

# steer

---

## Capabilitățile Arhitecturii Mamba în Procesarea Secvențelor: Studiu de Caz în Domeniul Vehiculelor Autonome

---

*Elev, clasa a 11-a*  
Asandei STEFAN-ALEXANDRU

*Profesor coordonator*  
Pădurariu EMANUELA-TATIANA

Concursului National de Proiecte de Stiinta, Tehnologie si Inginerie  
2 Noiembrie 2024, Suceava



## Rezumat

În ultimii ani, mecanismul de atenție a devenit metoda principală pentru dezvoltarea modelelor avansate de inteligență artificială. Modelele bazate pe arhitectura Transformer s-au dovedit extrem de performante în înțelegerea limbajului, însă prezintă o complexitate pătratică pe măsură ce se scalează. O nouă arhitectură, Mamba, bazată pe modele în spațiul stărilor, încearcă să rezolve aceste probleme, dar încă nu beneficiază de suficiente teste experimentale. În această lucrare, propunem un model bazat pe Mamba, care s-a dovedit foarte eficient în sarcina de a înțelege secvențe video în contextul vehiculelor autonome. Reteaua neuronală prezice viteza vehiculului, unghiul volanului și traectoria viitoare, trei caracteristici corelate prin care modelul caută să înțeleagă comportamentul acestuia. În plus, am antrenat două modele clasice pe același set de date, pentru a realiza benchmark-uri. Demonstrăm că modelul propus poate depăși modelele tradiționale, fiind totodată mai rapid în inferență. De asemenea, am publicat codul pentru antrenare și inferență, precum și datele modelului pre-antrenat<sup>1</sup>, ca resurse educaționale pentru dezvoltatorii interesați de arhitectura Mamba.

---

<sup>1</sup>Codul pentru inferenta si antrenament este disponibil pe <https://github.com/stefanasandei/steer>



# Cuprins

<b>1</b>	<b>Introducere</b>	<b>1</b>
<b>2</b>	<b>Context</b>	<b>3</b>
2.1	Metode clasice . . . . .	3
2.2	Modele in Spatiul Starilor . . . . .	4
2.3	Modele Selective in Spatiul Starilor . . . . .	4
<b>3</b>	<b>Lucrari relevante</b>	<b>5</b>
<b>4</b>	<b>Rezultate</b>	<b>6</b>
4.1	Benchmarks . . . . .	6
4.2	Observatii . . . . .	7
<b>5</b>	<b>Procesarea datelor</b>	<b>9</b>
5.1	Dataset . . . . .	9
5.2	Preprocesare . . . . .	9
5.3	Augmentarea datelor si curatare . . . . .	10
5.4	Vizualizare . . . . .	10
<b>6</b>	<b>Arhitectura modelului</b>	<b>11</b>
<b>7</b>	<b>Detalii privind antrenarea</b>	<b>13</b>
7.1	Rata de invatare optima . . . . .	13
7.2	Procesul de antrenare . . . . .	13
<b>8</b>	<b>Concluzie</b>	<b>15</b>
8.1	Modalitati de imbunatatire . . . . .	15

# 1. Introducere

Sarcina de procesare a datelor secvențiale a apărut ca o capacitate crucială pentru modelele de învățare profundă. Cazurile de utilizare ale acestui lucru conduc la modele de înțelegere a limbajului, a serii cronologice, a imaginilor și a videoclipurilor. Probabil, unul dintre cele mai cunoscute modele generative este GPT-4 de OpenAI, folosit în ChatGPT. Utilizează mecanismul de auto-attenție [1] pentru a modela limbajul natural și pentru a prezice simboluri viitoare. Un alt progres în acest domeniu este reprezentat de Vision Transformer [2], care depășește CNN-urile, după ce a fost instruit pe cantități mai mari de date. Contribuțiile tuturor acestor rezultate nu pot fi puse la îndoială; cu toate acestea, există o problemă de eficiență care încetinește Transformers. Mecanismul de auto-attenție necesită complexitate pătratică pe măsură ce fereastra contextului crește. Din acest motiv, Transformers necesită hardware relativ puternic pentru antrenament și inferență, ceea ce face mai dificilă implementarea și utilizarea unor astfel de modele. Cu puține modificări sau deloc în arhitectura modelului primar, doar îmbunătățiri ale benchmark-urilor sunt obținute prin creșterea numărului de parametri (în intervalul de miliarde: Llama 3.1 405B) și prin utilizarea seturilor de date mai bune. Modelele eficiente trebuie să fie implementate pe hardware cu resurse reduse, cum ar fi computere personale, smartphone-uri și dispozitive încorporate.

Recent, a existat o încercare de a contesta arhitectura Transformerului prin utilizarea modelelor spațiale de stat. Aceste rețele au complexitate liniară, putând fi calculate folosind convoluții sau recurențe. Mamba [3] folosește spații de stări selective pentru a propaga sau a uita în mod selectiv informațiile din fereastra de context. Dispune de un design conștient de hardware, care permite trecerea rapidă a modelului, de până la 5 ori mai rapid decât un Transformer, având în același timp o utilizare mai mică a memoriei. Au existat studii care extind capacitatea Mamba la sarcini de imagine [4] și video [5]. În acest proiect, vă prezint un model bazat pe Mamba, urmând arhitectura VideoMamba.

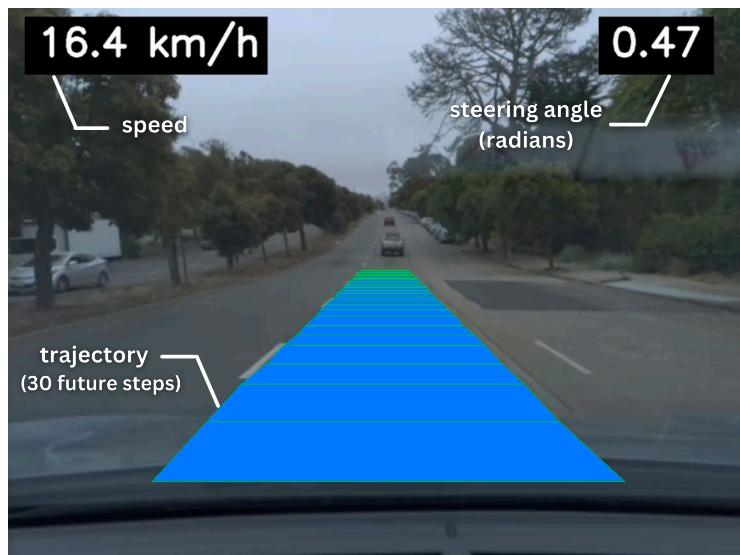


Figure 1.1. Exemplu de predictie pentru un cadru.

Pentru ca o arhitectură model să fie eficientă, necesită sprijin experimental, deoarece rețelele neuronale trebuie să fie observate pe date și scenarii din lumea reală. Mamba, fiind o arhitectură relativ nouă, are puțină utilizare în lumea reală, în special modelul de limbaj Mamba 3B din lucrarea originală și modelul Codestral-Mamba de la Mistral. Mi-am antrenat modelul, **Steer**, pe un set de date din lumea reală pentru conducerea autonomă. Procesează 10 cadre din trecut pentru a prezice traectoria viitoare a vehiculului, împreună cu unghiul și viteza volanului. Un astfel de model ar fi cel mai eficient, într-un scenariu real, implementat pe un sistem încorporat în interiorul unei mașini pentru inferență locală. Pentru aceasta, un model Transformer ar fi sub-optim, făcând rezultatele și mai pline de speranță.

Steer este o rețea neuronală end-to-end cu arhitectura VideoMamba la bază. În figuri 1.1 și 1.2, poate fi văzut un exemplu de rezultat al modelului. Trecerea înainte ia într-un context temporal al ultimelor 10 cadre, împreună cu datele de traectorie, pentru a prezice valorile. Figura este o vizualizare generată folosind un script din depozitul de proiect.

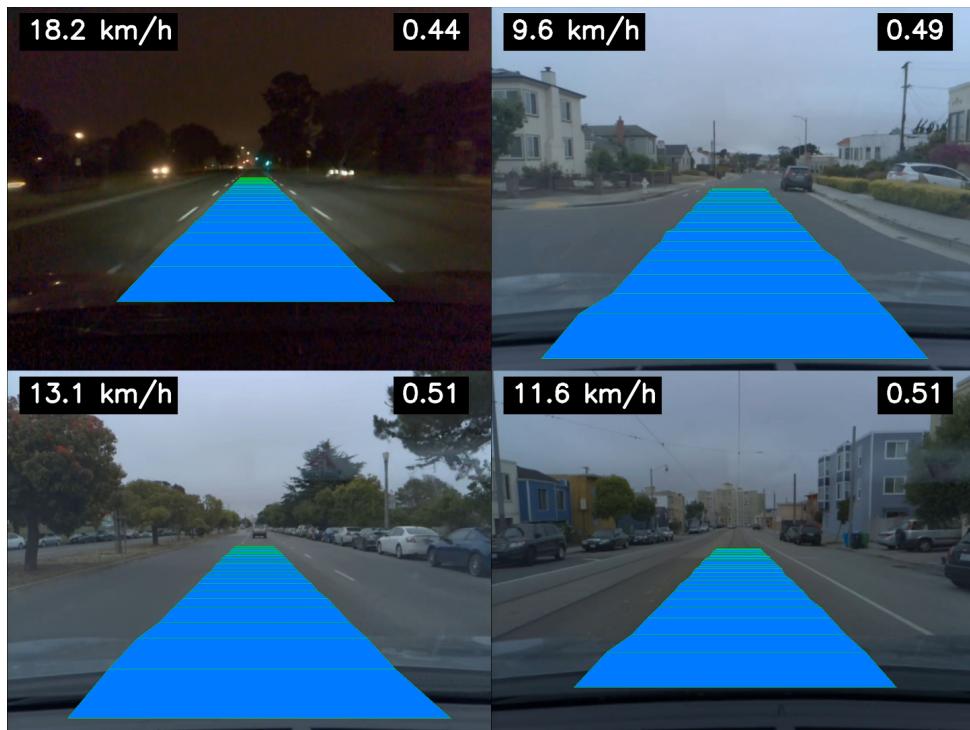


Figure 1.2. Predictii ale modelului în situații diferite de mediu și lumina.

Această sarcină a fost concepută în mod intenționat pentru a prelua mai multe intrări și a genera mai multe ieșiri, deoarece provoacă relațiile de date pe care modelul le poate înțelege, necesitând o analiză profundă a datelor de intrare. Celelalte două modele, folosite ca referință în benchmark-uri, s-au dovedit inferioare în prezicerea corectă a tuturor celor trei metrii.

Din cauza limitărilor resurselor noastre hardware, sfera acestui proiect a fost limitată. Nu au fost efectuate suficiente iterații de antrenament pentru a atinge potențialul maxim al acestor modele; totuși, considerăm că au fost făcute suficiente simulări pentru a ne face o idee corectă despre capacitatele fiecărui model. Aceste constatări sunt explicate și extinse în capitolele următoare.

## 2. Context

### 2.1 Metode clasice

**Retele Neuronale Convolutionale** CNN-urile [6] sunt folosite în mod tradițional pentru lucrul cu imagini, cum ar fi clasificarea imaginilor și extragerea caracteristicilor. Luați în considerare o hartă a caracteristicilor de intrare  $X \in \mathbb{R}^{H \times W}$  și un filtru de convolução 2D  $W \in \mathbb{R}^{K_H \times K_W}$ , unde  $K_H$  și  $K_W$  sunt înălțimea și, respectiv, lățimea nucleului. Operația de convoluție 2D cu pas (stride)  $S$  și padding  $P$  produce o hartă caracteristică de ieșire  $Y \in \mathbb{R}^{H' \times W'}$ , unde:

$$H' = \left\lfloor \frac{H + 2P - K_H}{S} \right\rfloor + 1 \quad W' = \left\lfloor \frac{W + 2P - K_W}{S} \right\rfloor + 1 \quad (2.1)$$

Fiecare element  $Y[i, j]$ , cu stride  $S$  și padding  $P$  este calculat ca:

$$Y[i, j] = \sum_{m=0}^{K_H-1} \sum_{n=0}^{K_W-1} W[m, n] \cdot X[i \cdot S + m - P, j \cdot S + n - P] \quad (2.2)$$

Sarcina modelului este de a determina ponderile lui  $W$ . Filtrul glisează spațial peste intrarea  $X$ . La fiecare poziție, înmulțirea în funcție de elemente este efectuată între filtru și regiunea corespunzătoare din intrare, urmată de o însumare pentru a produce o singură ieșire scalară.

CNN-urile sunt puternice datorită localității și echivalenței translarii. Localitatea asigură că CNN-urile se concentrează pe modele locale, reducând complexitatea de calcul și captând caracteristici esențiale. Echivalența translarii permite CNN-urilor să recunoască tipare indiferent de locația lor spațială, asigurând o detectare consecventă a caracteristicilor în intrare.

**Modele Transformer** Modelele Transformer au fost introduse de [1], inițial pentru modelarea limbajului. Utilizează mecanismul de Atentie pentru a înțelege contextul și pentru a prezice următorul simbol. Matricea de atenție este calculată utilizând o operație "Scaled Dot-Product Attention", cu tensorii key, query și value, de dimensiune  $d_k$ :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.3)$$

Această abordare permite o paralelizare mai mare, deoarece atenția necesită doar input-ul global și nu output-ul anterior a stării ascunse, ca în cazul RNN-urilor. Cu toate acestea, această operație se scalează pătratic pe măsură ce lungimea contextului crește. Pentru a genera o secvență, un transformer necesită ca toate token-urile generate anterior să fie atașate la intrare pentru a genera următorul token.

## 2.2 Modele in Spatiul Starilor

SSM-urile au fost introduse pentru prima dată în anii 1960 în domeniul ingineriei de control. Aici ne referim la ele în contextul inteligenței artificiale. SSM-urile definesc un sistem continuu invariant în timp, cu o stare ascunsă fixă  $h \in \mathbb{R}^N$ , trei parametri de învățare ( $A$ ,  $B$ ,  $C$ ) și o intrare  $x \in \mathbb{R}^T$  cu o ieșire  $y \in \mathbb{R}^T$ :

$$\begin{aligned} h'(t) &= Ah(t) + Bx(t) & h_t &= \bar{A}h_{t-1} + \bar{B}x_t \\ y(t) &= Ch(t) & y_t &= Ch_t \end{aligned} \quad (2.4) \quad (2.5)$$

Pentru a utiliza acest model, trebuie să fie discret, aşa cum se vede în ecuații 2.5. Importanța ca sistemul să fie continuu este că adaugă un element temporal, deoarece permite eșantionarea parametrilor discreți la diferite intervale. Această recurență liniară poate fi calculată eficient în paralel folosind o convoluție.

Informația este comprimată în spațiul latent  $h$ . Principalul avantaj al SSM este că se scalează liniar cu contextul, permitând performanțe mai bune decât Transformers la o scară mai mare. Complexitatea teoretică a unui model Mamba implementat folosind FFT este  $O(L \log(L))$ , în timp ce pentru un transformator este  $O(L^2 \cdot D)$ , unde  $L$  este dimensiunea secvenței și  $D$  este dimensiunea de încorporare.

## 2.3 Modele Selective in Spatiul Starilor

Modele Selective in Spatiul Starilor, abreviate și ca S6, au fost introduse (în domeniul AI) de [3] pentru a provoca arhitectura Transformer. Ei adaugă mecanismul de selecție la SSM, pentru a modela mai bine relațiile de date. Ideea de bază a acestui mecanism este de a avea parametrii ca funcții dependente de timp. În acest fel, informațiile pot fi memorate și uitate, permitând scalarea contextului lung:

$$\begin{aligned} h'(t) &= Ah(t) + B(t)x(t) & h_t &= \bar{A}_t h_{t-1} + \bar{B}_t x_t \\ y(t) &= C(t)h(t) & y_t &= C_t h_t \end{aligned} \quad (2.6) \quad (2.7)$$

SSSM-urile modeleză  $\Delta_t$  ca o funcție de timp, permitând modelului să învețe cum să discretezeze sistemul continuu. Acest abordare rezolva problemele anterioare, cum ar fi capacitatea de a modifica starea ascunsă pe baza informațiilor noi. Dorim să putem ignora informațiile irelevante și să ne resetăm starea ascunsă. Când  $\Delta_t \rightarrow 1$ ,  $\bar{A}_t$  tinde la 1, filtrând astfel token-ul. De asemenea, putem reseta starea ascunsă, când  $\Delta_t \rightarrow \infty$ , ceea ce are sens și din punct de vedere al timpului, deoarece informațiile sunt uitate după mult timp:

$$\bar{A}_t = \exp(\Delta_t A) \quad (2.8) \quad \bar{B}_t = (\bar{A}_t - 1) \frac{B(t)}{A} \quad (2.9)$$

Mamba, care este o arhitectură S6, beneficiază și de îmbunătățiri ale vitezei, datorită designului său optimizat pentru hardware (accesare optimizată a memoriei SRAM și HBM).

### 3. Lucrari relevante

**Invatare End to End** Prima lucrare publicată de introducere a rețelelor neuronale end-to-end în domeniul conducerii autonome este PilotNet [7] de la Nvidia. Acestea prezintă un model care ia ca intrare un singur cadru de imagine, din vedere din față a vehiculului, și calculează unghiul volanului. Ideea de bază este că toată procesarea se face în interiorul unei rețele neuronale, o ”cutie neagră”, care nu trebuie să învețe în mod explicit toate caracteristicile pe care le-ar putea avea un om: segmente de bandă, suprafața drumului, etc. Ea dezvoltă toată logica necesară pentru a crea cu succes pixelii imaginii la unghiul volanului. Arhitectura este împărțită în două părți principale: o listă de straturi convoluționale pentru a reduce progresiv dimensiunea de intrare, urmată de straturi complet conectate pentru a scoate în cele din urmă valoarea dorită. Acest lucru funcționează foarte bine, datorită părtinirii inductive a CNN-urilor, echivalenței traducerii și localității. În scopul acestui benchmark, am modificat PilotNet pentru a îmbina caracteristicile imaginii, după straturile convoluționale, cu datele de traseu trecut, înainte de a le alimenta în straturile complet conectate. În implementarea noastră, rezultatul este o stare ascunsă, discutată în continuare în capitolul 6.

**Modele de Seq2Seq** În unele scenarii specifice, existența unui context de date din trecut poate îmbunătăți predicțiile viitoare. Acest lucru este valabil în cazul conducerii autonome, demonstrat de ReasonNet [8] și alți cercetători. Principalul defect al PilotNet constă în simplitatea sa, luând ca context doar cadrul actual. ReasonNet propune un model cu memorie pe termen scurt și pe termen lung. Acest lucru poate ajuta modelul să înțeleagă mai bine mediul său, de exemplu, contextul pe termen lung poate ajuta în scenarii de ocluzie, cum ar fi un obstacol blocat de la vedere de către o altă mașină. Arhitectura ReasonNet este destul de complexă, deoarece funcționează cu mai multe date de telemetrie: LiDAR, intrare multi-view, date de semne de trafic etc. Acest lucru poate îmbunătăți modelul pentru un scenariu real; cu toate acestea, am respins aceste caracteristici pentru experimentul nostru, adoptând o abordare mai simplă, urmând [9].

**Modele Transformer** Mecanismul atenției a fost conceput inițial pentru a fi utilizat în sarcinile de procesare a limbajului natural [1]. Această lucrare a fost extinsă de [2] pentru a permite utilizarea modelelor Transformer în sarcini de viziune și video [10]. S-a dovedit că un Transformer poate depăși un CNN în clasificarea imaginilor, totuși, cu prețul unui set de date mai mare și mai multă instruire. Capacitatea lor de a condensa astfel de informații a permis modelelor multimodale să funcționeze, putem avea un model care înțelege atât limbajul, cât și viziunea și poate funcționa pe ambele. ViT-urile sunt cel mai bine utilizate în situațiile în care există caracteristici complexe și nuanțate, în timp ce prezintă doar rezultate marginale mai bune în problemele comune. ViT-urile necesită mai multe resurse de calcul, deoarece Transformers trebuie să depășească prin antrenament lipsa de părtiniri inductive. Similar cu Mamba, la începutul antrenamentului, Transformerul nu are informații poziționale (sau temporale în cazul Transformatorului Video), deoarece trebuie să învețe înglobările poziționale pentru patch-urile de imagine. O abordare similară este folosită de modelele Mamba Vision [4], cu toate acestea, folosind un bloc de codificator Mamba la bază în loc de un bloc de atenție cu mai multe capete.

## 4. Rezultate

Pentru a verifica capabilitățile modelului nostru, am implementat două modele suplimentare: PilotNet [7] și Seq2Seq [9, 11]. Aceste două modele variază în modul în care valorifică caracteristicile temporale și în complexitatea lor. PilotNet folosește doar cadrul curent ca context, în timp ce Seq2Seq folosește o arhitectură codificator-decodor pentru a procesa cadrele anterioare. În ceea ce privește arhitectura lor, PilotNet folosește o serie de straturi convecționale urmate de straturi liniare, iar Seq2Seq folosește un encoder sub forma unui extractor de caracteristici RegNet și un decodor folosind o rețea GRU. Modelul nostru, Steer, a arătat rezultate promițătoare în comparație, totuși credem că ar putea avea îmbunătățiri majore cu mai multe instruiriri și date.

### 4.1 Benchmarks

Am testat pierderea finală de validare, capacitatea de a înțelege relațiile de date și eficiența computațională.

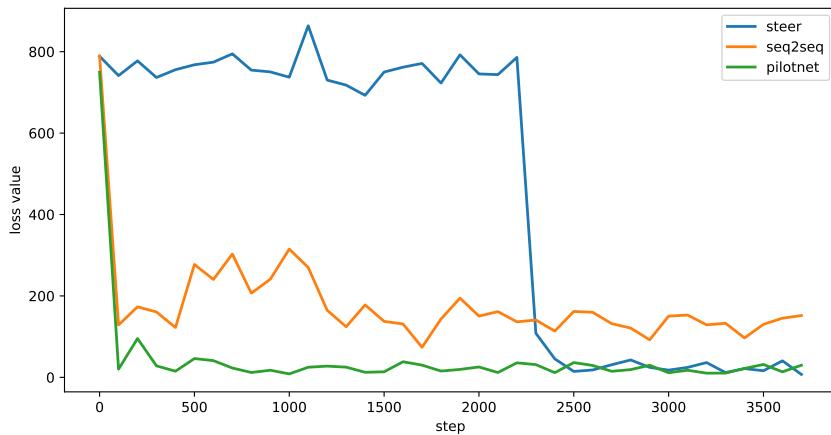


Figure 4.1. Valorile validation loss al fiecăruia model, calculate la intervale de 100 de pași.

În timpul procesului de antrenament, aşa cum se vede în figura 4.1, modelul Steer a durat mult mai mult ( 6 epoci, 2200 de pași) pentru a obține o îmbunătățire majoră a pierderii. Acest lucru este în contrast cu forma clasică a ”bețului de hochei” a graficelor de pierderi ale celorlalte două modele. Acest fenomen este bine cunoscut pentru modelele bazate pe Transformer și Mamba [2, 12], deoarece aceste familii de modele nu au părtiniri inductive, cum ar fi localitatea. Informațiile despre poziție sunt învățate prin adăugarea de înglobări pozitionale și temporale care pot fi învățate [10]. După cum se arată în grafic, modelul necesită timp pentru a învăța aceste modele, cu toate acestea, oferă mai multă flexibilitate în modelarea relațiilor de date. Din cauza acestei și a altor detalii legate de natura liniară a modelului, acesta necesită mult mai mult antrenament decât o rețea neuronală de convecție. Cu condiția ca modelul să aibă suficiente iterații și date de antrenament, poate trece modelele clasice. Pierderea finală de validare este comparabilă cu cea a PilotNet, însă, aşa cum se discută în secțiunile ulterioare, cu o relație de date mult

Model	Milliseconds/Iteration	Params	MFLOPs	Validation Loss
PilotNet	89,31	830.288	119	29.70
Seq2Seq	363,93	5.940.334	9417	151.80
<b>Steer</b>	<b>177,18</b>	<b>6.409.756</b>	<b>5130</b>	<b>7.16</b>

Table 4.1. Resursele utilizate de fiecare model, masurate pe o placă RTX 2060.

mai bună. Modelul Seq2Seq a avut o pierdere mult mai mare la sfârșitul antrenamentului. Valorile finale ale pierderilor pot fi găsite în tabel ??.

Performanța modelului poate fi observată din tabel ?? . În ciuda numărului relativ mare de parametri, în comparație cu celelalte două modele, modelul nostru a obținut performanțe bune cu 0,2 secunde pe trecere înainte. Modelul nostru a arătat o eficiență de calcul mai bună decât modelul Seq2Seq, care are și mai mulți parametri. Acest lucru poate fi optimizat în continuare cu nuclee unificate, cu toate acestea, au fost deja adoptate câteva tehnici de optimizare. În primul rând, pentru blocurile de bază Mamba am folosit implementarea de referință [3], care acordă o mare atenție unui design conștient de hardware. Această implementare este optimizată pentru un consum minim de lățime de bandă a GPU, reducând cel mai mare blocaj. Pe lângă un design orientat către date, acesta dispune și de nuclee Triton optimizate pentru o operare eficientă de scanare. Aceste blocuri Mamba sunt utilizate în arhitectura generală, care este, de asemenea, optimizată ulterior prin tortă folosind o fază de compilare.

## 4.2 Observatii

Una dintre provocările de bază impuse de natura acestei sarcini este reprezentată de calcularea mai multor valori de ieșire. Traекторia, unghiul de virare și viteza pot fi văzute în valori corelate: viteza influențează traectoria, unghiul de virare schimbă traseul etc. Din cele trei modele prezentate, doar Steer a arătat promisiuni în modelarea unor relații bune de date. Acest lucru este exemplificat în figura 4.3. Poza prezintă 3 cadre dintr-o înregistrare în care o mașină oprește la un semafor. În primul cadru are viteza și traectorie normale, în timp ce în cadrele următoare încetinesc. Viteza are o valoare semnificativ mai mică, în timp ce volanul este afectat doar marginal. În timp ce, la alte modele, schimbarea vitezei ar schimba liniar unghiul de virare, ceea ce este greșit. De asemenea, traectoria pare să se "să se prăbușească" în ultimul cadru, care este un comportament complet dedus de model, deoarece nu a fost antrenat pe segmente cu o viteza mediană sub  $10 \frac{\text{km}}{\text{h}}$ . Acest comportament poate fi considerat, totuși, corect, deoarece modelul nu mai poate vedea o



Figure 4.2. Predictii ale modelului după insuficientă antrenare.

cale viitoare în acel moment.

Un alt fapt interesant este subliniat de progresul realizat de model. În primele etape ale antrenamentului, rezultatul modelului poate fi văzut în figura 4.2. Prima imagine arată o cale complet aleatorie și greșită. Între timp, în a doua imagine, putem observa deja că modelul începe să învețe forma trapezoidală a traectoriei. Este de așteptat ca modelul să dureze ceva timp pentru a obține o setare implicită bună. Avantajul arhitecturii noastre este că, în momentul în care modelul realizează această mică corelatie, își dă deja seama de înglobări spațiale și temporale decente care ajută la îmbunătățirile ulterioare. Modelul PilotNet poate fi observat stagnând după câteva epoci.

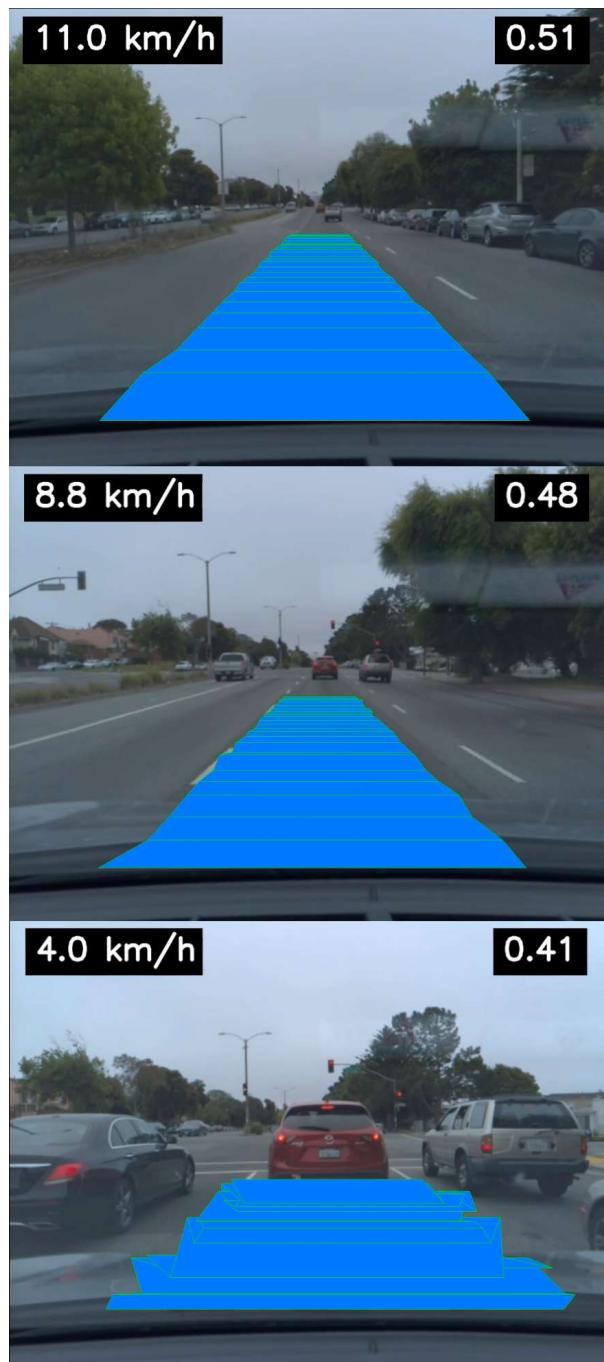


Figure 4.3. Predictii ale modelului în timpul opririi mașinii la un semafor.

## 5. Procesarea datelor

### 5.1 Dataset

Setul de date folosit pentru antrenament este **Comma 2k19** [13]. A fost creat de compania Comma AI și licențiat sub licență MIT. Este aranjat în segmente video din 2019, fiecare conținând un videoclip de 1 minut împreună cu datele senzorului (relevant aici: unghiul volanului și viteza vehiculului). Datele au fost colectate de o mașină care circula pe o autostradă din California, între San Jose și San Francisco. Setul de date are o dimensiune totală de aproximativ 80 GB și este împărțit în opt bucăți de 10 GB.

Din cauza limitărilor noastre de resurse hardware, am ales să lucrăm doar cu prima bucată a setului de date. Modelul a fost antrenat pe 4 epoci cu doar 20% din prima bucată. Pentru ca modelul să se îmbunătățească și să învețe, este nevoie de mai multe epoci (iterație completă a setului de date), cu toate acestea, resursa noastră nu ne-a permis să repetăm de mai multe ori pe întregul set de date. Folosind doar un fragment din datele disponibile, am fost activați să rulăm pentru mai multe "sub-epoci" și să obținem îmbunătățiri ale modelului. După cum se discută în capitolul Lucrări ulterioare, ar trebui realizate îmbunătățiri majore în calitatea modelului dacă acesta ar fi instruit pe mai multe date și pentru mai multe epoci.

### 5.2 Preprocesare

Pentru o instruire eficientă, trebuia să ne pregătim setul de date într-un format gata de servit. Această sarcină a constat în organizarea datelor, alegerea informațiilor utile, sincronizarea datelor senzorilor, conversia în formate de date utile și curățarea datelor. Am scris un script pentru realizarea acestei lucrări, `prepare.py`, care ar trebui să fie rulat înainte de scriptul de antrenament. În primul rând, toate segmentele dintr-o rută sunt îmbinate împreună în ordine cronologică, aceasta include cadre video și datele senzorilor. Cadrele sunt extrase din videoclipuri și salvate ca imagini comprimate. Datele senzorului sunt apoi sincronizate cu numărul cadrului. În acest fel, putem extrage o intrare de model utilizând un singur id de marcaj temporal, care leagă cadrul video curent, cadrele trecute pentru context și datele senzorului curent. Datele senzorului sunt scrise în fișiere matrice numpy comprimate, astfel minimizăm dimensiunea fișierului și obținem citire și încărcare eficiente. Un exemplu de prezentare generală a directorului procesat poate fi văzut în figura 5.1 de mai jos. Facem o împărțire a setului de date, 80% de formare și 20% validare.

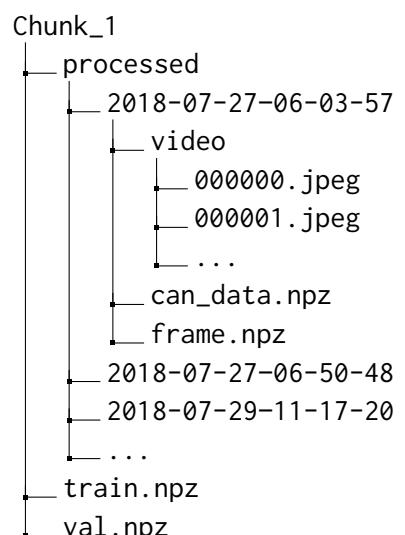


Figure 5.1 Un exemplu de un chunk procesat.

### 5.3 Augmentarea datelor si curatare

Pentru a profita la maximum de datele disponibile, am eliminat valorile aberante și datele inutile. În cazul nostru, am ales să eliminăm segmentele video în care mașina are o viteză medie mai mică de  $10 \frac{\text{km}}{\text{h}}$ , precum și cadrele cu marcaje temporale care nu permit contextul temporal necesar. Pentru antrenament am folosit un context de 10 cadre, adică am putea folosi cadre începând doar de la 11 și mergând până la ultimul indice minus 30, pentru că avem nevoie de următoarele 30 de cadre pentru a calcula pierderea. După aceasta, cadrele au fost reduse la 224 cu 224. Expunerea cadrului a fost ajustată aleatoriu pentru cadrele selectate, în speranța de a reduce distorsionile modelului.

### 5.4 Vizualizare

În scopul modelului nostru, am ales să folosim doar datele CAN, deoarece oferă o modalitate simplă de sincronizare a intrărilor senzorului cu cadrele video și are o marjă de eroare relativ mică. Setul de date oferă date post-procesate, precum și date GNSS, așa cum se vede în graficul de mai jos.

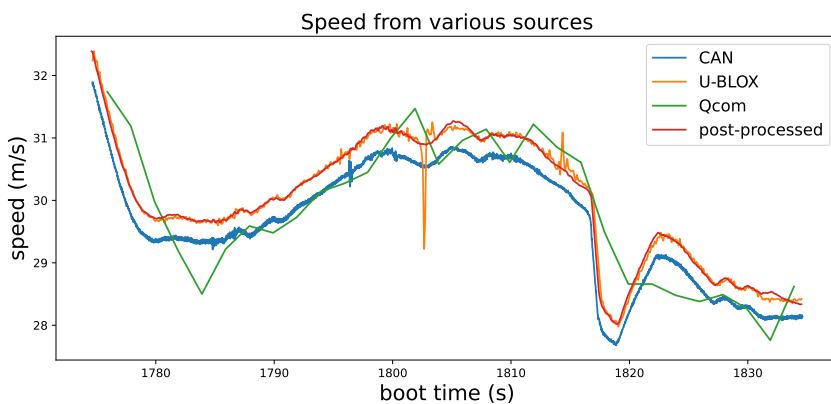


Figure 5.2. Date privind viteza vehiculului, colectate de la diferiti senzori.

Intr-o alta reprezentare a orientării mașinii, roll și pitch sunt aproape de zero, cu toate acestea, senzorii nu sunt montați perfect, așa că au erori. Se așteaptă ca viarea să se schimbe pe măsură ce vehiculul ia o viraj.

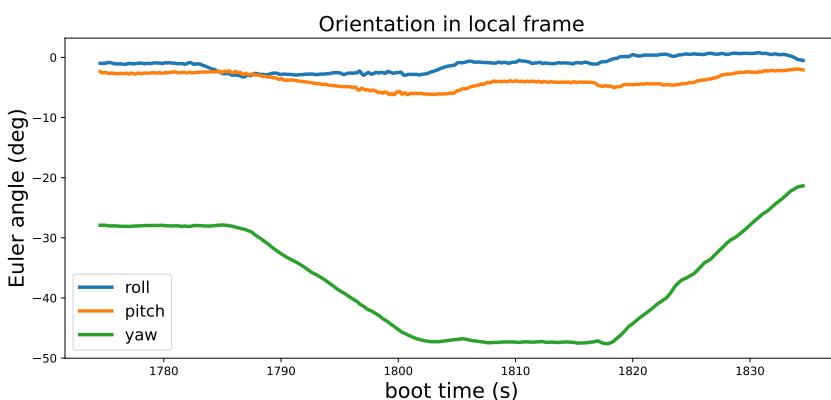


Figure 5.3. Unghiuri Euler fata de planul local.

## 6. Arhitectura modelului

Pentru proiectarea acestui model, am urmat arhitectura Video Mamba [5], care se bazează în principal, pe principiile introduse de modelul ViT [2].

**Date de intrare** Modelul ia ca intrare doi tensori. Contextul video este compus din  $T + 1$  cadre trecute, cea suplimentară fiind cadrul curent și este de formă  $(B, C, T + 1, W, H)$ . Informațiile despre calea trecută sunt formate din matrice  $T + 1$ , fiecare cu trei numere întregi: coordonatele cadru local x, y și z la marcadul de timp specific.

**Date de ieșire** Modelul produce un singur tensor de formă  $(B, T+1, 3)$ , cu valori pentru traectoria viitoarei trasee, viteza vehiculului și unghiul volanului. Datele pentru traectorie pot fi găsite în primele elemente  $T$ , fiecare element conținând coordonatele xyz. Elementul  $T + 1$  are unghiul de virare la poziția 0 și viteza la poziția 1, ultima poziție fiind lăsată ca 0.0. Am decis această abordare pentru rezultatele pentru eficiență, deoarece este mai rapid să citiți și să mutați un singur tensor numeric, spre deosebire de un dicționar cu trei tensori.

**Blocuri codificate** Pentru blocurile principale Mamba[3], am folosit o furcă de la ViM[4], care a adăugat scanări bidirectionale. Acest lucru îmbunătățește recunoașterea modelelor în contextul sarcinilor de vedere. Acestea sunt implementate folosind nuclee Triton, fuzionate mai târziu și cu compilatorul torch.

**Codificare video** Primul bloc din conducta de codificare video este pentru extragerea de patch-uri de imagine 3D. Aceasta este cunoscută sub numele de "Tubelet Embeddings", iar o vizualizare poate fi văzută în figura 6. Se realizează prin utilizarea unei convoluții 3D a dimensiunii nucleului și a pasului egal cu  $(K, P, P)$  (adică,  $1 \times 16 \times 16$ ). Videoclipul  $X \in \mathbb{R}^{T \times 3 \times W \times H}$  este redimensionat la  $X_p \in \mathbb{R}^{T_p \times C}$ , cu  $T_p = T \cdot \frac{W}{P} \cdot \frac{H}{P}$  și  $C$  fiind dimensiunea de încorporare. În continuare, adăugăm două înglobări care pot fi învățate, una pentru informații spațiale ( $p_s \in \mathbb{R}^{(\frac{W \cdot H}{P^2} + 1) \times C}$ : cadru local date de poziție) și unul pentru informații temporale ( $p_t \in \mathbb{R}^{T \times C}$ : date globale de timp video):  $X = [X_{cls}, X] + p_s + p_t$ . După crearea patch-urilor, datele sunt trecute prin  $N_1$  blocuri Mamba, cu normalizarea stratului aplicată înaintea fiecărui și cu conexiuni reziduale.

**Codarea căilor** Căile au forma  $(B, T + 1, 3)$ . Deoarece mai târziu trebuie să concatenăm căile și caracteristicile video, acestea vor trebui să aibă aceeași formă  $(B, T_p, C)$  sau  $(B, 1, C)$ . Pentru considerente de performanță, deoarece informațiile despre cale sunt mult mai mici decât videoclipul, caracteristicile căii sunt avansate în doi pași. În primul rând, folosim un strat liniar pentru a obține  $P_1 \in \mathbb{R}^{\frac{C}{6}}$ , din  $P_0 \in \mathbb{R}^{T \times 3}$ . Această codificare va fi transmisă prin blocuri Mamba  $N_2$ , similar codării video; totuși, folosim  $N_2 = \frac{N_1}{2}$ . După aceasta, caracteristicile traseului sunt upscale la  $P \in \mathbb{R}^C$ , de formă  $(B, 1, C)$ . În cele din urmă, adăugăm cele două caracteristici pentru a obține funcțiile ascunse.

**Head** Pentru procesarea suplimentară a caracteristicilor ascunse, folosim un cap MLP.

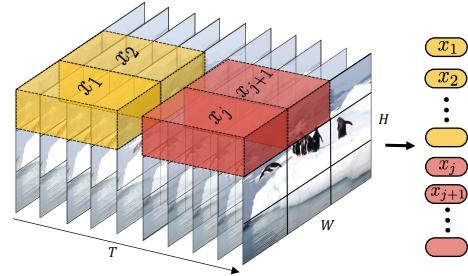


Figure 6.1. Vizualizare a Tubelet embeddings [10].

MLP conține două straturi cu o neliniaritate GELU. Pentru a reduce supraadaptarea, se folosește, de asemenea, un strat de renunțare, cu valoarea 0,1. O prezentare generală a întregului model poate fi văzută în figura 6.2.

Algoritmul este prezentat pe scurt în ecuațiile de mai jos:

$$\mathbf{X} = [\mathbf{X}_{cls}, \mathbf{X}] + \mathbf{p}_s + \mathbf{p}_t, \quad \mathbf{X} \in \mathbb{R}^{(T \cdot \frac{W \cdot H}{P^2} + 1) \times C}, \mathbf{p}_s \in \mathbb{R}^{(\frac{W \cdot H}{P^2} + 1) \times C}, \mathbf{p}_t \in \mathbb{R}^{T \times C} \quad (6.1)$$

$$\mathbf{P} = SSSM(LN(\mathbf{P}_i)) + \mathbf{P}_i, \quad \mathbf{P} \in \mathbb{R}^{1 \times C}, i = 1 \dots N_2 \quad (6.2)$$

$$\mathbf{E} = \mathbf{P}_{N_2} + \mathbf{X}, \quad \mathbf{E} \in \mathbb{R}^{(T \cdot \frac{W \cdot H}{P^2} + 1) \times C} \quad (6.3)$$

$$\hat{\mathbf{y}} = MLP(E), \quad \hat{\mathbf{y}} \in \mathbb{R}^C \quad (6.4)$$

**Ieșiri de calcul** Caracteristicile de ieșire reprezintă doar o altă stare ascunsă în model. Pentru a calcula cele trei valori de ieșire a modelului (traекторia viitoare, viteza și unghiul volanului), starea ascunsă este transmisă la trei MLP-uri. Fiecare are două straturi liniare cu o neliniaritate GELU între ele. Dimensiunea stratului fiecărui este proporțională cu dimensiunea de ieșire. Acest lucru asigură că modelul poate extrage caracteristicile relevante pentru fiecare valoare.

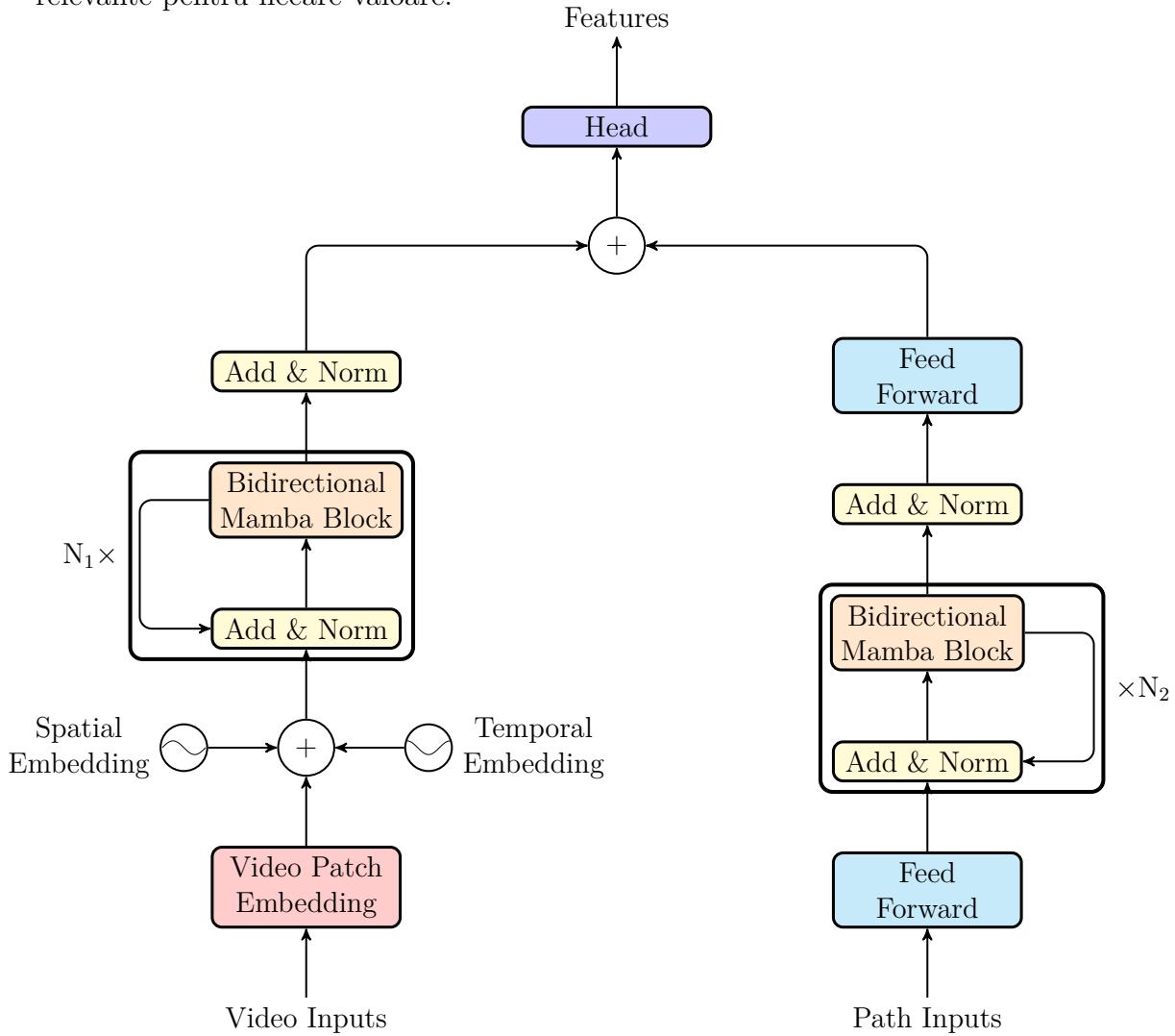


Figure 6.2. Arhitectura modelului Steer.

## 7. Detalii privind antrenarea

### 7.1 Rata de invatare optima

Pentru a găsi rata optimă de învățare, am scris un script care rulează o buclă de antrenament pentru iterații de 500 de pasi. Rata de învățare este modificată liniar de la  $10^{-6.0}$  la  $10^{-1.5}$  pe parcursul întregului proces. Valoarea pierderii a fost calculată și reprezentată grafic la fiecare pas, astfel încât să putem analiza ce rata de învățare dă o optimizare stabilă. Graficul rezultat poate fi văzut în figura 7.1. Din motive de vizibilitate, valoarea loss-ului a fost plafonată la 1000. Pentru analiza plotului, vorbim în termeni de exponent al ratei de învățare. Intervalul  $[-6, -4]$  nu duce la nicio îmbunătățire pentru model, ceea ce înseamnă că rata de învățare este prea mică. Intervalul  $[-4, -2]$  este stabil și optim. Exponenții mari de -2 fac modelul să explodeze, ducând la o optimizare instabilă.

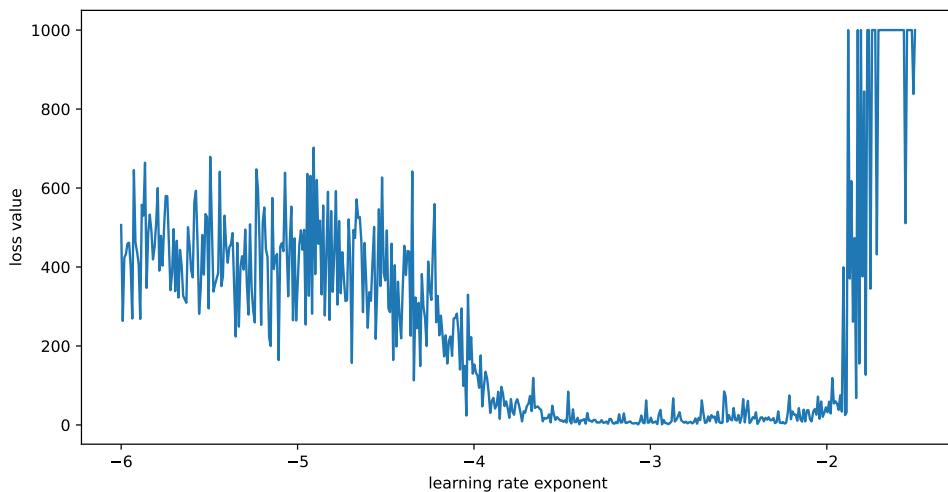


Figure 7.1. Grafic experimental pentru a găsi rata de invatare optima.

Pentru procesul de formare propriu-zis, am folosit o rată minimă de învățare de  $5 \cdot 10^{-3}$  și un maxim de  $10^{-5}$ . Am făcut o încălzire liniară peste 3% din totalul iterațiilor. După aceea, am făcut o scădere a ratei de învățare cosinus, de la maxim la minim, peste restul de 97% din pașii de antrenament.

### 7.2 Procesul de antrenare

Arhitectura modelului, Mamba, necesită mai multe iterații de antrenament pentru rezultate bune. Din cauza asta și a limitărilor hardware, am ales să folosim doar 2% din setul de date disponibil. Având în vedere acest lucru, o epocă, cu o dimensiune a lotului de 4, durează doar pași de 370. Am optat pentru o rulare pe epoci de 10, cu iterații de 3700. Modelul a fost antrenat pe un GPU RTX 2060 6GB. AdamW a fost folosit ca optimizator, cu  $\beta_1 = 0,9$  și  $\beta_2 = 0,999$ ,  $\epsilon = 1e-08$  și o scădere a greutății de  $10^{-2}$ . Rezultatele au fost monitorizate în timp real folosind Wandb.

Am folosit o pierdere L1 pentru traiectoria traiectoriei viitoare și o pierdere MSE pentru unghiul și viteza volanului. Am adăugat aceste pierderi, pentru o valoare finală a pierderii. Am ales diferite funcții de pierdere, pentru a sublinia o pierdere mai mică pentru traiectorie și pentru a permite vitezei și unghiului de virare să aibă pante mai mari comparativ. Aceasta este valoarea pe care am raportat-o și pe care am aplicat propagarea inversă.

**Tehnici de regularizare** Pentru a reduce supraadaptarea, am aplicat o serie de tehnici de regularizare. Straturile de renunțare, cu o valoare de 0.1, au fost utilizate între mai multe straturi liniare. Au fost utilizate conexiuni reziduale, între conexiunile bloc și înainte și după secvența de codificare. Am folosit decăderea ratei de învățare, cu valori aşa cum se vede în figura 7.2. Gradientii au fost tăiați la o valoare de 5.0, în timp ce am experimentat și cu o valoare de 1.0, ceea ce a condus la rezultate mai proaste. Normalizarea stratului a fost aplicată după fiecare conexiune reziduală.

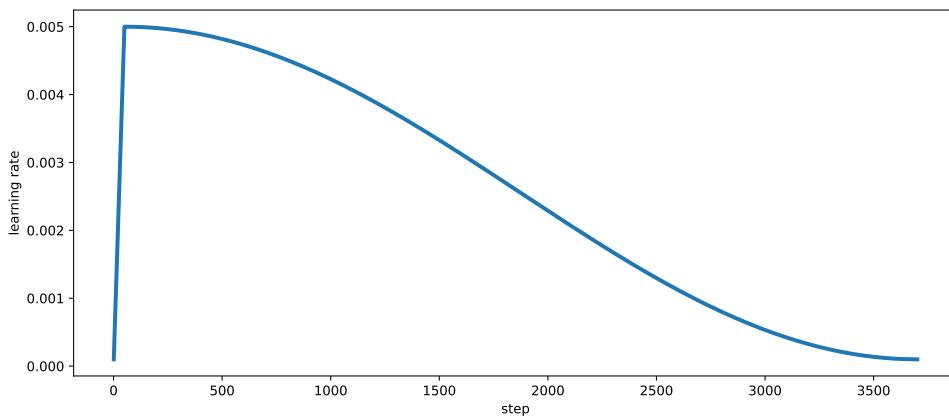


Figure 7.2. Valoarea ratei de invatare.

**Considerații de performanță** Pentru a profita la maximum de hardware-ul disponibil, am făcut câteva optimizări de performanță. Am folosit cadrul PyTorch, cu TF32 și precizie de multiplicare a matricei "înaltă", în loc de "cel mai mare". Trecerea înainte a fost turnată la bfloat16 și am aplicat scalarea gradientului pentru a reduce erorile de precizie. Modelul a returnat un singur tensor, care conține toate valorile, în loc de un dicționar, pentru a îmbunătăți operațiunile critice ale memoriei. În timpul procesului de antrenament, am măsurat  $\sim 700 \frac{\text{ms}}{\text{iter}}$ .

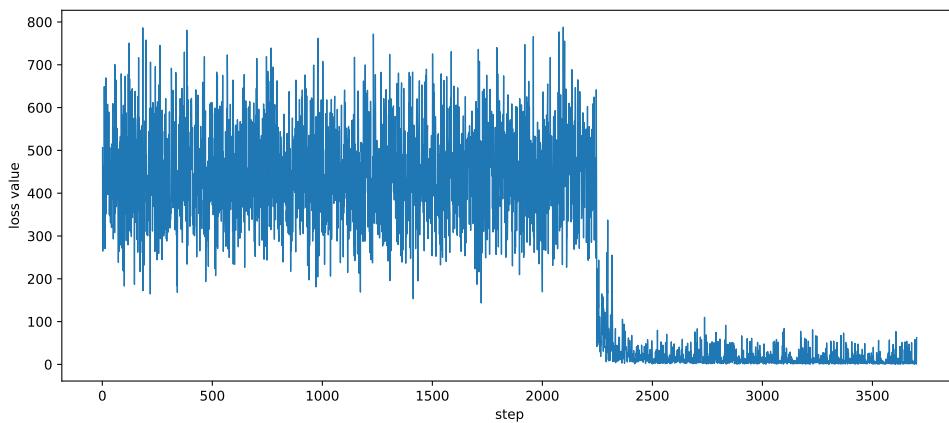


Figure 7.3. Valorile loss-ului in timpului antrenarii.

## 8. Concluzie

Experimentul nostru prezintă un model bazat pe Mamba capabil să învețe cum să conduceă un vehicul. Este eficient din punct de vedere al calculului, cu complexitate liniară și un design conștient de hardware. Funcționează comparabil cu modelele anterioare de ultimă generație, PilotNet și Seq2Seq în experimentele noastre. Are o amprentă de memorie redusă, în comparație cu Transformers și alte arhitecturi, permitându-i să fie utilizat cu rezoluții mai mari de imagine și context temporal mai lung. Deși durează mai mult antrenamentul decât un model de bază, demonstrează că este capabil să obțină rezultate mai bune.

Aceste constatări sunt deosebit de relevante în contextul sistemelor încorporate. Acestea necesită o latență și un consum de energie minime, astfel încât necesită un model eficient pentru inferență locală. Modelul nostru se potrivește acestor cerințe, cu capacitați mari. Cu mai multă muncă pusă în modelele de la capăt la cap al spațiului de stat selectiv, acestea ar putea deveni o arhitectură universală pentru implementarea la cerere pentru dispozitive încorporate și de consum redus.

### 8.1 Modalitati de imbunatatire

Pentru rezultate mult mai bune, modelul necesită mai multă pregătire. Ar trebui să fie antrenat pentru cel puțin 30 de epoci pe setul de date complet. Este unul dintre dezavantajele arhitecturii că necesită mai multă pregătire decât un model clasic, cu toate acestea, cantitatea de pregătire efectuată a fost insuficientă chiar și pentru un model clasic (CNN). Acest lucru se datorează resurselor noastre hardware limitate.

Mărirea datelor ar trebui, de asemenea, utilizată pentru a pregăti modelul pentru scenarii mai diverse. În plus, utilizarea unui simulator de conducere poate îmbunătăți mult distribuirea datelor și flexibilitatea antrenamentului, aşa cum se precizează în [11]. Modelele bazate pe Mamba se scalează liniar în funcție de context, acest lucru permite caracteristici de date mult mai mari și permite posibile modele fără simbol[14]. Acest lucru este important într-un context în care folosim o abordare de învățare prin întărire, în care modelul principal învăță dintr-o simulare de conducere, generată și de un alt model. Arhitectura Mamba permite posibilitatea modelelor mari de putere redusă cu context suficient.

# Bibliografie

- [1] Ashish Vaswani et al. “Attention Is All You Need”. Aug. 1, 2023. DOI: [10.48550/arXiv.1706.03762](https://doi.org/10.48550/arXiv.1706.03762). arXiv: [1706.03762\[cs\]](https://arxiv.org/abs/1706.03762). URL: <http://arxiv.org/abs/1706.03762> (pages 1, 3, 5).
- [2] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. June 3, 2021. DOI: [10.48550/arXiv.2010.11929](https://doi.org/10.48550/arXiv.2010.11929). arXiv: [2010.11929\[cs\]](https://arxiv.org/abs/2010.11929). URL: <http://arxiv.org/abs/2010.11929> (pages 1, 5, 6, 11).
- [3] Albert Gu and Tri Dao. “Mamba: Linear-Time Sequence Modeling with Selective State Spaces”. May 31, 2024. DOI: [10.48550/arXiv.2312.00752](https://doi.org/10.48550/arXiv.2312.00752). arXiv: [2312.00752\[cs\]](https://arxiv.org/abs/2312.00752). URL: <http://arxiv.org/abs/2312.00752> (pages 1, 4, 7, 11).
- [4] Lianghui Zhu et al. “Vision Mamba: Efficient Visual Representation Learning with Bidirectional State Space Model”. Feb. 10, 2024. DOI: [10.48550/arXiv.2401.09417](https://doi.org/10.48550/arXiv.2401.09417). arXiv: [2401.09417\[cs\]](https://arxiv.org/abs/2401.09417). URL: <http://arxiv.org/abs/2401.09417> (pages 1, 5, 11).
- [5] Kunchang Li et al. “VideoMamba: State Space Model for Efficient Video Understanding”. Mar. 12, 2024. DOI: [10.48550/arXiv.2403.06977](https://doi.org/10.48550/arXiv.2403.06977). arXiv: [2403.06977\[cs\]](https://arxiv.org/abs/2403.06977). URL: <http://arxiv.org/abs/2403.06977> (pages 1, 11).
- [6] Yann LeCun and Yoshua Bengio. “Convolutional networks for images, speech, and time-series”. In: *The handbook of brain theory and neural networks*. MIT Press, 1995, pp. 255–258 (page 3).
- [7] Mariusz Bojarski et al. “End to End Learning for Self-Driving Cars”. Apr. 25, 2016. arXiv: [1604.07316\[cs\]](https://arxiv.org/abs/1604.07316). URL: <http://arxiv.org/abs/1604.07316> (pages 5, 6).
- [8] Hao Shao et al. “ReasonNet: End-to-End Driving with Temporal and Global Reasoning”. May 17, 2023. DOI: [10.48550/arXiv.2305.10507](https://doi.org/10.48550/arXiv.2305.10507). arXiv: [2305.10507\[cs\]](https://arxiv.org/abs/2305.10507). URL: <http://arxiv.org/abs/2305.10507> (page 5).
- [9] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. Dec. 14, 2014. arXiv: [1409.3215\[cs\]](https://arxiv.org/abs/1409.3215). URL: <http://arxiv.org/abs/1409.3215> (pages 5, 6).
- [10] Anurag Arnab et al. “ViViT: A Video Vision Transformer”. Nov. 1, 2021. DOI: [10.48550/arXiv.2103.15691](https://doi.org/10.48550/arXiv.2103.15691). arXiv: [2103.15691\[cs\]](https://arxiv.org/abs/2103.15691). URL: <http://arxiv.org/abs/2103.15691> (pages 5, 6, 11).
- [11] Eder Santana and George Hotz. “Learning a Driving Simulator”. Aug. 3, 2016. arXiv: [1608.01230\[cs, stat\]](https://arxiv.org/abs/1608.01230). URL: <http://arxiv.org/abs/1608.01230> (pages 6, 15).
- [12] Rui Xu et al. “Visual Mamba: A Survey and New Outlooks”. July 6, 2024. arXiv: [2404.18861\[cs\]](https://arxiv.org/abs/2404.18861). URL: <http://arxiv.org/abs/2404.18861> (page 6).
- [13] Harald Schafer et al. “A Commute in Data: The comma2k19 Dataset”. Dec. 13, 2018. arXiv: [1812.05752\[cs\]](https://arxiv.org/abs/1812.05752). URL: <http://arxiv.org/abs/1812.05752> (page 9).
- [14] Junxiong Wang et al. “MambaByte: Token-free Selective State Space Model”. Aug. 9, 2024. DOI: [10.48550/arXiv.2401.13660](https://doi.org/10.48550/arXiv.2401.13660). arXiv: [2401.13660\[cs\]](https://arxiv.org/abs/2401.13660). URL: <http://arxiv.org/abs/2401.13660> (page 15).