actor scans over a number of cognitively feasible decision subspaces, selects a subspace, evaluates candidate futures within its horizon, collapses down to a single decision, then iterates.

Algorithm 1 Model Predictive Control Algorithm

$s_{t,x} = x_0$, $t = 0$, $E_t = E_0$ $N =$ **Terminal Time**
while $t < N$ **do**
  $\tilde{u} = \arg\min_U J(U_t^{t+\Delta}, s_{t,x})$
  $\tilde{d} = \sum_{j=0}^{m} a_j(s_t) D(P_j)$
  $E_{t+1} = H(s_t, \tilde{u}(t))$
  $s_{t,x} = s_{t,x} + \tilde{u}(t)$
  $t = t + 1$
end while

We propose one small adjustment to the algorithm listed in section 3.2 and formalized above in order to imbue the agent with a sense of memory. Recall that MPC functions by making projections over a time horizon Δ, but only uses the first part of that projection to adjust its behavior before generating another projection. Instead we propose taking a weighted linear combination of the K past projections. For example if an agent is currently at time t, and is optimizing the function J to determine an action at time t, then it would weight the actions taken at time t in each of the last K future projection control sequence $U_{t-k}^{t+\Delta-k}(t)$. We could also reasonably adjust the weights γ of the linear combination to reflect the degree to which an agent prioritizes new future projections over older projections.

Algorithm 2 Model Predictive Control Algorithm With Memory

$s_{t,x} = x_0$, $t = 0$, $E_t = E_0$ $N =$ **Terminal Time**
while $t < N$ **do**
    $\tilde{u} = \sum_{k=0}^{K} \gamma^k \arg\min_U J(U_{t-k}^{t-k+\Delta}, s_{t-k,x})$
    $\tilde{d} = \sum_{j=0}^{m} a_j(s_t) D(P_j)$
    $E_{t+1} = H(s_t, \tilde{u}(t))$
    $s_{t,x} = s_{t,x} + \tilde{u}(t)$
    $t = t + 1$
end while

5 A Potential Application to Large Language Models (LLMs)

As reference in section 1, the applications of energy efficient agent-based decision making models like the MPC and RL models developed above have potential applications that span far beyond basic food/shelter foraging. In fact we propose that the models we developed in this proposal have direct applications to the development of more efficient LLM models.

For the purposes of applying our 2 models to improving LLM models, we must first re frame the LLM model as an agent in and of itself. This agent has an evolving state which is an expanding sequence of words/tokens t_i. Whereas the action space, of the agent is the next word in the sequence that the agent can predict.

$$s_n = \{t_1, t_2, t_3, ..., t_n\} \tag{34}$$

$$A = \{a - z, 0 - 9, \tag{35}$$

At every time step the the agent evaluates which token in the action space A maximizes a likelihood of that token coming next given a current sequence of tokens s_t, then chooses that token and adjusts the state. Under this framing, it becomes fairly clear why an RL or MPC model can be useful in this setting. In the case of the MPC and RL models, the LLM agent would be capable of predicting N tokens in advance before making a prediction. Furthermore the two models also allow the LLM agent to actively explore multiple sub spaces of potential token sequences in an energy-efficient manner before committing to a prediction. This approach may allow LLMs to avoid providing inaccurate information by blindly choosing the next most probable token, and replacing that approach with one that can generate text in the same way as a measured human speaker (i.e Thinking before you speak!).

References

1. Briola, Antonio, et al. Deep Reinforcement Learning for Active High Frequency Trading, 19 Aug. 2023, arxiv.org/abs/2101.07107.
2. Brunton, Steven Lee, and José Nathan Kutz. Data-Driven Science and Engineering: Machine Learning, Dynamical Systems and Control. Cambridge University Press, 2021.
3. Dai, Zhenwen, et al. In-Context Exploration-Exploitation for Reinforcement Learning, 11 Mar. 2024, arxiv.org/abs/2403.06826.
4. De Asis, Kristopher, et al. Multi-Step Reinforcement Learning: A Unifying Algorithm, 11 June 2018, arxiv.org/abs/1703.01327.
5. Hill, R. Carter, et al. Principles of Econometrics. Wiley, 2018.
6. Luo, Yunfei, and Zhangqi Duan. Agent Performing Autonomous Stock Trading under Good and Bad Situations, 6 June 2023, arxiv.org/abs/2306.03985.
7. Mnih, Volodymyr, et al. Playing Atari with Deep Reinforcement Learning, 19 Dec. 2013, arxiv.org/abs/1312.5602.
8. Pricope, Tidor-Vlad. Deep Reinforcement Learning in Quantitative Algorithmic Trading: A Review, 31 May 2021, arxiv.org/abs/2106.00123.
9. Ravichandiran, Sudharsan. Deep Reinforcement Learning with Python: Master Classic RL, Deep RL, Distributional RL, Inverse RL, and More with Openai Gym and Tensorflow. Packt, 2020.
10. Singh, Prabhsimran. "PSKRUNNER14/Trading-BOT: Stock Trading Bot Using Deep Q Learning." GitHub, 31 Dec. 2020, github.com/pskrunner14/trading-bot/.
11. Sutton, Richard S., and Andrew Barto. Reinforcement Learning: An Introduction. The MIT Press, 2020.
12. Engl, E. and Attwell, D. Non-signalling energy use in the brain. *J Physiol*, 593: 3417-3429, 2015.
13. McCubbins, Colin H. and McCubbins, Mathew D. and Turner, Mark B., Building a New Rationality from the New Cognitive Neuroscience, July 9, 2018, in Riccardo Viale and Konstantinos Katsikopoulos (eds.) *Handbook on Bounded Rationality*, Routledge Publishing House (Forthcoming), Duke Law School Public Law & Legal Theory Series No. 2018-52.
14. Sarishma, D., Sangwan, S., Tomar, R., Srivastava, R. A Review on Cognitive Computational Neuroscience: Overview, Models, and Applications. In: Tomar, R., Hina, M.D., Zitouni, R., Ramdane-Cherif, A. (eds) *Innovative Trends in Computational Intelligence*. EAI/Springer Innovations in Communication and Computing. Springer, Cham, 2022.
15. Slovic, Paul, Finucane, Melissa, Peters, Ellen, MacGregor, Donald G. Rational actors or rational fools: implications of the affect heuristic for behavioral economics, *The Journal of Socio-Economics*, Volume 31, Issue 4, 2002, Pages 329-342, ISSN 1053-5357.
16. National Research Council (US) Committee on Opportunities in Neuroscience for Future Army Applications. Opportunities in Neuroscience for Future Army Applications. Washington (DC): National Academies Press (US); 2009. 4, Optimizing Decision Making.
17. Saks, Michael J., and Kidd, Robert F. "Human information processing and adjudication: Trial by heuristics." *Law & Soc'y Rev*. 15 (1980): 123.
18. Sarkar, Soumodip, Osiyevskyy, Oleksiy. Organizational change and rigidity during crisis: A review of the paradox, *European Management Journal*, Volume 36, Issue 1, 2018, Pages 47-58.