Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εξάμηνο : 9ο
Μάθημα : Νευρο-ασαφής Έλεγχος και Εφαρμογές
Διδάσκων : Ι. Κορδώνης
Ακαδημαϊκό έτος : 2025-26

Εξαμηνιαία εργασία

Google Dreamer V1

Οναματεπώνυμο: Μωραΐτης Νικόλαος
Αριθμός Μητρώου: 031 21172

Περιεχόμενα

1 Περίληψη

Στόχος αυτής της εργασίας ήταν η κατανόηση και υλοποίηση του Google Dream V1. Τί είναι το google dreamer; Το google dreamer είναι ένας αλγόριθμος ενισχυτικής μάθησης που χρησιμοποιεί τον λανθάνον χώρο για εκπαίδευση. Αποτελείται από έναν Variational Auto Encoder (VAE) τον οποίο έχουμε σπάσει σε Encoder και Decoder και ένα RSSM (Recurrent State Space Model). Στόχος του είναι να πραγματοποιήσει έλεγχο σε ένα σύστημα όπως τετράποδο ρομπότ ανάποδο εκκρεμές και άλλα συστήματα. Εμείς θα εκπαιδεύσουμε το Dreamer στο παιχνίδι του gymnasium CarRacing-V3 όπου στόχος είναι το αυτοκίνητο να παραμείνει εντός ενός δρόμου για όσο το δυνατό περισσότερο. Εκπαιδεύουμε για όσο το δυνατόν περισσότερα βήματα, χωρίς gpu και τοπικά και στο colab. Για λόγους οικονομίας στην εκπαίδευση βάλαμε ασπρόμαυρες εικόνες αντί για έγχρωμες.

2 Εισαγωγή

2.1 Reinforcement Learning

Η ενισχυτική μάθηση (Reinforcement Learning, RL) αποτελεί μια περιοχή της τεχνητής νοημοσύνης που εστιάζει στην εκμάθηση συμπεριφορών μέσα από την αλληλεπίδραση ενός πράκτορα (agent) με ένα περιβάλλον. Ο πράκτορας επιλέγει ενέργειες έχοντας ως στόχο τη μεγιστοποίηση της συνολικής επιβράβευσης, χρησιμοποιώντας εμπειρία που συλλέγεται κατά τη διάρκεια της εκπαίδευσης. Παρά την αποτελεσματικότητά της σε απλά περιβάλλοντα, η ενισχυτική μάθηση συχνά δυσκολεύεται σε περιβάλλοντα υψηλής διάστασης, όπου οι παρατηρήσεις (π.χ. εικόνες) περιέχουν μεγάλο όγκο περιττής πληροφορίας.

Η ενισχυτική μάθηση μοντελοποιεί τη διαδικασία λήψης αποφάσεων ως ένα Markov Decision Process (MDP), ορίζοντας: State (s): Η τρέχουσα κατάσταση του περιβάλλοντος η οποία, στο δικό μας σύστημα περιλαμβάνει ένα ντετερμινιστικό κομμάτι h και ένα στοχαστικό κομματι z. Το ντετερμινιστικό κομμάτι εκφράζει την προβλέψιμη δυναμική του συστήματος. Το στοχαστικό κομμάτι αντιπροσωπεύει αβεβαιότητες και επιτρέπει την πρόβλεψη πιθανών μελλοντικών σεναρίων. Action (a): Η ενέργεια που εκτελεί ο πράκτορας, π.χ. επιτάχυνση ή στροφή του αυτοκινήτου. Reward (r): Η επιβράβευση που λαμβάνει ο πράκτορας για κάθε ενέργεια. Policy (π): Στρατηγική επιλογής ενεργειών βάσει της κατάστασης. Value function (V): Εκτιμά το συνολικό αναμενόμενο reward από μία κατάσταση. Στόχος του πράκτορα είναι η μεγιστοποίηση του συνολικού αναμενόμενου reward μέσω της εκπαίδευσης της policy του.

2.2 Google Dreamer

Ο αλγόριθμος του Google Dreamer V1 που προτάθηκε από τους Hafner et al. το 2020 αποτελεί μία σύγχρονη προσέγγιση ενισχυτικής μάθησης που προσπαθεί να αντιμετωπίσει αυτές τις προκλήσεις. Το Dreamer παίρνει έμπνευση από τα μοντέλα λανθάνοντος χώρου και τα συνδυάζει με actor-critic πολιτικές, επιτρέποντας στον agent να ονειρεύεται μελλοντικές καταστάσεις και να εκπαιδεύεται σε έναν λανθάνοντα χώρο αντί σε ακριβείς πραγματικές παρατηρήσεις. Αυτό επιτρέπει την ταχύτερη και λιγότερο κοστοβόρα εκπαίδευση αποφεύγοντας την επανεκκίνηση του χώρου προσομοίωσης σε κάθε βήμα. Το οποίο είναι ιδιαίτερα χρήσιμο σε περιβάλλοντα με υψηλές διαστάσεις όπως τα παιχνίδια με εικόνες από τις οποίες πρέπει να εξάγουμε

μόνο την χρήσιμη πληροφορία. Στην παρούσα εργασία υλοποιούμε από την αρχή και αναλύουμε το google dreamer V1 στο περιβάλλον CarRacing-V3 του gymnasium. Στόχος είναι η εκπαίδευση ενός πράκτορα που ελέγχει το αυτοκίνητο ώστε να παραμένει εντός του δρόμου για όσο το δυνατόν περισσότερο χρόνο.

2.3 Model-Based vs Model-Free RL

Σε αντίθεση με αλγορίθμους Model-Free (όπως PPO, DQN) που μαθαίνουν απευθείας από την αλληλεπίδραση, το Dreamer (Model-Based) προσφέρει:

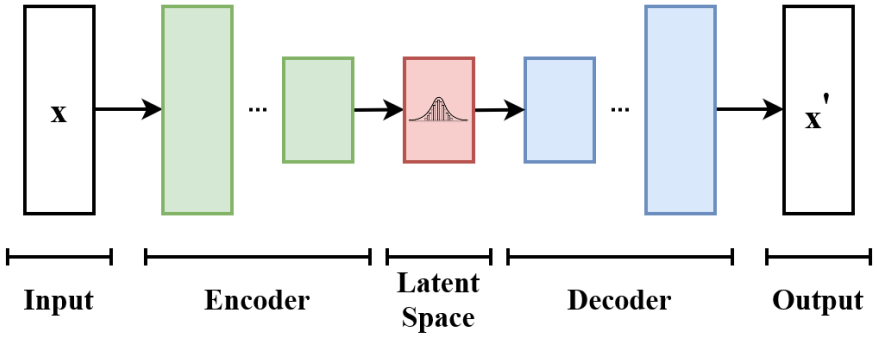
- **Sample Efficiency:** Χρειάζεται λιγότερα βήματα στο περιβάλλον, καθώς εκπαιδεύεται χιλιάδες φορές πάνω στα ίδια δεδομένα εντός της ”φαντασίας” του.
- **Long-horizon Prediction:** Μέσω του RSSM, μπορεί να προβλέψει τις συνέπειες ενεργειών πολύ πιο μακριά στο μέλλον από ό,τι ένας απλός Q-learning αλγόριθμος.

3 Components

Ας δούμε όλα τα components ένα ένα:

3.1 Variational Autoencoder (VAE)

Ο VAE αποτελεί ένα βασικό εργαλείο για την κωδικοποίηση εικόνων σε λανθάνοντα χώρο. Περιλαμβάνει έναν encoder και έναν decoder



Σχήμα 1: *Βασική δομή ενός VAE [5]*

3.1.1 Encoder: $p(s_t \mid s_{t-1}, a_{t-1}, o_t)$

Ο ρόλος του είναι να παίρνει μία εικόνα και να την μετατρέπει σε μοντέλο. Το να επεξεργαζόμαστε συνεχώς μία εικόνα δεν έχει νόημα καθώς εμπεριέχει πολλή περιττή πληροφορία και βαραίνει την εκπαίδευση. Αντίθετα αυτό που χρειαζόμαστε είναι να εξάγουμε την δυναμική του μοντέλου. Αντί να την εξάγουμε μέσω φυσικής την προσεγγίζουμε με την χρήση του encoder [4].

- Είσοδος: 64x64 (ασπρόμαυρη εικόνα)
- Έξοδος: Ένα διάνυσμα με μέγεθος 1024 bits

```
self.net = nn.Sequential(
    # (1, 64, 64) -> (32, 31, 31)
    nn.Conv2d(in_channels, 32, kernel_size=4, stride=2),
    nn.ReLU(),

    # (32, 31, 31) -> (64, 14, 14)
    nn.Conv2d(32, 64, kernel_size=4, stride=2),
```

```
    nn.ReLU(),

    # (64, 14, 14) -> (128, 6, 6)
    nn.Conv2d(64, 128, kernel_size=4, stride=2),
    nn.ReLU(),

    # (128, 6, 6) -> (256, 2, 2)
    nn.Conv2d(128, 256, kernel_size=4, stride=2),
    nn.ReLU(),

    # (256, 2, 2) -> (1024,)
    nn.Flatten()
)

# Output dimension
self.embed_dim = 256 * 2 * 2  # = 1024
```

3.1.2 Decoder: $p(\hat{o}_t \mid s_t)$

Ο ρόλος του είναι να λάβει ένα state [h, z] και να την μετατρέπει σε ασπρόμαυρη εικόνα. Μας είναι χρήσιμος καθώς μας επιτρέπει να απεικονίσουμε την φαντασία και να έχουμε μία καλύτερη εποπτεία με μια "γενικευμένη εικόνα" του δρόμου.

- Είσοδος: 230 αριθμοί
- Έξοδος: 64x64 (ασπρόμαυρη εικόνα)

```
def __init__(self, state_dim=230, out_channels=1):
    """
    Args:
        state_dim: Dimension of model state (h, z). Default: 200 + 30 = 230
        out_channels: Number of output image channels (1 for grayscale)
    """
    super().__init__()

    # Project state to spatial format
    self.fc = nn.Linear(state_dim, 1024)

    # Transposed convolutions to upsample
    self.net = nn.Sequential(
```

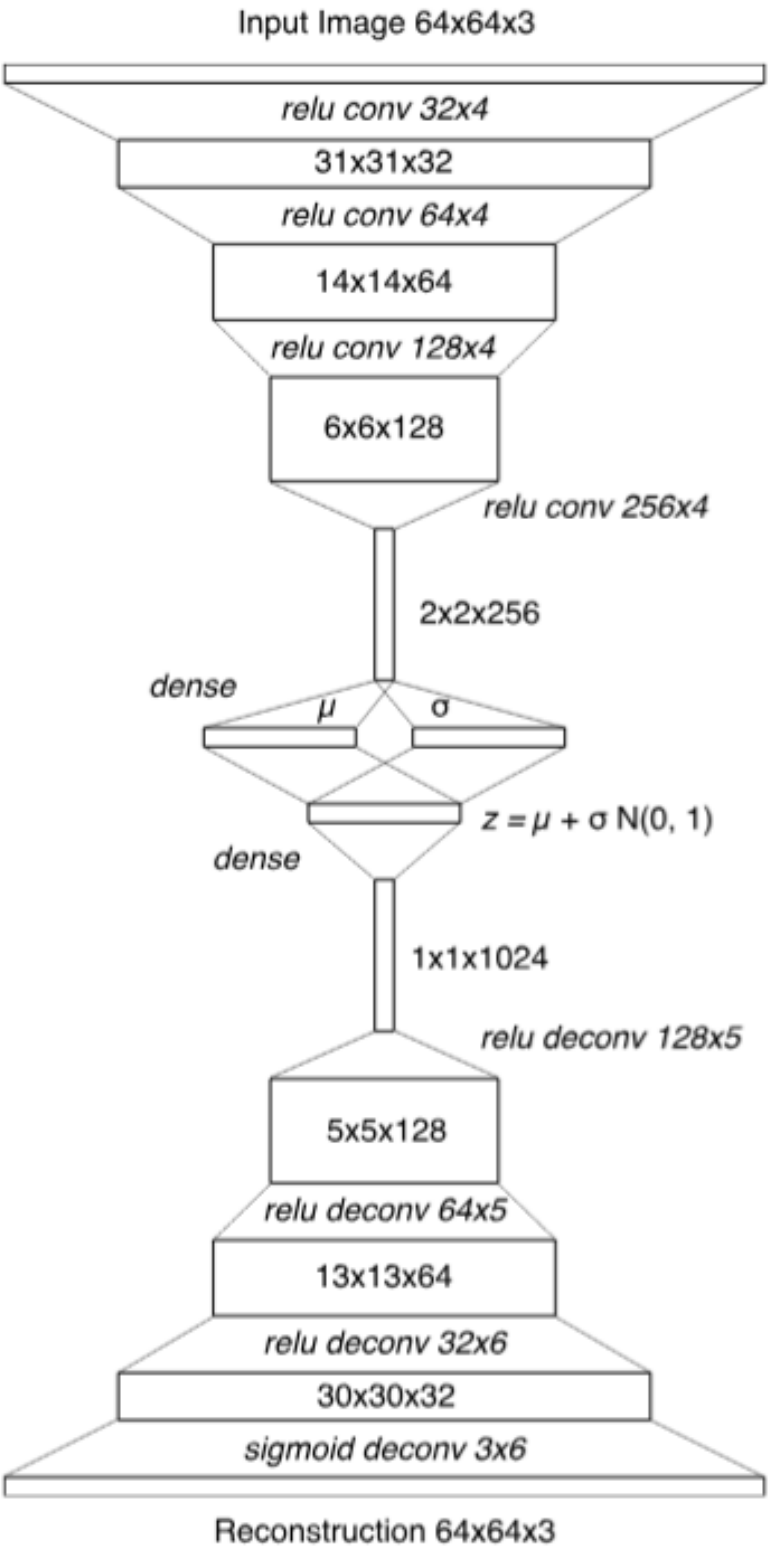
```
        # (1024,) -> (1024, 1, 1)
        nn.Unflatten(1, (1024, 1, 1)),

        # (1024, 1, 1) -> (128, 5, 5)
        nn.ConvTranspose2d(1024, 128, kernel_size=5, stride=2),
        nn.ReLU(),

        # (128, 5, 5) -> (64, 13, 13)
        nn.ConvTranspose2d(128, 64, kernel_size=5, stride=2),
        nn.ReLU(),

        # (64, 13, 13) -> (32, 30, 30)
        nn.ConvTranspose2d(64, 32, kernel_size=6, stride=2),
        nn.ReLU(),

        # (32, 30, 30) -> (1, 64, 64)
        nn.ConvTranspose2d(32, out_channels, kernel_size=6, stride=2),
        nn.Sigmoid()  # Output in [0, 1] range
    )
```



Σχήμα 2: *Η αναλυτική δομή του VAE που χρησιμοποιείται στον Dreamer [3]*

3.1.3 Επιλογή Ασπρόμαυρης Επεξεργασίας

Η επιλογή μετατροπής των εικόνων σε ασπρόμαυρες (grayscale) αντί για έγχρωμες (RGB) έγινε για τους εξής λόγους:

- **Μείωση υπολογιστικού κόστους:** Μία ασπρόμαυρη εικόνα 64×64 έχει 4,096 τιμές, ενώ μία έγχρωμη έχει 12,288 (3× περισσότερες). Αυτό μειώνει σημαντικά τον χρόνο εκπαίδευσης, ιδιαίτερα χωρίς GPU.
- **Επαρκής πληροφορία:** Στο περιβάλλον CarRacing-V3, η κρίσιμη πληροφορία (θέση δρόμου, όρια πίστας, θέση αυτοκινήτου) διατηρείται πλήρως στην ασπρόμαυρη αναπαράσταση. Το χρώμα δεν προσφέρει επιπλέον σημασιολογική πληροφορία για την εργασία.
- **Μικρότερο μοντέλο:** Ο encoder χρειάζεται λιγότερες παραμέτρους (1 κανάλι εισόδου αντί για 3), μειώνοντας την πιθανότητα overfitting με περιορισμένα δεδομένα εκπαίδευσης.
- **Συμβατότητα με World Models:** Η αρχιτεκτονική VAE που χρησιμοποιούμε βασίζεται στην εργασία των Ha & Schmidhuber [3], η οποία επίσης χρησιμοποιεί ασπρόμαυρες εικόνες.

3.2 Recurrent State Space Model (RSSM)

Αποτελείται από 2 μέρη. Μία ντετερμινιστική κατάσταση h και μία στοχαστική κατάσταση z που αντιπροσωπεύει την αβεβαιότητα της παρούσας κατάστασης. Ο συνδυασμός μας επιτρέπει να προβλέπουμε πιθανά σενάρια όχι μόνο τα σίγουρα.

- Η λειτουργία **Transition** του RSSM πραγματοποιεί την μετάβαση $q(s_t \mid s_{t-1}, a_{t-1})$. Πηγαίνει στην λανθάνουσα κατάσταση $h_t = \text{GRU}(h_{t-1}, \text{concat}(z_{t-1}, a_{t-1}))$ που είναι η πρόβλεψη για το μέλλον. Στη συνέχεια υπολογίζει την πιθανότητα της επόμενης κατάστασης με βάση την παρούσα δηλαδή την prior πιθανότητα $p(z_t \mid h_t)$. στον κώδικα το Transition υλοποιείται από την συνάρτηση imagine.
- Η λειτουργία **Observe** πραγματοποιεί τον υπολογισμό $p(s_t \mid s_{t-1}, a_{t-1}, o_t)$. Δηλαδή, μετά το όνειρο, εκτιμάται η posterior πιθανότητα $q(z_t \mid h_t, e_t)$ όπου e_t το σφάλμα μεταξύ z_t και h_t. Μέσω αυτής ενημερώνεται το μοντέλο.

Στον πυρήνα του RSSM βρίσκεται ένα multi-layer Gated Recurrent Unit (GRU) RNN.

3.3 Reward Model

Ο ρόλος του είναι να λάβει την παρούσα κατάσταση $[h, z]$ και να προβλέψει την επιβράβευση της κατάστασης δηλαδή υλοποιεί την $q(r_t \mid s_t)$. Αυτό πραγματοποιείται μέσω ενός MLP.

- **Είσοδος:** $[h, z]$ (len($[h, z]$) = 230 αριθμοί)
- **΄Εξοδος:** Ένας αριθμός

```
def __init__(self, state_dim, hidden_dim=300):
    super().__init__()
    self.net = nn.Sequential(
        nn.Linear(state_dim, hidden_dim),
        nn.ELU(),
        nn.Linear(hidden_dim, hidden_dim),
        nn.ELU(),
        nn.Linear(hidden_dim, 1)
    )
```

3.4 Actor

Ο ρόλος του είναι να αποφασίσει τι δράση να κάνει σε κάθε περίπτωση. Με βάση την αναμενόμενη τιμή του κέρδους αποφασίζει ποια είναι η καλύτερη επόμενη κίνηση. Ωστόσο προσθέτουμε και μία τυχαιότητα στις κινήσεις με στόχο την εξερεύνηση ενός μεγαλύτερου χώρου κινήσεων. Για να μετατρέψουμε την κίνηση σε actuation χρησιμοποιούμε την tanh και κάνουμε squash τις τιμές της επιτάχυνσης (γκάζι, φρένο) και στροφής στο χωρίο $[-1, 1]$.

```
def __init__(self, state_dim, action_dim, hidden_dim=300, min_std=0.1, init_std
    =5.0):
    super().__init__()
    self.min_std = min_std
    self.init_std = init_std

    self.net = nn.Sequential(
        nn.Linear(state_dim, hidden_dim),
        nn.ELU(),
        nn.Linear(hidden_dim, hidden_dim),
        nn.ELU(),
```

```
    )

    self.mean_head = nn.Linear(hidden_dim, action_dim)
    self.std_head = nn.Linear(hidden_dim, action_dim)
```

3.4.1 Reparameterization Trick

Μία κρίσιμη λεπτομέρεια για την εκπαίδευση του VAE και του Actor είναι η δυνατότητα υπολογισμού παραγώγων μέσω στοχαστικών κόμβων. Κανονικά, η δειγματοληψία $z \sim \mathcal{N}(\mu, \sigma)$ δεν είναι παραγωγίσιμη πράξη. Ο Dreamer χρησιμοποιεί το **Reparameterization Trick**:

$$z = \mu + \sigma \cdot \epsilon, \quad \text{όπου } \epsilon \sim \mathcal{N}(0, I) \tag{1}$$

Αυτό επιτρέπει στα gradients να περάσουν ανεμπόδιστα προς τα μ και σ, επιτρέποντας την εκπαίδευση του Encoder και του Actor μέσω backpropagation through time (BPTT).

3.5 Critic

Ο κριτής έχει ως στόχο την παραγωγή μίας εκτίμησης για το συνολικό κέρδος μέχρι τον ορίζοντα των 15 βημάτων.

```
def __init__(self, state_dim, hidden_dim=300):
    super().__init__()
    self.net = nn.Sequential(
        nn.Linear(state_dim, hidden_dim),
        nn.ELU(),
        nn.Linear(hidden_dim, hidden_dim),
        nn.ELU(),
        nn.Linear(hidden_dim, 1)
    )
```

3.6 Value Estimation (V_λ)

Για να εξισορροπηθεί η μεροληψία (bias) και η διακύμανση (variance), ο Dreamer χρησιμοποιεί τον εκτιμητή V_λ, ο οποίος είναι ένας εκθετικά σταθμισμένος μέσος όρος των εκτιμήσεων για διαφορετικούς ορίζοντες k.

$$V_{\mathrm{R}}(s_\tau) \doteq \mathrm{E}_{q_\theta, q_\phi}\left(\sum_{n=\tau}^{t+H} r_n\right), \tag{4}$$

$$V_{\mathrm{N}}^{k}(s_\tau) \doteq \mathrm{E}_{q_\theta, q_\phi}\left(\sum_{n=\tau}^{h-1} \gamma^{n-\tau} r_n + \gamma^{h-\tau} v_\psi(s_h)\right) \quad \text{with} \quad h = \min(\tau + k, t + H), \tag{5}$$

$$V_\lambda(s_\tau) \doteq (1-\lambda) \sum_{n=1}^{H-1} \lambda^{n-1} V_{\mathrm{N}}^{n}(s_\tau) + \lambda^{H-1} V_{\mathrm{N}}^{H}(s_\tau), \tag{6}$$

4 Αλγόριθμος

Η διαδικασία που ακολουθεί το Dreamer κατά την διάρκεια της εκπαίδευσης είναι η εξής:

Algorithm 1: Dreamer

Initialize dataset $\mathcal{D}$ with S random seed episodes.
Initialize neural network parameters θ, ϕ, ψ randomly.
while *not converged* **do**
  for *update step* $c = 1..C$ **do**
    `// Dynamics learning`
    Draw B data sequences $\{(a_t, o_t, r_t)\}_{t=k}^{k+L} \sim \mathcal{D}$.
    Compute model states $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$.
    Update θ using representation learning.
    `// Behavior learning`
    Imagine trajectories $\{(s_\tau, a_\tau)\}_{\tau=t}^{t+H}$ from each s_t.
    Predict rewards $\mathrm{E}\big(q_\theta(r_\tau \mid s_\tau)\big)$ and values $v_\psi(s_\tau)$.
    Compute value estimates $\mathrm{V}_\lambda(s_\tau)$ via Equation 6.
    Update $\phi \leftarrow \phi + \alpha \nabla_\phi \sum_{\tau=t}^{t+H} \mathrm{V}_\lambda(s_\tau)$.
    Update $\psi \leftarrow \psi - \alpha \nabla_\psi \sum_{\tau=t}^{t+H} \frac{1}{2} \big\| v_\psi(s_\tau) - \mathrm{V}_\lambda(s_\tau) \big\|^2$.
  `// Environment interaction`
  $o_1 \leftarrow$ `env.reset()`
  for *time step* $t = 1..T$ **do**
    Compute $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$ from history.
    Compute $a_t \sim q_\phi(a_t \mid s_t)$ with the action model.
    Add exploration noise to action.
    $r_t, o_{t+1} \leftarrow$ `env.step(`a_t`)`.
  Add experience to dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^{T}\}$.

Model components

Representation	$p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$
Transition	$q_\theta(s_t \mid s_{t-1}, a_{t-1})$
Reward	$q_\theta(r_t \mid s_t)$
Action	$q_\phi(a_t \mid s_t)$
Value	$v_\psi(s_t)$

Hyper parameters

Seed episodes	S
Collect interval	C
Batch size	B
Sequence length	L
Imagination horizon	H
Learning rate	α

Ποιο αναλυτικά και διαισθητικά, η παραπάνω διαδικασία περιγράφεται παρακάτω:

4.1 Αρχικοποίηση

Αρχικοποιούμε το βέλτιστο μέσο reward σε $-\infty$, τις παραμέτρους του μοντέλου (θ) και του κριτή (ψ) τυχαία και παράγουμε $S = 5$ επεισόδια. Επιπλέον θέτουμε τις υπερπαραμέτρους ως εξής:

- Until done = 500.000 βήματα
- $C = 100, B = 50, L = 50, H = 15, T = 50$
- **Learning rates:**
  - World model (encoder, RSSM, decoder, reward): 6×10^{-4}
  - Value model (critic): 8×10^{-5}
  - Action model (actor): 8×10^{-5}

4.2 Φάση 1: Συλλογή δεδομένων

1. **Προεπεξεργασία:** Μετατροπή του frame o_t σε ασπρόμαυρο, κανονικοποίηση των τιμών.
2. Ο encoder μετατρέπει το o_t σε εκτίμηση της κατάστασης του συστήματος $[h, z]$.
3. Το RSSM παρατηρεί την τωρινή κατάσταση και εκτιμά τις prior και posterior της προηγούμενης κατάστασης.
4. Ο Actor επιλέγει μία κίνηση (action) με βάση την τωρινή κατάσταση.
5. Προσθέτουμε θόρυβο στο action για να επιτρέψουμε την εξερεύνηση νέων πολιτικών.
6. Πραγματοποιούμε την κίνηση στον πραγματικό κόσμο και παίρνουμε την επιβράβευση μας καθώς και το επόμενο frame o_{t+1}.
7. Αποθηκεύουμε την νέα κίνηση στον Replay Buffer.
8. Επαναλαμβάνουμε τα βήματα 1-8 μέχρι τις C επαναλήψεις.

4.3 Φάση 2: Representation Learning

1. Παίρνουμε ένα σύνολο κινήσεων από τον replay buffer (batch).
2. Αρχικοποιούμε τις T καταστάσεις του batch.
3. Πραγματοποιούμε ένα βήμα εκπαίδευσης T φορές:
   (α') Παίρνουμε το frame και το κωδικοποιούμε ως e_t.
   (β') Το RSSM γνωρίζοντας την προηγούμενη κατάσταση, το προηγούμενο action και το τωρινό encoding παράγει την prior, posterior και το $[h, z] = s_t$.
   (γ') Πραγματοποιούμε decode του παρόντος state s_t και βρίσκουμε το σφάλμα του Decoder με βάση την πραγματική εικόνα o_t. Το σφάλμα υπολογίζεται με MSE.
   (δ') Προβλέπουμε το reward του s_t που προβλέψαμε $\hat{r}_t$ και το συγκρίνουμε με το αποθηκευμένο στον replay buffer reward r_t. Βρίσκουμε MSE.
   (ε') Υπολογίζουμε $KL(\text{prior} \parallel \text{posterior})$ όπου KL = Kullback–Leibler divergence.
4. Με βάση τα παραπάνω υπολογίζουμε το συνολικό σφάλμα ως εξής:

$$\text{total\_loss} = \text{reconstruction\_loss} + \text{reward\_loss} + \beta \times \text{kl\_loss}$$

5. Πραγματοποιούμε back propagation και στη συνέχεια ενημερώνουμε τις παραμέτρους

του μοντέλου θ (world_params) χρησιμοποιώντας Adam optimizer. Έτσι τελειώνει το μέρος της εκπαίδευσης του μοντέλου.

4.4 Φάση 3: Χώρος Λανθάνουσας Φαντασίας - Εκπαίδευση Συμπεριφοράς

Στη συνέχεια μπαίνουμε στον λανθάνοντα χώρο. Παίρνουμε μία κίνηση από το ιστορικό (η οποία είχε παραχθεί από τον Actor) και το RSSM προσπαθεί να «ονειρευτεί» την επόμενη κατάσταση όπως στο 11β. Επαναλαμβάνουμε τα παρακάτω T φορές:

1. Όνειρο και εύρεση $[h_\tau, z_\tau]$ (τ συμβολίζει τον λανθάνον χρόνο).
2. Το Reward model προβλέπει το reward της επόμενης κατάστασης.
3. Αποθήκευση κατάστασης και reward r_τ.
4. Ο Critic προβλέπει το reward μέχρι το τέλος του ορίζοντα (H επαναλήψεις) για κάθε ένα από τα T states που φανταστήκαμε (έχουν παραχθεί από τον Actor).
5. Υπολογίζουμε το V_λ χρησιμοποιώντας τα r_τ και $v_\psi(s_\tau)$.
6. Υπολογίζουμε τα $v_\psi(s_\tau)$ ξανά με gradients.
7. Πραγματοποιούμε back propagation στο actor_loss $= -V_\lambda$.mean.
8. Πραγματοποιούμε Adam optimization στον actor ενημερώνοντας την πολιτική του.
9. Κάνουμε detach στα imagined states, δηλαδή για $T \times H$ καταστάσεις παγώνοντας τις επιλογές που έγιναν αφού δεν θέλουμε να τις αλλάξουμε.
10. Συγκρίνουμε με τις τιμές v_ψ με τα V_λ που βρήκαμε στο 16 και βρίσκουμε το MSE.
11. Πραγματοποιούμε backpropagation στο σφάλμα MSE του κριτή.
12. Ενημερώνουμε τον τρόπο εκτίμησης επιβράβευσης του Critic χρησιμοποιώντας Adam optimizer.
13. Επαναλαμβάνουμε τις 3 παραπάνω φάσεις για κάθε βήμα εκπαίδευσης με σύνολο 23.000 βήματα (Η google έκανε 5.000.000).

Σημασία του KL Divergence: Ο όρος L_{KL} στη συνάρτηση κόστους εξυπηρετεί δύο σκοπούς:

1. Λειτουργεί ως regularization, αποτρέποντας τον λανθάνοντα χώρο από το να γίνει ασυνεχής (κάτι που θα δυσκόλευε το "όνειρο").
2. Εξασφαλίζει ότι η posterior κατανομή $q(z_t|\cdot)$ παραμένει κοντά στην prior $p(z_t|\cdot)$, επιτρέποντας στο μοντέλο να γενικεύει σε νέες καταστάσεις.

Ορισμός KL Divergence: Η απόκλιση Kullback-Leibler (D_{KL}) είναι ένα μέτρο της διαφοράς μεταξύ δύο κατανομών πιθανότητας P και Q. Για συνεχείς τυχαίες μεταβλητές, ορίζεται ως το ολοκλήρωμα:

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx \tag{2}$$

Ουσιαστικά, εκφράζει την "πληροφοριακή απόσταση" που χάνεται αν χρησιμοποιήσουμε την κατανομή Q για να προσεγγίσουμε την πραγματική κατανομή P.

KL Divergence για Κανονικές Κατανομές: Στον αλγόριθμο Dreamer, όπου οι κατανομές posterior $q(z|x)$ και prior $p(z)$ είναι πολυμεταβλητές Γκαουσιανές (Multivariate Gaussians), η KL Divergence υπολογίζεται αναλυτικά ως:

$$D_{KL}(q \parallel p) = \frac{1}{2} \left(\mathsf{tr}(\Sigma_p^{-1}\Sigma_q) + (\mu_p - \mu_q)^\top \Sigma_p^{-1}(\mu_p - \mu_q) - k + \ln \left(\frac{\det \Sigma_p}{\det \Sigma_q} \right) \right) \tag{3}$$

όπου k είναι η διάσταση του λανθάνοντος χώρου, μ οι μέσες τιμές και Σ οι πίνακες διακύμανσης.

5 Αποτελέσματα

5.1 Standard Ρύθμιση

- **Device:** cpu
- **Loaded model from:** 'dreamer_best.pth'

Η εκπαίδευση πραγματοποιήθηκε με την ακόλουθη διαμόρφωση:

Παράμετρος	**Τιμή**
Περιβάλλον	
Environment	CarRacing-V3
Observation	64×64 grayscale
Action space	[steering, gas, brake] $\in [-1, 1]^3$
Action repeat	2
Αρχιτεκτονική	
Stochastic dim (z)	30
Deterministic dim (h)	200
Hidden dim	300
Embedding dim	1024
Εκπαίδευση	
Total steps	500,000 (target) / 23,500 (actual)
Seed episodes	5
Batch size	50
Sequence length	50
Imagination horizon	15
Hyperparameters	
World model LR	6×10^{-4}
Actor LR	8×10^{-5}
Critic LR	8×10^{-5}
Discount (γ)	0.99
Lambda (λ)	0.95
Free nats	3.0
Gradient clip	100
Hardware	
Device	CPU
Training time	$\sim$ 48 hours

Πίνακας 1: *Παράμετροι εκπαίδευσης*

5.2 Σύνοψη Εκπαίδευσης

Το μονέλο εκπαιδεύτηκε συνολικά για 47 επεισόδια, με βήματα που κυμαίνονται από 500 έως 23,500. Τα στατιστικά εκπαίδευσης δείχνουν σημαντική μείωση στο σφάλμα ανακατασκευής του μοντέλου, αν και το σήμα επιβράβευσης παρουσιάζει

υψηλή διακύμανση, κάτι που είναι τυπικό για την αρχική φάση της ενισχυτικής μάθησης σε σύνθετα περιβάλλοντα όπως το `CarRacing-V3`.

5.3 Σύγκριση Επιδόσεων (Performance Comparison)

Metric	Initial Model	Optimized Model (Final)
Mean Reward	-42.07	**670.21**
Std Deviation	8.17	165.03
Min Reward	-53.18	372.13
Max Reward	-27.97	837.04

Πίνακας 2: *Σύγκριση επιδόσεων μεταξύ αρχικού και τελικού μοντέλου.*

Αναλυτικά τα 5 επεισόδια της τελικής αξιολόγησης:

Episode	Steps	Reward
1	1000	837.04
2	1000	654.97
3	1000	678.17
4	1000	372.13
5	1000	808.75

Πίνακας 3: *Αναλυτικά αποτελέσματα του βελτιστοποιημένου μοντέλου.*

Το υψηλό μέσο reward (670.21) υποδεικνύει ότι ο πράκτορας έμαθε επιτυχώς να πλοηγείται στην πίστα, παραμένοντας εντός δρόμου και διατηρώντας ικανοποιητική ταχύτητα.

5.4 Οπτική Παρατήρηση

Παρακάτω δείχνουμε κάποιες εικόνες από το περιβάλλον προσομοίωσης με μία γρήγορη εκπαίδευση 5000 βημάτων. Η πρώτη μας δείχνει ότι το αυτοκίνητο παρόλο που ο δρόμος είναι ευθεία πάει να στρίψει. Στην δεύτερη έχει βγει εκτός πορείας.

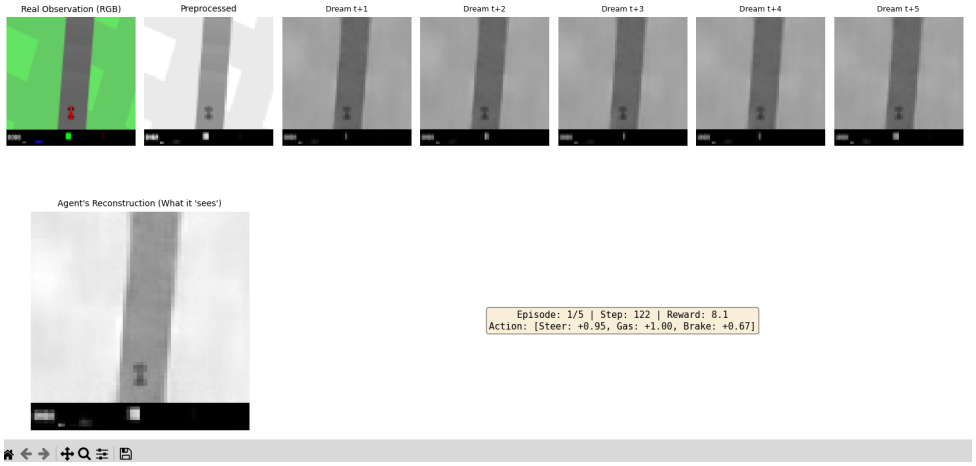
(α') *Αυτοκίνητο που στρίβει σε ευθεία*

(β') *Αυτοκίνητο εκτός πορείας*

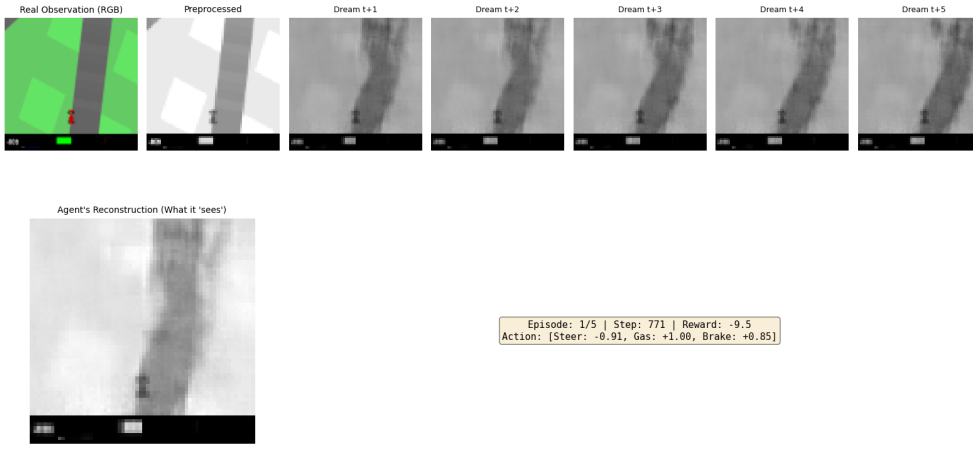
Σχήμα 3: *Εικόνες από την προσομοίωση*

Παρατηρούμε λοιπόν ότι το μοντέλο δεν έχει μάθει ακόμα να κρατιέται στην πορεία, δεν καταλαβαίνει καν τι χρειάζεται να κάνει. Αυτό είναι λογικό και αναμενόμενο αφού η μέθοδος του Dreamer απαιτεί πολλές ώρες εκπαίδευσης και μεγάλο αριθμό βημάτων δοκιμής και αποτυχίας μέχρι να φτάσει σε ικανοποιητικά αποτελέσματα.

Παρακάτω βλέπουμε πώς απεικονίζονται τα όνειρα κατά την διάρκεια του testing.



(α') *Αυτοκίνητο που κινείται κανονικά σε ευθεία, 5 όνειρα και ανακατασκευή*

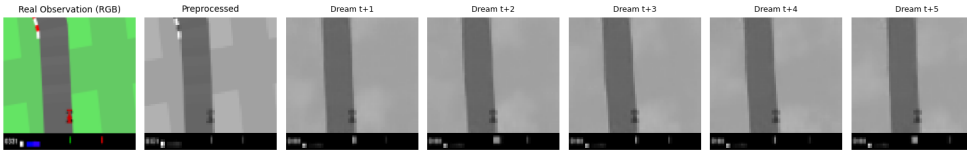


(β') *Αυτοκίνητο οριακά εκτός πορείας σε ευθεία, 5 όνειρα και ανακατασκευή*

Σχήμα 4: *Εικόνες από το testing με όνειρα και ανακατασκευές*

Στη συνέχεια πραγματοποιήσαμε εκπαίδευση για 23500 βήματα και παρακάτω παρουσιάζουμε τα όνειρα (dreams) και τις ανακατασκευές (reconstructions) που παράγει το μοντέλο.

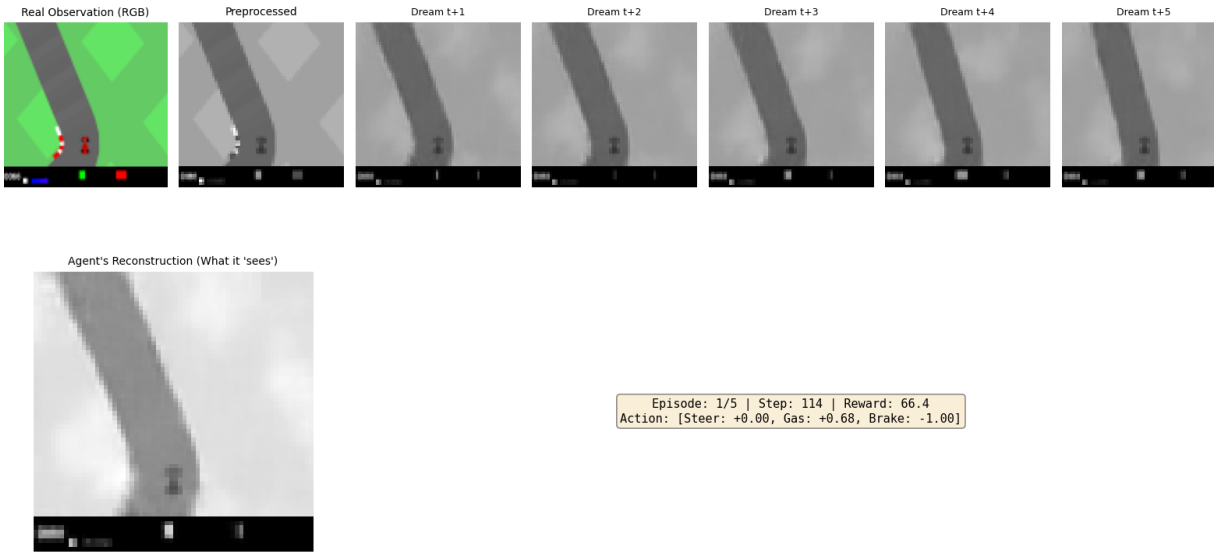
(α') *Αυτοκίνητο εντός πορείας σε στροφή και 5 όνειρα*



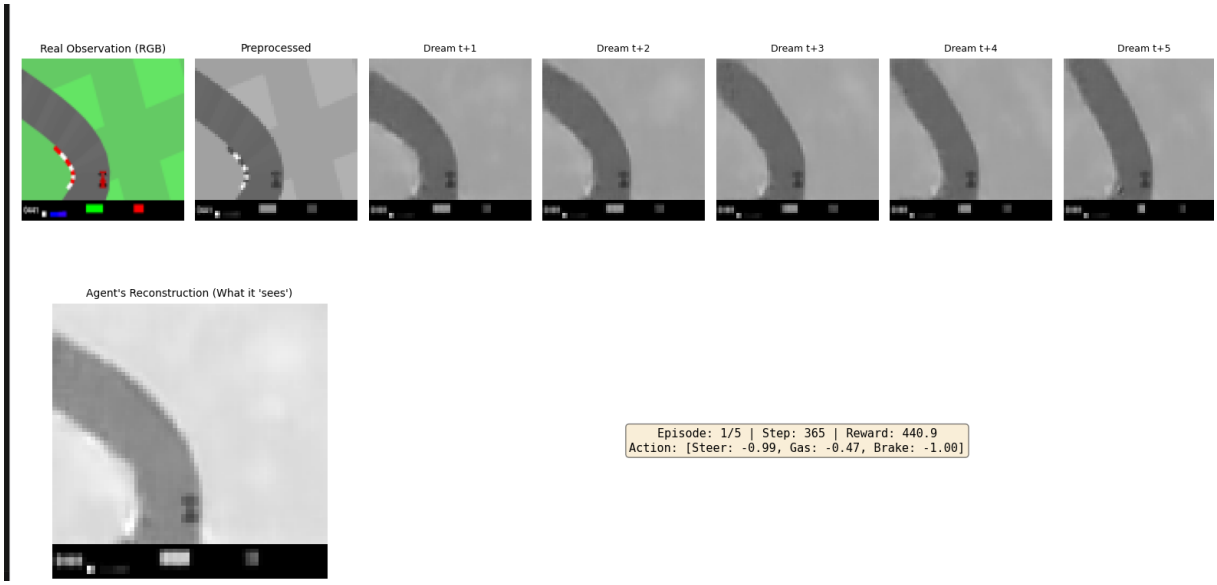
(β') *Αυτοκίνητο οριακά εντός πορείας σε ευθεία και 5 όνειρα*

Σχήμα 5: *Εικόνες από το testing με όνειρα*

Παρατηρούμε ότι με 23000 βήματα εκπαίδευσης τα αποτελέσματα είναι πολύ καλύτερα αλλά κάποιες φορές πάει να βγει εκτός πορείας. Τέλος, παρακάτω παρουσιάζουμε όνειρα και ανακατασκευές από ένα καλύτερο run.



Σχήμα 6: *Επιτυχημένη οδήγηση με όνειρα και ανακατασκευή*



Σχήμα 7: *Επιτυχημένη οδήγηση με όνειρα και ανακατασκευή (απότομη στροφή)*

Παρατηρούμε ότι τα όνειρα διατηρούν βασικά χαρακτηριστικά της πραγματικής εικόνας, όπως η θέση του δρόμου και του αυτοκινήτου, αλλά ταυτόχρονα παρουσιάζουν παραμορφώσεις και απώλειες λεπτομερειών, όσο εξελισσεται ο δρόμος και ειδικά προς τις στροφές. Αυτή είναι η δουλειά του ονείρου να προβλέπει την μελλοντική κατάσταση βασισμένο στην τρέχουσα, χωρίς να έχει απόλυτη βεβαιότητα/ακριβεια.

Οι ανακατασκευές είναι αρκετά πιστές στις αρχικές εικόνες, υποδεικνύοντας ότι το VAE κατάφερε να μάθει μια αποτελεσματική αναπαράσταση του περιβάλλοντος τόσο για κωδικοποίηση όσο και για ανακατασκευή.

Ακόμα και σε κλειστές στροφές βρίσκει τον τρόπο να παραμείνει στην πορεία, δείχνοντας ότι ο πράκτορας έχει μάθει μια ικανοποιητική πολιτική οδήγησης.

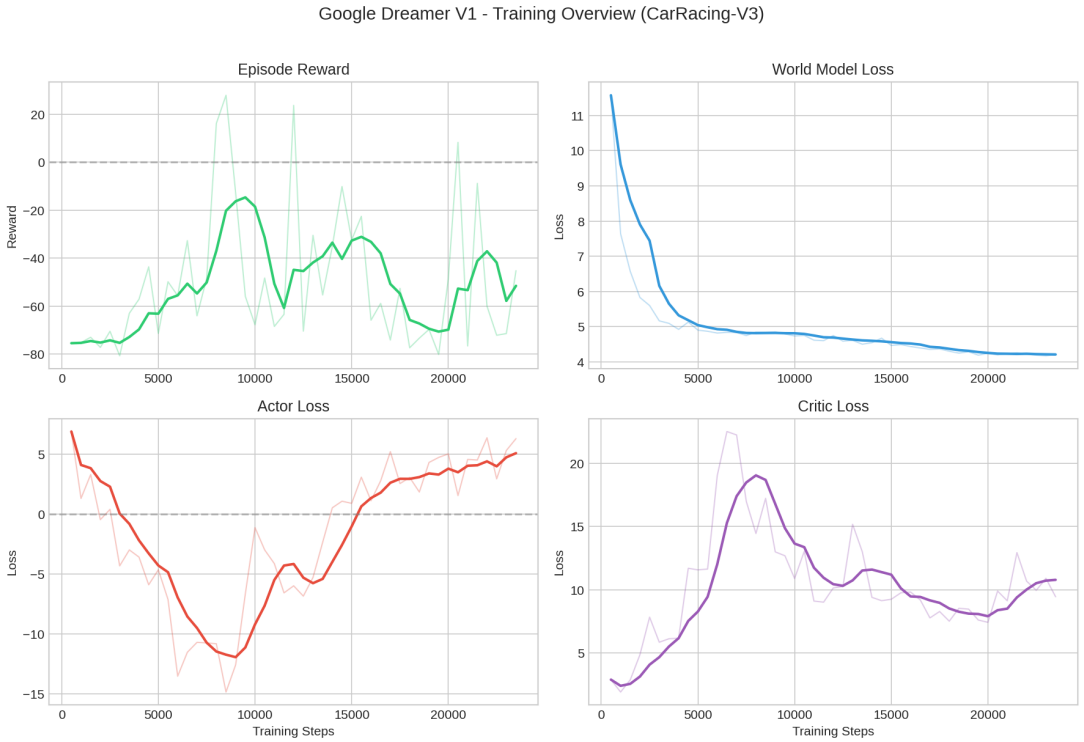
6 Συμπεράσματα

Στην παρούσα εργασία υλοποιήσαμε τον αλγόριθμο Google Dreamer V1 από την αρχή και τον εφαρμόσαμε στο περιβάλλον `CarRacing-V3` του Gymnasium. Μέσω της ανάλυσης της εκπαιδευτικής διαδικασίας και των αποτελεσμάτων, καταλήγουμε στα ακόλουθα συμπεράσματα.

6.1 Επιτεύγματα

Η υλοποίηση ανέδειξε τη δύναμη των μοντέλων λανθάνοντος χώρου:

- **Εκμάθηση Δυναμικής:** Το world model κατάφερε επιτυχώς να μάθει τη δυναμική του περιβάλλοντος, γεγονός που επιβεβαιώνεται από τη σημαντική μείωση του reconstruction loss.
- **Latent Imagination:** Η αρχιτεκτονική RSSM επέτρεψε την εκπαίδευση του πράκτορα «στη φαντασία» (λανθάνοντα χώρο), μειώνοντας δραστικά την ανάγκη για συνεχή αλληλεπίδραση με το πραγματικό περιβάλλον.
- **Αποδοτικότητα Πόρων:** Η στρατηγική χρήση ασπρόμαυρων εικόνων (grayscale) κατέστησε εφικτή την εκπαίδευση του μοντέλου σε λογικό χρόνο, ακόμα και χωρίς τη χρήση GPU.



Σχήμα 8: *Συνολική επισκόπηση της πορείας εκπαίδευσης (Training Overview).*

Αναφορές

[1] Hafner, D., Lillicrap, T., Ba, J., & Norouzi, M. (2020). *Dream to Control: Learning Behaviors by Latent Imagination*. International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1912.01603

[2] Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., & Davidson, J. (2019). *Learning Latent Dynamics for Planning from Pixels*. International Conference on Machine Learning (ICML). https://arxiv.org/abs/1811.04551

[3] Ha, D., & Schmidhuber, J. (2018). *World Models*. https://arxiv.org/abs/1803.10122

[4] Kingma, D. P., & Welling, M. (2014). *Auto-Encoding Variational Bayes*. International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1312.6114

[5] Wikipedia contributors. (2024). *Variational autoencoder*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Variational_autoencoder

[6] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. EMNLP. https://arxiv.org/abs/1406.1078

[7] Farama Foundation. (2023). *Gymnasium: A Standard API for Reinforcement Learning*. https://gymnasium.farama.org/