Supplemental Materials for Efficient Equivariant Transformer for Self-Driving Agent Modeling

Scott Xu[1,2], **Dian Chen**[†], **Kelvin Wong**[1,2], **Chris Zhang**[1,2], **Kion Fallah**[†], **Raquel Urtasun**[1,2]
Waabi[1], University of Toronto[2]
{sxu, kwong, czhang, urtasun}@waabi.ai

1. Theoretical analysis

We provide additional theoretical analysis in the following sections. Section 1.1 provides additional preliminaries for 2D projective geometric algebra $\mathbb{R}^*_{2,0,1}$. Section 1.2 provides proofs of the satisfied properties of our derived $\mathbb{R}^*_{2,0,1}$ encodings. Section 1.3 provides proofs for SE(2) equivariance for each DriveGATr layer.

1.1. Additional geometric algebra preliminaries

In the Euclidean space $\mathbb{R}^n$, one may define a bilinear product of two vectors x and y, called the *geometric product* xy, characterized by the fundamental equation $v^2 = \langle v, v\rangle$ where $\langle\cdot,\cdot\rangle$ is the standard inner product. From the fundamental equation one can deduce that the geometric product is anticommutative on orthogonal basis vectors: for $i \neq j$, $(e_i + e_j)^2 = \langle e_i + e_j, e_i + e_j\rangle \implies e_i^2 + e_ie_j + e_je_i + e_j^2 = e_i^2 + 2\langle e_i, e_j\rangle + e_j^2 \implies e_ie_j = -e_je_i$. Multiplying vectors together give us *multivectors*; for $n = 2$, the Euclidean geometric algebra consists of multivectors of the form $x = x' + x_1e_1 + x_2e_2 + x_{12}e_{12}$, where x', x_1, x_2, x_{12} are real coefficients and $e_{12} := e_1e_2$.

To get a *projective* geometric algebra, we adjoin to $\mathbb{R}^n$ an orthogonal basis vector e_0 with norm zero, that is, $e_0^2 = 0$. For $n = 2$, the resulting algebra, denoted by $\mathbb{R}^*_{2,0,1}$, is 8-dimensional, spanned by the scalar 1, three vectors e_0, e_1, e_2, three *bivectors* e_{01}, e_{20}, e_{12}, and one *pseudoscalar* e_{012}. Using the properties that $e_0^2 = 0, e_1^2 = e_2^2 = 1, e_{ij} = -e_{ji}$ for $i \neq j$, one can compute a table of geometric products xy:

$x\backslash y$	1	e_0	e_1	e_2	e_{01}	e_{20}	e_{12}	e_{012}
1	1	e_0	e_1	e_2	e_{01}	e_{20}	e_{12}	e_{012}
e_0	e_0	0	e_{01}	$-e_{20}$	0	0	e_{012}	0
e_1	e_1	$-e_{01}$	1	e_{12}	$-e_0$	e_{012}	e_2	e_{20}
e_2	e_2	e_{20}	$-e_{12}$	1	e_{012}	e_0	$-e_1$	e_{01}
e_{01}	e_{01}	0	e_0	e_{012}	0	0	$-e_{20}$	0
e_{20}	e_{20}	0	e_{012}	$-e_0$	0	0	e_{01}	0
e_{12}	e_{12}	e_{012}	$-e_2$	e_1	e_{20}	$-e_{01}$	-1	$-e_0$
e_{012}	e_{012}	0	e_{20}	e_{01}	0	0	$-e_0$	0

A *sandwich product* is an expression of the form uxu^{-1}, where u, x are multivectors and u has a multiplicative inverse with respect to the geometric product.

We also define the *wedge product* $x \wedge y$, which is another bilinear product, but is defined by the fundamental equation $v \wedge v = 0$. One can derive that $e_i \wedge e_j = -e_j \wedge e_i$ for orthogonal basis vectors and compute a table of wedge products $x \wedge y$:

[†] Work done while at Waabi.

$x\backslash y$	1	e_0	e_1	e_2	e_{01}	e_{20}	e_{12}	e_{012}
1	1	e_0	e_1	e_2	e_{01}	e_{20}	e_{12}	e_{012}
e_0	e_0	0	e_{01}	$-e_{20}$	0	0	e_{012}	0
e_1	e_1	$-e_{01}$	0	e_{12}	0	e_{012}	0	0
e_2	e_2	e_{20}	$-e_{12}$	0	e_{012}	0	0	0
e_{01}	e_{01}	0	0	e_{012}	0	0	0	0
e_{20}	e_{20}	0	e_{012}	0	0	0	0	0
e_{12}	e_{12}	e_{012}	0	0	0	0	0	0
e_{012}	e_{012}	0	0	0	0	0	0	0

The *multivector dual* $x \mapsto x^*$ is a linear operator, defined for a basis multivector x as the basis multivector such that $x \wedge x^* = e_{012}$. For a general $\mathbb{R}^*_{2,0,1}$ multivector

$$x = x' + x_0e_0 + x_1e_1 + x_2e_2 + x_{01}e_{01} + x_{20}e_{20} + x_{12}e_{12} + x_{012}e_{012}$$

we have

$$x^* := x_{012} + x_{12}e_0 + x_{20}e_1 + x_{01}e_2 + x_2e_{01} + x_1e_{20} + x_0e_{12} + x'e_{012}$$

which has the same coefficients as x, in reverse order.

The *join* operator of two multivectors is defined as $\text{Join}(x, y) = (x^* \wedge y^*)^*$.

Finally, we define *k-grade projection* $\langle\cdot\rangle_k$ for $k = 0, 1, 2, 3$, which selects specific components of x:

$$\begin{aligned}
\langle x\rangle_0 &:= x' \\
\langle x\rangle_1 &:= x_0e_0 + x_1e_1 + x_2e_2 \\
\langle x\rangle_2 &:= x_{01}e_{01} + x_{20}e_{20} + x_{12}e_{12} \\
\langle x\rangle_3 &:= x_{012}e_{012}
\end{aligned}$$

and we define the *invariant inner product* $\langle\cdot,\cdot\rangle$ between two multivectors:

$$\langle x, y\rangle := x'y' + x_1y_1 + x_2y_2 + x_{12}y_{12} \in \mathbb{R}$$

which ignores components containing e_0.

1.2. Encodings

We encode 2D objects and transformations into $\mathbb{R}^*_{2,0,1}$ using Table 1. These encodings satisfy many nice properties described in subsequent propositions.

Proposition 1. *The result of applying an operator u to an object x is given by a sandwich product $u[x] := uxu^{-1}$.*

Remark. *Note that for SE(2)-equivariance, we have only considered when u is a non-mirroring transformation. If u is a mirroring transformation (e.g. a reflection), one must instead use $u[x] := u\hat{x}u^{-1}$ where $\hat{x}$ denotes grade involution, which flips the sign of the odd-grade components of x.*

Proof. Since geometric product is bilinear, then $u[\cdot]$ is linear for any fixed u. As such, it is helpful to first compute sandwich products $u[x]$ for *basis* multivectors x. When $u = t$ is a translation by (a, b),

$$\begin{aligned}
t[1] &= 1 \\
t[e_0] &= (1 - \tfrac{a}{2}e_{01} + \tfrac{b}{2}e_{20})e_0(1 + \tfrac{a}{2}e_{01} - \tfrac{b}{2}e_{20}) \\
&= e_0 \\
t[e_1] &= (1 - \tfrac{a}{2}e_{01} + \tfrac{b}{2}e_{20})e_1(1 + \tfrac{a}{2}e_{01} - \tfrac{b}{2}e_{20}) \\
&= e_1 + \tfrac{a}{2}e_1e_{01} - \tfrac{b}{2}e_1e_{20} - \tfrac{a}{2}e_{01}e_1 + \tfrac{b}{2}e_{20}e_1 \\
&= e_1 - ae_0 \\
t[e_2] &= (1 - \tfrac{a}{2}e_{01} + \tfrac{b}{2}e_{20})e_2(1 + \tfrac{a}{2}e_{01} - \tfrac{b}{2}e_{20})
\end{aligned}$$

$$
\begin{aligned}
&= e_2 + \tfrac{a}{2}e_2e_{01} - \tfrac{b}{2}e_2e_{20} - \tfrac{a}{2}e_{01}e_2 + \tfrac{b}{2}e_{20}e_2 \\
&= e_2 - be_0 \\
t[e_{01}] &= (1 - \tfrac{a}{2}e_{01} + \tfrac{b}{2}e_{20})e_{01}(1 + \tfrac{a}{2}e_{01} - \tfrac{b}{2}e_{20}) \\
&= e_{01} \\
t[e_{20}] &= (1 - \tfrac{a}{2}e_{01} + \tfrac{b}{2}e_{20})e_{20}(1 + \tfrac{a}{2}e_{01} - \tfrac{b}{2}e_{20}) \\
&= e_{20} \\
t[e_{12}] &= (1 - \tfrac{a}{2}e_{01} + \tfrac{b}{2}e_{20})e_{12}(1 + \tfrac{a}{2}e_{01} - \tfrac{b}{2}e_{20}) \\
&= e_{12} + \tfrac{a}{2}e_{12}e_{01} - \tfrac{b}{2}e_{12}e_{20} - \tfrac{a}{2}e_{01}e_{12} + \tfrac{b}{2}e_{20}e_{12} \\
&= e_{12} + be_{01} + ae_{20} \\
t[e_{012}] &= (1 - \tfrac{a}{2}e_{01} + \tfrac{b}{2}e_{20})e_{012}(1 + \tfrac{a}{2}e_{01} - \tfrac{b}{2}e_{20}) \\
&= e_{012}
\end{aligned}
$$

When $u = r$ is a rotation by θ,

$$
\begin{aligned}
r[1] &= 1 \\
r[e_0] &= (\cos\tfrac{\theta}{2} - \sin\tfrac{\theta}{2}e_{12})e_0(\cos\tfrac{\theta}{2} + \sin\tfrac{\theta}{2}e_{12}) \\
&= \cos^2\tfrac{\theta}{2}e_0 + \cos\tfrac{\theta}{2}\sin\tfrac{\theta}{2}e_0e_{12} - \sin\tfrac{\theta}{2}\cos\tfrac{\theta}{2}e_{12}e_0 - \sin^2\tfrac{\theta}{2}e_{12}e_0e_{12} \\
&= e_0 \\
r[e_1] &= (\cos\tfrac{\theta}{2} - \sin\tfrac{\theta}{2}e_{12})e_1(\cos\tfrac{\theta}{2} + \sin\tfrac{\theta}{2}e_{12}) \\
&= \cos^2\tfrac{\theta}{2}e_1 + \cos\tfrac{\theta}{2}\sin\tfrac{\theta}{2}e_1e_{12} - \sin\tfrac{\theta}{2}\cos\tfrac{\theta}{2}e_{12}e_1 - \sin^2\tfrac{\theta}{2}e_{12}e_1e_{12} \\
&= \cos\theta e_1 + \sin\theta e_2 \\
r[e_2] &= (\cos\tfrac{\theta}{2} - \sin\tfrac{\theta}{2}e_{12})e_2(\cos\tfrac{\theta}{2} + \sin\tfrac{\theta}{2}e_{12}) \\
&= \cos^2\tfrac{\theta}{2}e_2 + \cos\tfrac{\theta}{2}\sin\tfrac{\theta}{2}e_2e_{12} - \sin\tfrac{\theta}{2}\cos\tfrac{\theta}{2}e_{12}e_2 - \sin^2\tfrac{\theta}{2}e_{12}e_2e_{12} \\
&= \cos\theta e_2 - \sin\theta e_1 \\
r[e_{01}] &= (\cos\tfrac{\theta}{2} - \sin\tfrac{\theta}{2}e_{12})e_{01}(\cos\tfrac{\theta}{2} + \sin\tfrac{\theta}{2}e_{12}) \\
&= \cos^2\tfrac{\theta}{2}e_{01} + \cos\tfrac{\theta}{2}\sin\tfrac{\theta}{2}e_{01}e_{12} - \sin\tfrac{\theta}{2}\cos\tfrac{\theta}{2}e_{12}e_{01} - \sin^2\tfrac{\theta}{2}e_{12}e_{01}e_{12} \\
&= \cos\theta e_{01} - \sin\theta e_{20} \\
r[e_{20}] &= (\cos\tfrac{\theta}{2} - \sin\tfrac{\theta}{2}e_{12})e_{20}(\cos\tfrac{\theta}{2} + \sin\tfrac{\theta}{2}e_{12}) \\
&= \cos^2\tfrac{\theta}{2}e_{20} + \cos\tfrac{\theta}{2}\sin\tfrac{\theta}{2}e_{20}e_{12} - \sin\tfrac{\theta}{2}\cos\tfrac{\theta}{2}e_{12}e_{20} - \sin^2\tfrac{\theta}{2}e_{12}e_{20}e_{12} \\
&= \sin\theta e_{01} + \cos\theta e_{20} \\
r[e_{12}] &= (\cos\tfrac{\theta}{2} - \sin\tfrac{\theta}{2}e_{12})e_{12}(\cos\tfrac{\theta}{2} + \sin\tfrac{\theta}{2}e_{12}) \\
&= \cos^2\tfrac{\theta}{2}e_{12} + \cos\tfrac{\theta}{2}\sin\tfrac{\theta}{2}e_{12}e_{12} - \sin\tfrac{\theta}{2}\cos\tfrac{\theta}{2}e_{12}e_{12} - \sin^2\tfrac{\theta}{2}e_{12}e_{12}e_{12} \\
&= e_{12} \\
r[e_{012}] &= (\cos\tfrac{\theta}{2} - \sin\tfrac{\theta}{2}e_{12})e_{012}(\cos\tfrac{\theta}{2} + \sin\tfrac{\theta}{2}e_{12}) \\
&= \cos^2\tfrac{\theta}{2}e_{012} + \cos\tfrac{\theta}{2}\sin\tfrac{\theta}{2}e_{012}e_{12} - \sin\tfrac{\theta}{2}\cos\tfrac{\theta}{2}e_{12}e_{012} - \sin^2\tfrac{\theta}{2}e_{12}e_{012}e_{12} \\
&= e_{012}
\end{aligned}
$$

Using the above, we can now verify that

1. A translation by $(a, b) \in \mathbb{R}^2$ of the point $p = (x, y)$ yields the point $(x', y') = (x + a, y + b)$:

$$
\begin{aligned}
t[p] &= t[xe_{20} + ye_{01} + e_{12}] \\
&= x \cdot t[e_{20}] + y \cdot t[e_{01}] + t[e_{12}] \\
&= xe_{20} + ye_{01} + (e_{12} + be_{01} + ae_{20}) \\
&= x'e_{20} + y'e_{01} + e_{12}
\end{aligned}
$$

2. A translation by $(a, b) \in \mathbb{R}^2$ of the line $\ell : Ax+By+C = 0$ yields the line $Ax+By+C' = 0$, where $C' = C - Aa - Bb$:

$$\begin{aligned} t[\ell] &= t[Ae_1 + Be_2 + Ce_0] \\ &= A \cdot t[e_1] + B \cdot t[e_2] + C \cdot t[e_0] \\ &= A(e_1 - ae_0) + B(e_2 - be_0) + Ce_0 \\ &= Ae_1 + Be_2 + C'e_0 \end{aligned}$$

3. A rotation by angle $\theta \in \mathbb{R}$ of the point $p = (x, y)$ yields the point $(x', y') = (x \cos\theta - y \sin\theta, x \sin\theta + y \cos\theta)$:

$$\begin{aligned} r[p] &= r[xe_{20} + ye_{01} + e_{12}] \\ &= x \cdot r[e_{20}] + y \cdot r[e_{01}] + r[e_{12}] \\ &= x(\sin\theta e_{01} + \cos\theta e_{20}) + y(\cos\theta e_{01} - \sin\theta e_{20}) + e_{12} \\ &= x'e_{20} + y'e_{01} + e_{12} \end{aligned}$$

4. A rotation by angle $\theta \in \mathbb{R}$ of the line $\ell : Ax + By + C = 0$ yields the line $A'x + B'y + C = 0$, where $(A', B') = (A \cos\theta - B \sin\theta, A \sin\theta + B \cos\theta)$:

$$\begin{aligned} r[\ell] &= r[Ae_1 + Be_2 + Ce_0] \\ &= A \cdot r[e_1] + B \cdot r[e_2] + C \cdot r[e_0] \\ &= A(\cos\theta e_1 + \sin\theta e_2) + B(\cos\theta e_2 - \sin\theta e_1) + Ce_0 \\ &= A'e_1 + B'e_2 + Ce_0 \end{aligned}$$

This concludes the proof. □

Proposition 2. *The intersection of two lines is given by a wedge product.*

Proof. For lines $a_1x + b_1y + c_1 = 0$ and $a_2x + b_2y + c_2 = 0$, their intersection point $x_0 = \dfrac{b_1c_2 - b_2c_1}{a_1b_2 - a_2b_1}$, $y_0 = \dfrac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1}$ can be computed via

$$\begin{aligned} \ell_1 \wedge \ell_2 &= (a_1e_1 + b_1e_2 + c_1e_0) \wedge (a_2e_1 + b_2e_2 + c_2e_0) \\ &= (a_2c_1 - a_1c_2)e_{01} + (b_1c_2 - b_2c_1)e_{20} + (a_1b_2 - a_2b_1)e_{12} \\ &= (a_1b_2 - a_2b_1)(x_0e_{20} + y_0e_{01} + e_{12}) \end{aligned}$$

□

Proposition 3. *The line joining two points is given by a join operator.*

Proof. If we join the points (a, b) and (c, d), we obtain

$$\begin{aligned} &((ae_{20} + be_{01} + e_{12})^* \wedge (ce_{20} + de_{01} + e_{12})^*)^* \\ &= ((ae_1 + be_2 + e_0) \wedge (ce_1 + de_2 + e_0))^* \\ &= ((c - a)e_{01} + (b - d)e_{20} + (ad - bc)e_{12})^* \\ &= (c - a)e_2 + (b - d)e_1 + (ad - bc)e_0 \end{aligned}$$

which is the encoding of the line $(b - d)x + (c - a)y + (ad - bc) = 0$ passing through both points. □

Proposition 4. *The distance from a point to a line is given by a join operator.*

Proof. If we join the point (x_0, y_0) and the line $ax + by + c = 0$ where $\sqrt{a^2 + b^2} = 1$, we obtain

$$\begin{aligned} &((x_0e_{20} + y_0e_{01} + e_{12})^* \wedge (ae_1 + be_2 + ce_0)^*)^* \\ &= ((x_0e_1 + y_0e_2 + e_0) \wedge (ae_{20} + be_{01} + ce_{12}))^* \\ &= ((ax_0 + by_0 + c)e_{012})^* \\ &= ax_0 + by_0 + c \end{aligned}$$

which is the signed distance between them. □

1.3. Equivariant layers

To show that a function $f : \mathbb{R}^*_{2,0,1} \to \mathbb{R}^*_{2,0,1}$ is SE(2)-equivariant, one can verify that for any rotation or translation operator u and input multivector x, we have $f(u[x]) = u[f(x)]$.

We define linear maps $\phi : \mathbb{R}^*_{2,0,1} \to \mathbb{R}^*_{2,0,1}$ in the form

$$\phi(x) = \sum_{k=0}^{3} w_k \langle x \rangle_k + \sum_{k=0}^{2} v_k e_0 \langle x \rangle_k + \sum_{k=0}^{2} u_k e_{012} \langle x \rangle_k \tag{1}$$

Note that this differs from the linear layer defined in [1] as we have introduced additional parameters u_k. With these extra terms, ϕ is not equivariant to reflections.

This is how ϕ acts on the basis multivectors:

$$\begin{aligned}
\phi(1) &= w_0 + v_0 e_0 + u_0 e_{012} \\
\phi(e_0) &= w_1 e_0 \\
\phi(e_1) &= w_1 e_1 + v_1 e_{01} + u_1 e_{20} \\
\phi(e_2) &= w_1 e_2 - v_1 e_{20} + u_1 e_{01} \\
\phi(e_{01}) &= w_2 e_{01} \\
\phi(e_{20}) &= w_2 e_{20} \\
\phi(e_{12}) &= w_2 e_{12} + v_2 e_{012} - u_2 e_0 \\
\phi(e_{012}) &= w_3 e_{012}
\end{aligned}$$

Proposition 5. *The linear map defined in* equation (1) *is SE(2)-equivariant.*

Proof. Let $t = 1 - \frac{a}{2} e_{01} + \frac{b}{2} e_{20}$ be a translation operator and $r = \cos\frac{\theta}{2} - \sin\frac{\theta}{2} e_{12}$ be a rotation operator. Since ϕ and $u[\cdot]$ are linear, it suffices to show that $t[\phi(x)] = \phi(t[x])$ and $r[\phi(x)] = \phi(r[x])$ for the basis multivectors $x \in \{1, e_0, e_1, e_2, e_{01}, e_{20}, e_{12}, e_{012}\}$.

Referring to the proof of Proposition 1 for values of $t[x]$ and $r[x]$, we can verify these equations hold for translations:

$$\begin{aligned}
\phi(t[1]) &= \phi(1) = w_0 + v_0 e_0 + u_0 e_{012} \\
t[\phi(1)] &= t[w_0 + v_0 e_0 + u_0 e_{012}] = w_0 + v_0 e_0 + u_0 e_{012}
\end{aligned}$$

$$\begin{aligned}
\phi(t[e_0]) &= \phi(e_0) = w_1 e_0 \\
t[\phi(e_0)] &= t[w_1 e_0] = w_1 e_0
\end{aligned}$$

$$\begin{aligned}
\phi(t[e_1]) &= \phi(e_1 - a e_0) = (w_1 e_1 + v_1 e_{01} + u_1 e_{20}) - a w_1 e_0 \\
t[\phi(e_1)] &= t[w_1 e_1 + v_1 e_{01} + u_1 e_{20}] = w_1(e_1 - a e_0) + v_1 e_{01} + u_1 e_{20}
\end{aligned}$$

$$\begin{aligned}
\phi(t[e_2]) &= \phi(e_2 - b e_0) = (w_1 e_2 - v_1 e_{20} + u_1 e_{01}) - b w_1 e_0 \\
t[\phi(e_2)] &= t[w_1 e_2 - v_1 e_{20} + u_1 e_{01}] = w_1(e_2 - b e_0) - v_1 e_{20} + u_1 e_{01}
\end{aligned}$$

$$\begin{aligned}
\phi(t[e_{01}]) &= \phi(e_{01}) = w_2 e_{01} \\
t[\phi(e_{01})] &= t[w_2 e_{01}] = w_2 e_{01}
\end{aligned}$$

$$\begin{aligned}
\phi(t[e_{20}]) &= \phi(e_{20}) = w_2 e_{20} \\
t[\phi(e_{20})] &= t[w_2 e_{20}] = w_2 e_{20}
\end{aligned}$$

$$\begin{aligned}
\phi(t[e_{12}]) &= \phi(e_{12} + b e_{01} + a e_{20}) = (w_2 e_{12} + v_2 e_{012} - u_2 e_0) + b w_2 e_{01} + a w_2 e_{20} \\
t[\phi(e_{12})] &= t[w_2 e_{12} + v_2 e_{012} - u_2 e_0] = w_2(e_{12} + b e_{01} + a e_{20}) + v_2 e_{012} - u_2 e_0
\end{aligned}$$

$$\begin{aligned}
\phi(t[e_{012}]) &= \phi(e_{012}) = w_3 e_{012} \\
t[\phi(e_{012})] &= t[w_3 e_{012}] = w_3 e_{012}
\end{aligned}$$

and rotations:

$$\begin{aligned}
\phi(r[1]) &= \phi(1) = w_0 + v_0 e_0 + u_0 e_{012} \\
r[\phi(1)] &= r[w_0 + v_0 e_0 + u_0 e_{012}] = w_0 + v_0 e_0 + u_0 e_{012}
\end{aligned}$$

$$\begin{aligned}
\phi(r[e_0]) &= \phi(e_0) = w_1 e_0 \\
r[\phi(e_0)] &= r[w_1 e_0] = w_1 e_0
\end{aligned}$$

$$\begin{aligned}
\phi(r[e_1]) &= \phi(\cos\theta e_1 + \sin\theta e_2) = \cos\theta(w_1 e_1 + v_1 e_{01} + u_1 e_{20}) + \sin\theta(w_1 e_2 - v_1 e_{20} + u_1 e_{01}) \\
r[\phi(e_1)] &= r[w_1 e_1 + v_1 e_{01} + u_1 e_{20}] \\
&= w_1(\cos\theta e_1 + \sin\theta e_2) + v_1(\cos\theta e_{01} - \sin\theta e_{20}) + u_1(\sin\theta e_{01} + \cos\theta e_{20})
\end{aligned}$$

$$\begin{aligned}
\phi(r[e_2]) &= \phi(\cos\theta e_2 - \sin\theta e_1) = \cos\theta(w_1 e_2 - v_1 e_{20} + u_1 e_{01}) - \sin\theta(w_1 e_1 + v_1 e_{01} + u_1 e_{20}) \\
r[\phi(e_2)] &= r[w_1 e_2 - v_1 e_{20} + u_1 e_{01}] \\
&= w_1(\cos\theta e_2 - \sin\theta e_1) - v_1(\sin\theta e_{01} + \cos\theta e_{20}) + u_1(\cos\theta e_{01} - \sin\theta e_{20})
\end{aligned}$$

$$\begin{aligned}
\phi(r[e_{01}]) &= \phi(\cos\theta e_{01} - \sin\theta e_{20}) = w_2 \cos\theta e_{01} - w_2 \sin\theta e_{20} \\
r[\phi(e_{01})] &= r[w_2 e_{01}] = w_2(\cos\theta e_{01} - \sin\theta e_{20})
\end{aligned}$$

$$\begin{aligned}
\phi(r[e_{20}]) &= \phi(\sin\theta e_{01} + \cos\theta e_{20}) = w_2 \sin\theta e_{01} + w_2 \cos\theta e_{20} \\
r[\phi(e_{20})] &= r[w_2 e_{20}] = w_2(\sin\theta e_{01} + \cos\theta e_{20})
\end{aligned}$$

$$\begin{aligned}
\phi(r[e_{12}]) &= \phi(e_{12}) = w_2 e_{12} + v_2 e_{012} - u_2 e_0 \\
r[\phi(e_{12})] &= r[w_2 e_{12} + v_2 e_{012} - u_2 e_0] = w_2 e_{12} + v_2 e_{012} - u_2 e_0
\end{aligned}$$

$$\begin{aligned}
\phi(r[e_{012}]) &= \phi(e_{012}) = w_3 e_{012} \\
r[\phi(e_{012})] &= r[w_3 e_{012}] = w_3 e_{012}
\end{aligned}$$

□

Proposition 6. *The geometric bilinear layer Geometric(w, x, y, z) = Concatenate$(wx$, Join$(y, z))$ is SE(2)-equivariant.*

Proof. Geometric product is equivariant as $(uwu^{-1})(uxu^{-1}) = u(wx)u^{-1}$ for any w, x and operator u. That join is equivariant is proven in [1]. Alternatively, one can use linearity and check that $\text{Join}(uyu^{-1}, uzu^{-1}) = u \cdot \text{Join}(y, z) \cdot u^{-1}$ for any operator u and *basis* multivectors y, z. □

Lemma 1. *If $f : \mathbb{R}^*_{2,0,1} \to \mathbb{R}^*_{2,0,1}$ is equivariant and $\rho : \mathbb{R}^*_{2,0,1} \to \mathbb{R}$ is invariant, then $g(x) := \rho(x)f(x)$ is equivariant.*

Proof. For any operator u and multivector x,

$$\begin{aligned}
g(uxu^{-1}) &= \rho(uxu^{-1}) f(uxu^{-1}) \\
&= \rho(x) \cdot u f(x) u^{-1} \\
&= u g(x) u^{-1}
\end{aligned}$$

□

Proposition 7. *The scalar-gated activation GatedRELU(x) = RELU$(\langle x \rangle_0)x$ is SE(2)-equivariant.*

Proof. By Lemma 1 it suffices to show that $\langle x \rangle_0$ is SE(2)-invariant. Write

$$x = x' + x_0 e_0 + x_1 e_1 + x_2 e_2 + x_{01} e_{01} + x_{20} e_{20} + x_{12} e_{12} + x_{012} e_{012}$$

If $u = 1 - \frac{a}{2} e_{01} + \frac{b}{2} e_{20}$ is a translation, we can compute

$$\begin{aligned}
u[x] = x' &+ (x_0 - a x_1 - b x_2) e_0 + x_1 e_1 + x_2 e_2 \\
&+ (x_{01} + b x_{12}) e_{01} + (x_{20} + a x_{12}) e_{20} + x_{12} e_{12} + x_{012} e_{012} \qquad (2)
\end{aligned}$$

hence $\langle u[x]\rangle_0 = x' = \langle x\rangle_0$ so $\langle x\rangle_0$ is translation-invariant. Similarly, if $u = \cos\frac{\theta}{2} - \sin\frac{\theta}{2}e_{12}$ is a rotation,

$$\begin{aligned} u[x] = x' + x_0e_0 + (x_1\cos\theta - x_2\sin\theta)e_1 + (x_1\sin\theta + x_2\cos\theta)e_2 \\ + (x_{01}\cos\theta + x_{20}\sin\theta)e_{01} + (x_{20}\cos\theta - x_{01}\sin\theta)e_{20} + x_{12}e_{12} + x_{012}e_{012} \end{aligned} \tag{3}$$

hence $\langle x\rangle_0$ is rotation-invariant. □

Lemma 2. *The inner product* $\langle x, y\rangle := x'y' + x_1y_1 + x_2y_2 + x_{12}y_{12}$ *is SE(2)-invariant.*

Proof. If u is a translation, by equation (2)

$$\langle u[x], u[y]\rangle = x'y' + x_1y_1 + x_2y_2 + x_{12}y_{12} = \langle x, y\rangle$$

If u is a rotation, by equation (3)

$$\begin{aligned} \langle u[x], u[y]\rangle &= x'y' + (x_1\cos\theta - x_2\sin\theta)(y_1\cos\theta - y_2\sin\theta) \\ &\quad + (x_1\sin\theta + x_2\cos\theta)(y_1\sin\theta + y_2\cos\theta) + x_{12}y_{12} \\ &= x'y' + x_1y_1 + x_2y_2 + x_{12}y_{12} \\ &= \langle x, y\rangle \end{aligned}$$

□

Proposition 8. *The normalization layer LayerNorm*$(x) = x/\sqrt{\mathbb{E}\langle x, x\rangle + \varepsilon}$ *is SE(2)-equivariant.*

Proof. This follows from Lemma 1 and Lemma 2. □

Proposition 9. *Multivector attention is SE(2)-equivariant.*

Proof. By Lemma 2, attention logits

$$\frac{\sum_{c=1}^{C}\langle q_c, k_c\rangle}{\sqrt{4C}}$$

(and hence attention scores) are invariant. The result follows from applying Lemma 1 and observing that a sum of equivariant multivector features is equivariant. □

2. Additional qualitative results

Figure 1 provides additional comparisons on models' robustness to roto-translations. Figure 2 shows the overlayed multi-modal trajectories of DriveGATr's rollouts.

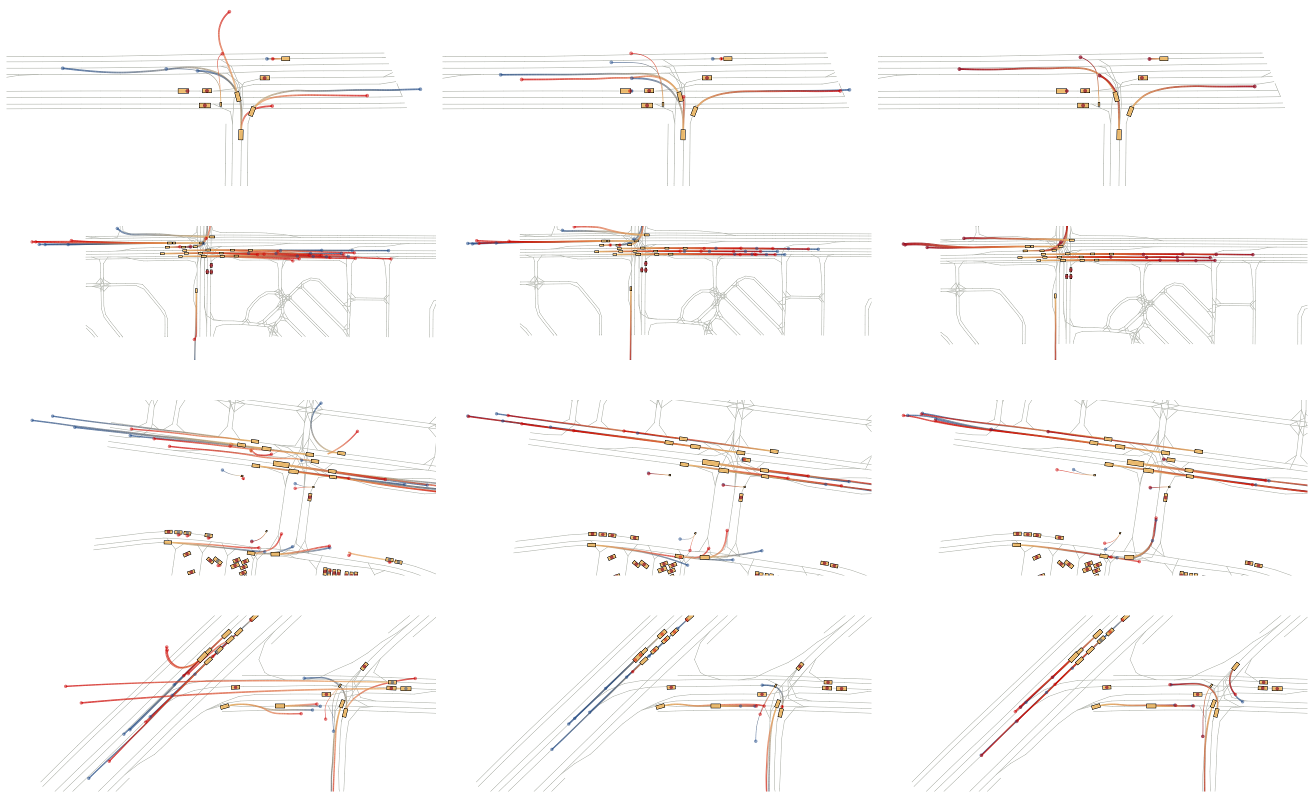

Figure 1. **Robustness to roto-translations.** Additional visualizations showing Transformer (**left**), Transformer + DRoPE (**middle**), and DriveGATr (**right**)'s robustness to roto-translations. In each figure, we overlay rollouts from the original coordinate frame vs one rotated by 90°and translated by 100m forward. Blue trajectories visualize model predictions in the original input, and red visualize predictions in the transformed scene.

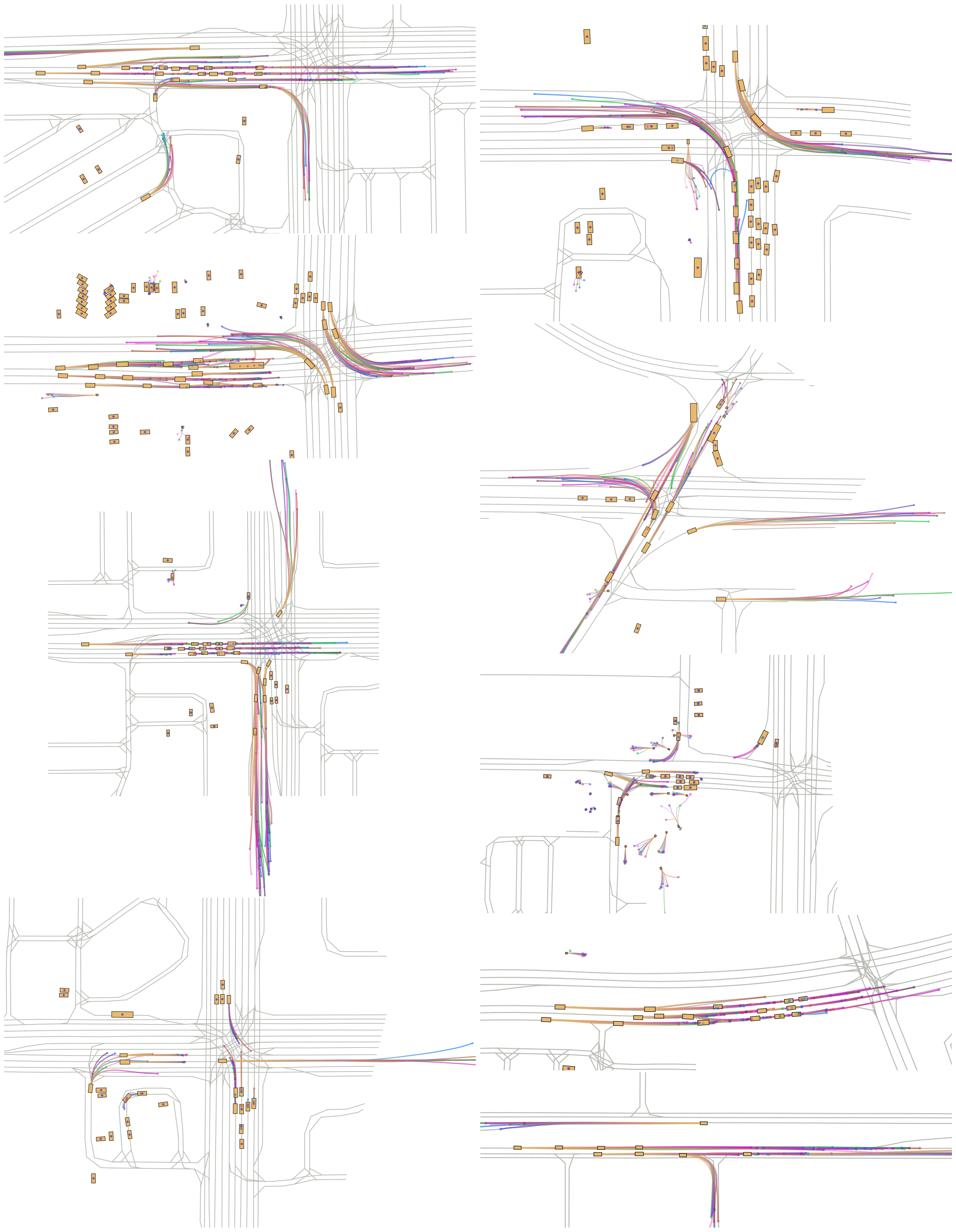

Figure 2. Visualizations of **multi-modal** DriveGATr rollouts. In each figure, we overlay trajectories from multiple ($k = 8$) rollouts. Different actors of the same color corresponds to the same rollout.

3. Additional details

3.1. Architecture details

In this section, we provide algorithmic details of the DriveGATr architecture. Multivector features are assumed to have shape [batch $\times$ num_tokens $\times$ mv_channels $\times$ 8] where the last axis corresponds to the 8 components of each multivector. Scalar features have shape [batch $\times$ num_tokens $\times$ channels].

Equivariant MLP. Throughout the model, when multivector features pass through an equivariant layer, scalar features pass through the standard version of that layer. Our equivariant MLP is analagous to a standard MLP, with the addition of the geometric bilinears.

```
function GEOMETRICBILINEAR(w, x, y, z)
    gp ← batch_geometric_product(w, x)
    join ← dual(batch_wedge_product(dual(y), dual(z)))

    // [batch × num_tokens × (2 * mv_channels) × 8]
    return concatenate_channels(gp, join)
end function

function EQUIVARIANTMLP(multivector_features, scalar_features)
    // normalization
    mv_features ← EquivariantLayerNorm(multivector_features)
    features ← LayerNorm(scalar_features)

    // geometric bilinears
    mv_features ← EquivariantLinear(mv_features)
    w, x, y, z ← mv_features.split(num_chunks = 4, dim = -2)
    mv_features ← GeometricBilinear(w, x, y, z)

    // standard MLP
    mv_features ← EquivariantLinear(mv_features)
    features ← Linear(features)
    mv_features ← GatedRELU(mv_features)
    features ← RELU(features)
    mv_features ← EquivariantLinear(mv_features)
    features ← Linear(features)

    // residual connection
    return mv_features + multivector_features, features + scalar_features
end function
```

Attention. We show how to compute multivector attention, where the logits are

$$\frac{\sum_{c=1}^{C}\langle q_c, k_c\rangle + \sum_{c=1}^{C'} q_c^s k_c^s}{\sqrt{4C + C'}}.$$

Observe that a sum of dot products can be equivalently expressed as a longer dot product, e.g. for $u_1, v_1, u_2, v_2 \in \mathbb{R}^n$, $u_1 \cdot v_1 + u_2 \cdot v_2 = \mathbf{u} \cdot \mathbf{v}$ where $\mathbf{u} = u_1 | u_2, \mathbf{v} = v_1 | v_2 \in \mathbb{R}^{2n}$ are concatenations. Therefore, we can use standard dot-product attention after concatenating the appropriate multivector components with the invariant scalars along the embedding dimension. To incorporate distance awareness, one just needs to further concatenate the components of the distance awareness maps $\phi(\cdot)$ and $\psi(\cdot)$ to the queries and keys, respectively.

```
function EQUIVARIANTATTENTION(mv_query, mv_key, mv_value, query, key, value, mask)
    // extract components for invariant inner product $\langle x, y\rangle = x_s y_s + x_1 y_1 + x_2 y_2 + x_{12} y_{12}$
    $q_s, \_, q_1, q_2, \_, \_, q_{12}, \_$ ← mv_query.split(dim = -1)
    $k_s, \_, k_1, k_2, \_, \_, k_{12}, \_ \leftarrow$ mv_key.split(dim = -1)

    // $[batch \times num\_queries \times (4 * mv\_channels + channels)]$
    flattened_q $\leftarrow$ concatenate_channels($q_s, q_1, q_2, q_{12}$, query)

    // $[batch \times num\_keys \times (4 * mv\_channels + channels)]$
    flattened_k $\leftarrow$ concatenate_channels($k_s, k_1, k_2, k_{12}$, key)

    // $[batch \times num\_keys \times (8 * mv\_channels + channels)]$
    flattened_v $\leftarrow$ concatenate_mv_components_and_scalars(mv_value, value)

    // $[batch \times num\_queries \times (8 * mv\_channels + channels)]$
    flattened_output $\leftarrow$ scaled_dot_product_attention(flattened_q, flattened_k, flattened_v, mask)

    // $[batch \times num\_queries \times mv\_channels \times 8]$, $[batch \times num\_queries \times channels]$
    return unconcatenate_mv_components_and_scalars(flattened_output)
end function

function ATTENTION(mv_q_features, q_features, mv_kv_features, kv_features, mask)
    // normalization
    mv_q_features $\leftarrow$ EquivariantLayerNorm(mv_q_features)
    mv_kv_features $\leftarrow$ EquivariantLayerNorm(mv_kv_features)
    q_features $\leftarrow$ LayerNorm(q_features)
    kv_features $\leftarrow$ LayerNorm(kv_features)

    // compute query, key, value tensors
    mv_query $\leftarrow$ EquivariantLinear(mv_q_features)
    mv_key $\leftarrow$ EquivariantLinear(mv_kv_features)
    mv_value $\leftarrow$ EquivariantLinear(mv_kv_features)
    query $\leftarrow$ Linear(q_features)
    key $\leftarrow$ Linear(kv_features)
    value $\leftarrow$ Linear(kv_features)

    mv_features, features $\leftarrow$ EquivariantAttention(mv_query, mv_key, mv_value, query, key, value, mask)

    // residual connection
    return mv_features + mv_q_features, features + q_features
end function
```

DriveGATr. Each DriveGATr block consists of three factorized attention layers followed by an equivariant MLP and an invariant adapter. In the invariant adapter, we compute an operator for each actor which converts global poses into that agent's coordinate frame. Specifically, if an actor has position (x, y) and heading θ, then the operator for that actor consists of a translation by $(-x, -y)$ followed by a rotation of angle $-\theta$.

```
function DRIVEGATR(mv_features, features, mv_map_features, map_features)
    for $i \leftarrow 1 \dots N$ do
        // actor-to-map cross-attention
        mv_features, features $\leftarrow$ Attention(mv_features, features, mv_map_features, map_features)

        // actor-to-actor self-attention (batched across timesteps)
        mv_features, features $\leftarrow$ Attention(mv_features, features, mv_features, features)

        // actor-to-time causal self-attention (batched across actors)
        mv_features, features $\leftarrow$ Attention(mv_features, features, mv_features, features, CAUSAL_MASK)
```

```
11:
12:         // MLP
13:         mv_features, features ← EquivariantMLP(mv_features, features)
14:
15:         // invariant adapter
16:         $u$ ← operator for each actor // [batch × num_actors × 8], duplicated across channels
17:         features ← MLP(flatten_components($u *$ mv_features $* u^{-1}$)) + features
18:     end for
19:
20:     // output invariant features only
21:     return features
22: end function
```

3.2. Baselines

We provide implementation details for the Transformer, Transformer + DRoPE, and Transformer + RPE baselines. All three baselines are edited from the original DriveGATr architecture to minimalize differences other than the ablated module. In particular, we first set the multi-vector dimension to 0, effectively make DriveGATr a standard transformer without positional encodings. We next add in changes specific to each baseline, detailed below.

3.3. Transformer

In addition the invariant features of each agent, we additonally include agent poses (x, y, θ) in the global coordinate frame to the inputs of the initial MLP that encodes the auxiliary scalars.

3.4. Transformer + DRoPE

At the time of submission, there is no publicly released codes nor implementation details of DRoPE Zhao et al. [4]. We hence faithfully reproduce it in our setup. In particular, we use the "intra-head" formulation, splitting the positional encodings of x, y and rotational θ separately by attention heads. For the 2D translational part of the rotary encodings, we use a learnable, mixed frequency setup Heo et al. [3]. Notably, we found it neccessary to uplift the dimention of query and key by a factor of 4 before attention – otherwise, each head only has 4 dimensions to be rotary-encoded which is too small.

3.5. Transformer + RPE

At each attention block, we additionally use an MLP to model the pairwise differences. We use the GoRELA Cui et al. [2] encoder to encode the pairwise pose features, and broad-cast add them to the keys and values before applying attention:

$$k^{ij}_{rpe}, v^{ij}_{rpe} = \text{GoRELA}(x^i, y^i, \theta^i, x^j, y^j, \theta^j) \tag{4}$$

$$k \leftarrow k + k_{rpe} \tag{5}$$

$$v \leftarrow v + v_{rpe} \tag{6}$$

4. Latency

Method	Training latency	Training peak memory	Inference latency	SE(2) equivariant?
Transformer	196.86 ms	3.28 GB	42.64 ms	×
Transformer + RPE	328.70 ms	15.19 GB	82.48 ms	✓
Transformer + DRoPE	195.51 ms	5.08 GB	46.03 ms	×
DriveGATr-3M	264.34 ms	6.35 GB	58.01 ms	✓

Table 1. Peak memory and latency comparisons of the baselines and DriveGATr. All methods in the same group use the same context length. The training latency times the combined forward and backward passes. The inference latency are averaged over the entire 8s rollout. Both training and inference use batch size = 8. All entries are evaluated a single A5000 with fixed random seed.

We use FLOPs as the main compute measurement since FLOPs are hardware agnostic, but provide additional memory footprint and latency comparisons in Table 1. Compared to a fully invariant baseline (Transformer + RPE), DriveGATr is faster and more memory efficient. For a fixed batch size, DriveGATr has lower throughput than the non-SE(2) equivariant baselines. However, we demonstrate that DriveGATr achieves the superior Pareto front in Figure 2 of the main manuscript by fixing the resources, training each model on the same device and maxing out the batch size for each model, which we believe makes a fair and sufficient comparison of Pareto fronts.

References

[1] Johann Brehmer, Pim De Haan, Sönke Behrends, and Taco S Cohen. Geometric algebra transformer. In *NeurIPS*, 2023. 5, 6

[2] Alexander Cui, Sergio Casas, Kelvin Wong, Simon Suo, and Raquel Urtasun. GoRela: Go relative for viewpoint-invariant motion forecasting. In *ICRA*, 2023. 12

[3] Byeongho Heo, Song Park, Dongyoon Han, and Sangdoo Yun. Rotary position embedding for vision transformer. In *ECCV*, 2024. 12

[4] Jianbo Zhao, Taiyu Ban, Zhihao Liu, Hangning Zhou, Xiyang Wang, Qibin Zhou, Hailong Qin, Mu Yang, Lei Liu, and Bin Li. Drope: Directional rotary position embedding for efficient agent interaction modeling. *arXiv*, 2025. 12