JAX - GPU accelerated linear algebra & Wigner's Batman

Hrvoje Abraham

JAX by Google

https://docs.jax.dev/en/latest/index.html

Originally, it stood for "Just After eXecution"

Transformable numerical computing at scale

JAX is a Python library for accelerator-oriented array computation and program transformation, designed for high-performance numerical computing and large-scale machine learning.

JAX by Google

Type '/' to search projects

Help Docs Sponsors Log in Register

jax 0.8.1

```
pip install jax
```

Latest version

Released: Nov 18, 2025

Differentiate, compile, and transform Numpy code.

```python
import jax
import jax.numpy as jnp

def predict(params, inputs):
  for W, b in params:
    outputs = jnp.dot(inputs, W) + b
    inputs = jnp.tanh(outputs)  # inputs to the next layer
  return outputs                # no activation on last layer

def loss(params, inputs, targets):
  preds = predict(params, inputs)
  return jnp.sum((preds - targets)**2)

grad_loss = jax.jit(jax.grad(loss))  # compiled gradient evaluation function
perex_grads = jax.jit(jax.vmap(grad_loss, in_axes=(None, 0, 0)))  # fast per-example grads
```

https://pypi.org/project/jax/

https://github.com/jax-ml/jax

JAX by Google

Hardware	Backend	Support Status
NVIDIA GPUs	CUDA	First-class, official support.
Google TPUs	TPU	First-class, official support.
AMD GPUs	ROCm	Official support (Linux only).
Apple Silicon	Metal	Experimental (via `jax-metal` plugin).
CPU	Host	Fallback for all systems.

No direct Vulkan support.

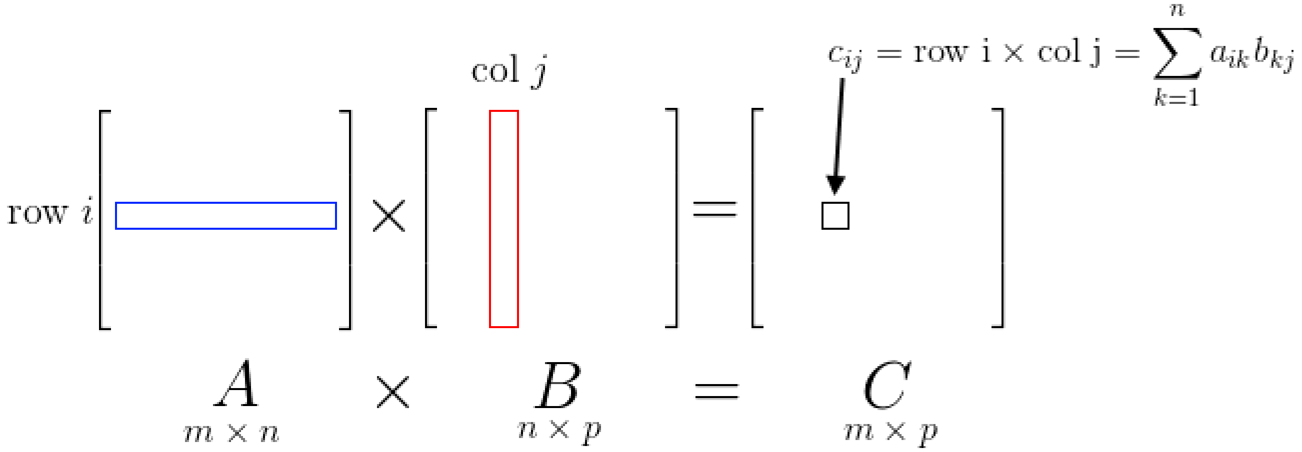
JAX by Google

	Linux, x86_64	Linux, aarch64	Mac, aarch64	Windows, x86_64	Windows WSL2, x86_64
CPU	yes	yes	yes	yes	yes
NVIDIA GPU	yes	yes	n/a	no	experimental
Google Cloud TPU	yes	n/a	n/a	n/a	n/a
AMD GPU	yes	no	n/a	no	experimental
Apple GPU	n/a	no	experimental	n/a	n/a
Intel GPU	experimental	n/a	n/a	no	no

Example 1 – Matrix multiplication

Matrix 1 × Matrix 2 = Product matrix

$$\underset{(3\text{x}2)}{\begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}} \times \underset{(2\text{x}3)}{\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix}} = \underset{(3\text{x}3)}{\begin{bmatrix} 3 & 3 & 3 \\ 6 & 6 & 6 \\ 9 & 9 & 9 \end{bmatrix}}$$



Example 1 – Matrix multiplication

$$
\begin{bmatrix}
1 & 0 & -\frac{1}{4} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & -\frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & -\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{1}{4} & 0 & 1 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -\frac{1}{4} & 1 & -\frac{1}{4} & 0 & -\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -\frac{1}{4} & 1 & -\frac{1}{4} & 0 & -\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{1}{4} & 0 & 0 & -\frac{1}{4} & 1 & 0 & 0 & -\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{1}{4} & 0 & 0 & -\frac{1}{4} & 0 & 0 & 1 & -\frac{1}{4} & 0 & 0 & -\frac{1}{4} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -\frac{1}{4} & 0 & -\frac{1}{4} & 1 & -\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -\frac{1}{4} & 0 & -\frac{1}{4} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -\frac{1}{2} & -\frac{1}{4} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{4} & 0 & 0 & -\frac{1}{4} & 1 & 0 & -\frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{4} & 0 & 1 & -\frac{1}{2} & -\frac{1}{4} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{4} & -\frac{1}{4} & 1 & 0 & -\frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{4} & 0 & 1 & -\frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{4} & -\frac{1}{4} & 1
\end{bmatrix}
$$

2202	2229	2256	2283	2310	2337	2364	2391	2418	2445	2472	2499	1	28	55	82	109	136	163	190	217	244	271	298	325	1577	1604	1631	1658	1685	1712	1739	1766	1793	1820	1847	1874	1251	1278	680	707	734	761	788	815	842	869	896	923	950
2228	2255	2282	2309	2336	2363	2390	2417	2444	2471	2498	1900	27	54	81	108	135	162	189	216	243	270	297	324	326	1603	1630	1657	1684	1711	1738	1765	1792	1819	1846	1873	1275	1277	1304	706	733	760	787	814	841	868	895	922	949	951
2254	2281	2308	2335	2362	2389	2416	2443	2470	2497	1899	1901	53	80	107	134	161	188	215	242	269	296	323	350	352	1629	1656	1683	1710	1737	1764	1791	1818	1845	1872	1274	1276	1303	1330	732	759	786	813	840	867	894	921	948	975	977
2280	2307	2334	2361	2388	2415	2442	2469	2496	1898	1925	1927	79	106	133	160	187	214	241	268	295	322	349	351	378	1655	1682	1709	1736	1763	1790	1817	1844	1871	1273	1300	1302	1329	1356	758	785	812	839	866	893	920	947	974	976	1003
2306	2333	2360	2387	2414	2441	2468	2495	1897	1924	1926	1953	105	132	159	186	213	240	267	294	321	348	375	377	404	1681	1708	1735	1762	1789	1816	1843	1870	1272	1299	1301	1328	1355	1382	784	811	838	865	892	919	946	973	1000	1002	1029
2332	2359	2386	2413	2440	2467	2494	1896	1923	1950	1952	1979	131	158	185	212	239	266	293	320	347	374	376	403	430	1707	1734	1761	1788	1815	1842	1869	1271	1298	1325	1327	1354	1381	1408	810	837	864	891	918	945	972	999	1001	1028	1055
2358	2385	2412	2439	2466	2493	1895	1922	1949	1951	1978	2005	157	184	211	238	265	292	319	346	373	400	402	429	456	1733	1760	1787	1814	1841	1868	1270	1297	1324	1326	1353	1380	1407	1434	836	863	890	917	944	971	998	1025	1027	1054	1081
2384	2411	2438	2465	2492	1894	1921	1948	1975	1977	2004	2031	183	210	237	264	291	318	345	372	399	401	428	455	482	1759	1786	1813	1840	1867	1269	1296	1323	1350	1352	1379	1406	1433	1460	862	889	916	943	970	997	1024	1026	1053	1080	1107
2410	2437	2464	2491	1893	1920	1947	1974	1976	2003	2030	2057	209	236	263	290	317	344	371	398	425	427	454	481	508	1785	1812	1839	1866	1268	1295	1322	1349	1351	1378	1405	1432	1459	1486	888	915	942	969	996	1023	1050	1052	1079	1106	1133
2436	2463	2490	1892	1919	1946	1973	2000	2002	2029	2056	2083	235	262	289	316	343	370	397	424	426	453	480	507	534	1811	1838	1865	1267	1294	1321	1348	1375	1377	1404	1431	1458	1485	1512	914	941	968	995	1022	1049	1051	1078	1105	1132	1159
2462	2489	1891	1918	1945	1972	1999	2001	2028	2055	2082	2109	261	288	315	342	369	396	423	450	452	479	506	533	560	1837	1864	1266	1293	1320	1347	1374	1376	1403	1430	1457	1484	1511	1538	940	967	994	1021	1048	1075	1077	1104	1131	1158	1185
2488	1890	1917	1944	1971	1998	2025	2027	2054	2081	2108	2135	287	314	341	368	395	422	449	451	478	505	532	559	586	1863	1265	1292	1319	1346	1373	1400	1402	1429	1456	1483	1510	1537	1564	966	993	1020	1047	1074	1076	1103	1130	1157	1184	1211
14	1916	1943	1970	1997	2024	2026	2053	2080	2107	2134	2161	2188	340	367	394	421	448	475	477	504	531	558	585	612	1264	1291	1318	1345	1372	1399	1401	1428	1455	1482	1509	1536	1563	1590	992	1019	1046	1073	1100	1102	1129	1156	1183	1210	1237
1915	1942	1969	1996	2023	2050	2052	2079	2106	2133	2160	2187	339	366	393	420	447	474	476	503	530	557	584	611	13	1290	1317	1344	1371	1398	1425	1427	1454	1481	1508	1535	1562	1589	1616	1018	1045	1072	1099	1101	1128	1155	1182	1209	1236	638
1941	1968	1995	2022	2049	2051	2078	2105	2132	2159	2186	2213	365	392	419	446	473	500	502	529	556	583	610	12	39	1316	1343	1370	1397	1424	1426	1453	1480	1507	1534	1561	1588	1615	1642	1044	1071	1098	1125	1127	1154	1181	1208	1235	637	664
1967	1994	2021	2048	2075	2077	2104	2131	2158	2185	2212	2239	391	418	445	472	499	501	528	555	582	609	11	38	65	1342	1369	1396	1423	1450	1452	1479	1506	1533	1560	1587	1614	1641	1668	1070	1097	1124	1126	1153	1180	1207	1234	636	663	690
1993	2020	2047	2074	2076	2103	2130	2157	2184	2211	2238	2265	417	444	471	498	525	527	554	581	608	10	37	64	91	1368	1395	1422	1449	1451	1478	1505	1532	1559	1586	1613	1640	1667	1694	1096	1123	1150	1152	1179	1206	1233	635	662	689	716
2019	2046	2073	2100	2102	2129	2156	2183	2210	2237	2264	2291	443	470	497	524	526	553	580	607	9	36	63	90	117	1394	1421	1448	1475	1477	1504	1531	1558	1585	1612	1639	1666	1693	1720	1122	1149	1151	1178	1205	1232	634	661	688	715	742
2045	2072	2099	2101	2128	2155	2182	2209	2236	2263	2290	2317	469	496	523	550	552	579	606	8	35	62	89	116	143	1420	1447	1474	1476	1503	1530	1557	1584	1611	1638	1665	1692	1719	1746	1148	1175	1177	1204	1231	633	660	687	714	741	768
2071	2098	2125	2127	2154	2181	2208	2235	2262	2289	2316	2343	495	522	549	551	578	605	7	34	61	88	115	142	169	1446	1473	1500	1502	1529	1556	1583	1610	1637	1664	1691	1718	1745	1772	1174	1176	1203	1230	632	659	686	713	740	767	794
2097	2124	2126	2153	2180	2207	2234	2261	2288	2315	2342	2369	521	548	575	577	604	6	33	60	87	114	141	168	195	1472	1499	1501	1528	1555	1582	1609	1636	1663	1690	1717	1744	1771	1798	1200	1202	1229	631	658	685	712	739	766	793	820
2123	2150	2152	2179	2206	2233	2260	2287	2314	2341	2368	2395	547	574	576	603	5	32	59	86	113	140	167	194	221	1498	1525	1527	1554	1581	1608	1635	1662	1689	1716	1743	1770	1797	1824	1201	1228	630	657	684	711	738	765	792	819	846
2149	2151	2178	2205	2232	2259	2286	2313	2340	2367	2394	2421	573	600	602	4	31	58	85	112	139	166	193	220	247	1524	1526	1553	1580	1607	1634	1661	1688	1715	1742	1769	1796	1823	1850	1227	629	656	683	710	737	764	791	818	845	872
2175	2177	2204	2231	2258	2285	2312	2339	2366	2393	2420	2447	599	601	3	30	57	84	111	138	165	192	219	246	273	1550	1552	1579	1606	1633	1660	1687	1714	1741	1768	1795	1822	1849	1851	628	655	682	709	736	763	790	817	844	871	898
2176	2203	2230	2257	2284	2311	2338	2365	2392	2419	2446	2473	625	2	29	56	83	110	137	164	191	218	245	272	299	1551	1578	1605	1632	1659	1686	1713	1740	1767	1794	1821	1848	1875	1252	654	681	708	735	762	789	816	843	870	897	924
327	354	381	408	435	462	489	516	543	570	597	624	1876	1903	1930	1957	1984	2011	2038	2065	2092	2119	2146	2173	2200	952	979	1006	1033	1060	1087	1114	1141	1168	1195	1222	1249	626	653	1305	1332	1359	1386	1413	1440	1467	1494	1521	1548	1575
353	380	407	434	461	488	515	542	569	596	623	25	1902	1929	1956	1983	2010	2037	2064	2091	2118	2145	2172	2199	2201	978	1005	1032	1059	1086	1113	1140	1167	1194	1221	1248	650	652	679	1331	1358	1385	1412	1439	1466	1493	1520	1547	1574	1576
379	406	433	460	487	514	541	568	595	622	24	26	1928	1955	1982	2009	2036	2063	2090	2117	2144	2171	2198	2225	2227	1004	1031	1058	1085	1112	1139	1166	1193	1220	1247	649	651	678	705	1357	1384	1411	1438	1465	1492	1519	1546	1573	1600	1602
405	432	459	486	513	540	567	594	621	23	50	52	1954	1981	2008	2035	2062	2089	2116	2143	2170	2197	2224	2226	2253	1030	1057	1084	1111	1138	1165	1192	1219	1246	648	675	677	704	731	1383	1410	1437	1464	1491	1518	1545	1572	1599	1601	1628
431	458	485	512	539	566	593	620	22	49	51	78	1980	2007	2034	2061	2088	2115	2142	2169	2196	2223	2250	2252	2279	1056	1083	1110	1137	1164	1191	1218	1245	647	674	676	703	730	757	1409	1436	1463	1490	1517	1544	1571	1598	1625	1627	1654
457	484	511	538	565	592	619	21	48	75	77	104	2006	2033	2060	2087	2114	2141	2168	2195	2222	2249	2251	2278	2305	1082	1109	1136	1163	1190	1217	1244	646	673	700	702	729	756	783	1435	1462	1489	1516	1543	1570	1597	1624	1626	1653	1680
483	510	537	564	591	618	20	47	74	76	103	130	2032	2059	2086	2113	2140	2167	2194	2221	2248	2275	2277	2304	2331	1108	1135	1162	1189	1216	1243	645	672	699	701	728	755	782	809	1461	1488	1515	1542	1569	1596	1623	1650	1652	1679	1706
509	536	563	590	617	19	46	73	100	102	129	156	2058	2085	2112	2139	2166	2193	2220	2247	2274	2276	2303	2330	2357	1134	1161	1188	1215	1242	644	671	698	725	727	754	781	808	835	1487	1514	1541	1568	1595	1622	1649	1651	1678	1705	1732
535	562	589	616	18	45	72	99	101	128	155	182	2084	2111	2138	2165	2192	2219	2246	2273	2300	2302	2329	2356	2383	1160	1187	1214	1241	643	670	697	724	726	753	780	807	834	861	1513	1540	1567	1594	1621	1648	1675	1677	1704	1731	1758
561	588	615	17	44	71	98	125	127	154	181	208	2110	2137	2164	2191	2218	2245	2272	2299	2301	2328	2355	2382	2409	1186	1213	1240	642	669	696	723	750	752	779	806	833	860	887	1539	1566	1593	1620	1647	1674	1676	1703	1730	1757	1784
587	614	16	43	70	97	124	126	153	180	207	234	2136	2163	2190	2217	2244	2271	2298	2325	2327	2354	2381	2408	2435	1212	1239	641	668	695	722	749	751	778	805	832	859	886	913	1565	1592	1619	1646	1673	1700	1702	1729	1756	1783	1810
613	15	42	69	96	123	150	152	179	206	233	260	2162	2189	2216	2243	2270	2297	2324	2326	2353	2380	2407	2434	2461	1238	640	667	694	721	748	775	777	804	831	858	885	912	939	1591	1618	1645	1672	1699	1701	1728	1755	1782	1809	1836
1889	41	68	95	122	149	151	178	205	232	259	286	313	2215	2242	2269	2296	2323	2350	2352	2379	2406	2433	2460	2487	639	666	693	720	747	774	776	803	830	857	884	911	938	965	1617	1644	1671	1698	1725	1727	1754	1781	1808	1835	1862
40	67	94	121	148	175	177	204	231	258	285	312	2214	2241	2268	2295	2322	2349	2351	2378	2405	2432	2459	2486	1888	665	692	719	746	773	800	802	829	856	883	910	937	964	991	1643	1670	1697	1724	1726	1753	1780	1807	1834	1861	1263
66	93	120	147	174	176	203	230	257	284	311	338	2240	2267	2294	2321	2348	2375	2377	2404	2431	2458	2485	1887	1914	691	718	745	772	799	801	828	855	882	909	936	963	990	1017	1669	1696	1723	1750	1752	1779	1806	1833	1860	1262	1289
92	119	146	173	200	202	229	256	283	310	337	364	2266	2293	2320	2347	2374	2376	2403	2430	2457	2484	1886	1913	1940	717	744	771	798	825	827	854	881	908	935	962	989	1016	1043	1695	1722	1749	1751	1778	1805	1832	1859	1261	1288	1315
118	145	172	199	201	228	255	282	309	336	363	390	2292	2319	2346	2373	2400	2402	2429	2456	2483	1885	1912	1939	1966	743	770	797	824	826	853	880	907	934	961	988	1015	1042	1069	1721	1748	1775	1777	1804	1831	1858	1260	1287	1314	1341
144	171	198	225	227	254	281	308	335	362	389	416	2318	2345	2372	2399	2401	2428	2455	2482	1884	1911	1938	1965	1992	769	796	823	850	852	879	906	933	960	987	1014	1041	1068	1095	1747	1774	1776	1803	1830	1857	1259	1286	1313	1340	1367
170	197	224	226	253	280	307	334	361	388	415	442	2344	2371	2398	2425	2427	2454	2481	1883	1910	1937	1964	1991	2018	795	822	849	851	878	905	932	959	986	1013	1040	1067	1094	1121	1773	1800	1802	1829	1856	1258	1285	1312	1339	1366	1393
196	223	250	252	279	306	333	360	387	414	441	468	2370	2397	2424	2426	2453	2480	1882	1909	1936	1963	1990	2017	2044	821	848	875	877	904	931	958	985	1012	1039	1066	1093	1120	1147	1799	1801	1828	1855	1257	1284	1311	1338	1365	1392	1419
222	249	251	278	305	332	359	386	413	440	467	494	2396	2423	2450	2452	2479	1881	1908	1935	1962	1989	2016	2043	2070	847	874	876	903	930	957	984	1011	1038	1065	1092	1119	1146	1173	1825	1827	1854	1256	1283	1310	1337	1364	1391	1418	1445
248	275	277	304	331	358	385	412	439	466	493	520	2422	2449	2451	2478	1880	1907	1934	1961	1988	2015	2042	2069	2096	873	900	902	929	956	983	1010	1037	1064	1091	1118	1145	1172	1199	1826	1853	1255	1282	1309	1336	1363	1390	1417	1444	1471
274	276	303	330	357	384	411	438	465	492	519	546	2448	2475	2477	1879	1906	1933	1960	1987	2014	2041	2068	2095	2122	899	901	928	955	982	1009	1036	1063	1090	1117	1144	1171	1198	1225	1852	1254	1281	1308	1335	1362	1389	1416	1443	1470	1497
300	302	329	356	383	410	437	464	491	518	545	572	2474	2476	1878	1905	1932	1959	1986	2013	2040	2067	2094	2121	2148	925	927	954	981	1008	1035	1062	1089	1116	1143	1170	1197	1224	1226	1253	1280	1307	1334	1361	1388	1415	1442	1469	1496	1523
301	328	355	382	409	436	463	490	517	544	571	598	2500	1877	1904	1931	1958	1985	2012	2039	2066	2093	2120	2147	2174	926	953	980	1007	1034	1061	1088	1115	1142	1169	1196	1223	1250	627	1279	1306	1333	1360	1387	1414	1441	1468	1495	1522	1549

Nvidia RTX 4080 16GB VRAM   Na=30,000 x Nb=30,000 => Nc=30,000

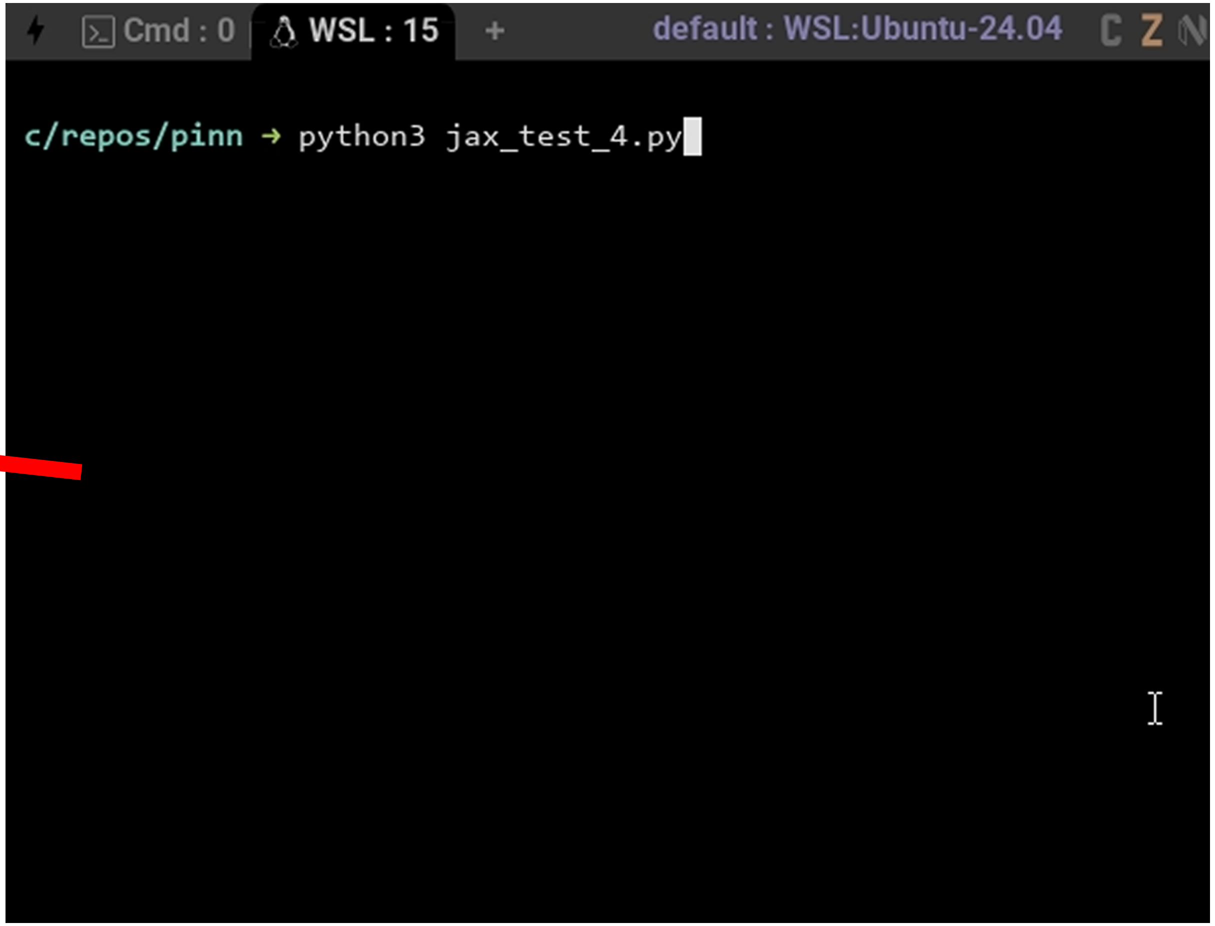
Example 1 – Matrix multiplication

```python
MATRIX_SIZE = 32000
DTYPE = jnp.float32
```

```python
A = jax.random.normal(k1, (N, N), dtype=dtype)
B = jax.random.normal(k2, (N, N), dtype=dtype)
```

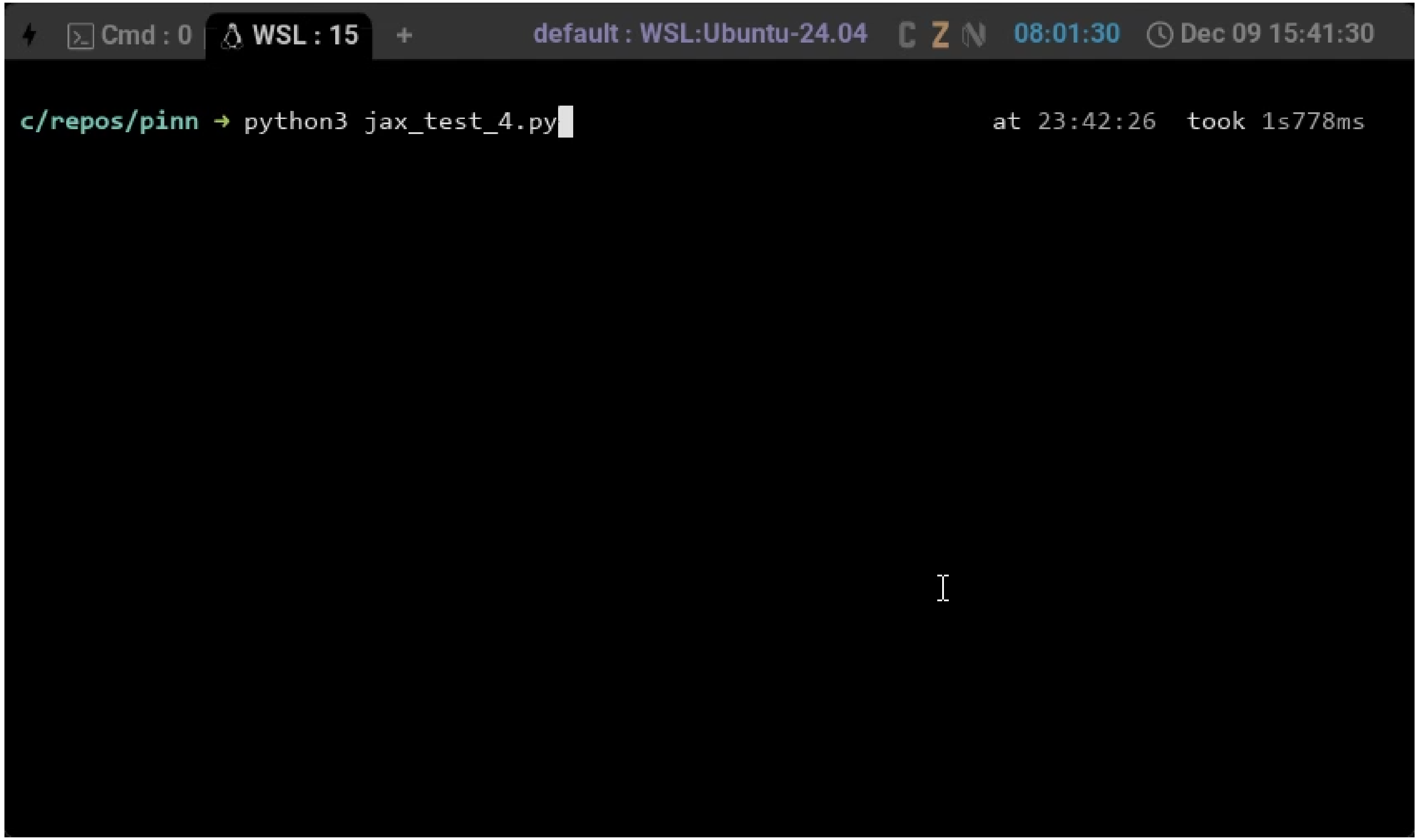
```python
total_sum = jnp.sum(jnp.matmul(A, B))
```

```python
flops = 2 * (N ** 3)
tflops = flops / duration / 1e12

print(f"Total Sum: {total_sum}")
print(f"Execution time: {duration:.4f} s")
print(f"Performance:    {tflops:.2f} TFLOPS")
```



Example 1 – Matrix multiplication

RTX 4080 16GB VRAM   Na=20,000 x Nb=20,000 => Nc=20,000



0.33s @ 48 TFLOPs

16,000 billion operations

Example 1 – Matrix multiplication

Device	Approx. Peak Performance (FP32 TFLOPS)	Time for 50 TFLOPs
NVIDIA RTX 4080 GPU	~48.7 TFLOPS TechPowerUp +1	~1.03 seconds
Intel Core i9-13800H CPU	~1.5–2.0 TFLOPS (estimated from benchmarks NanoReview +1)	~25–33 seconds
HP Notebook (5 years old, midrange)	~0.2–0.3 TFLOPS (typical integrated GPU or older CPU Microsoft Learn +1)	~3–4 minutes

200x speedup

Example 1 – Matrix multiplication – Warmup phase

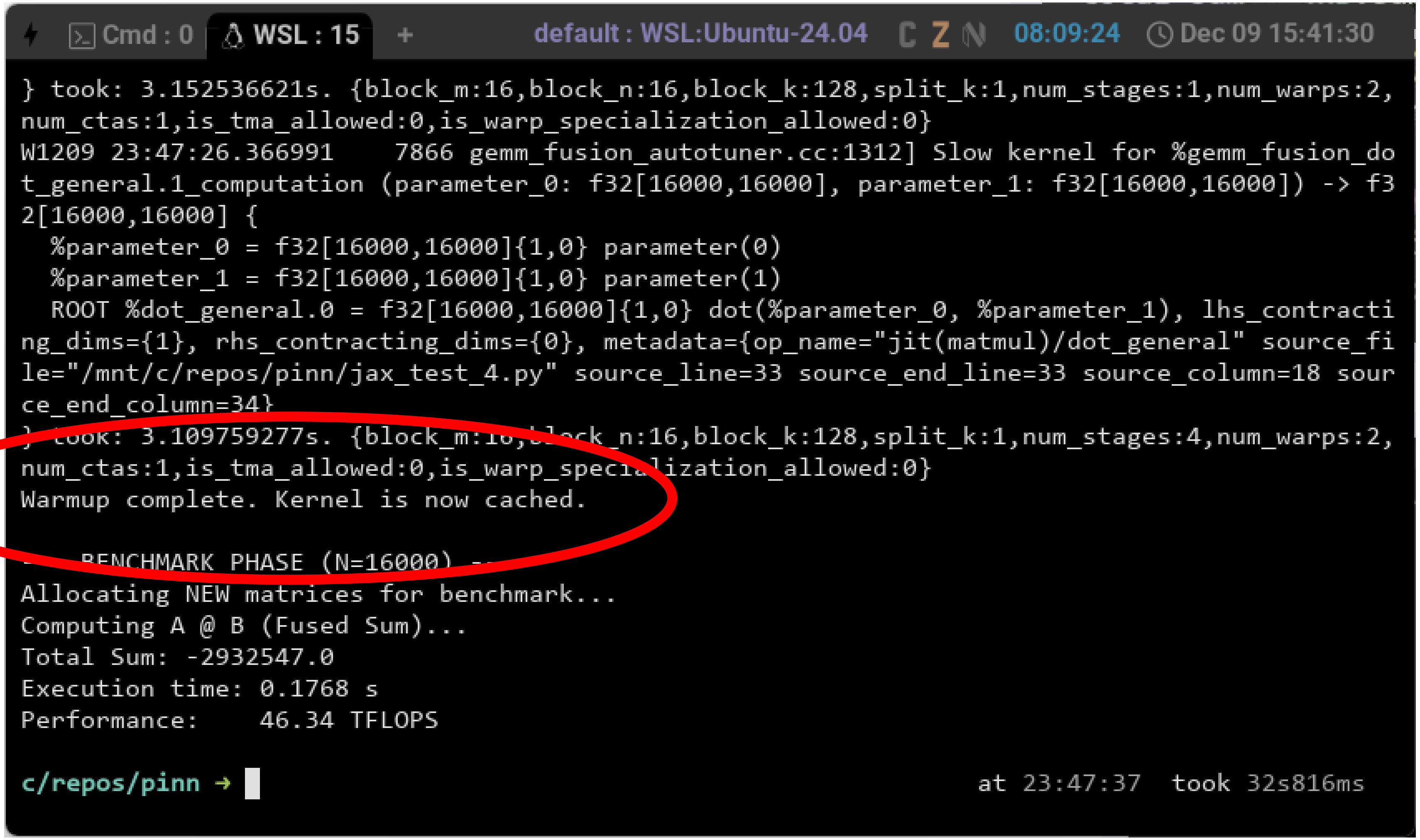
```
Cmd : 0   WSL : 15   +        default : WSL:Ubuntu-24.04   C Z N   08:08:46   Dec 09 15:41:30

c/repos/pinn ➜ python3 jax_test_4.py                                      at 23:46:33   took 25s966ms
W1209 23:47:06.031883    7965 cuda_executor.cc:1802] GPU interconnect information not availa
ble: INTERNAL: NVML doesn't support extracting fabric info or NVLink is not used by the devi
ce.
W1209 23:47:06.034310    7866 cuda_executor.cc:1802] GPU interconnect information not availa
ble: INTERNAL: NVML doesn't support extracting fabric info or NVLink is not used by the devi
ce.
JAX Backend: GPU
Device:      NVIDIA GeForce RTX 4080

--- WARMUP PHASE (N=16000) ---
Generating dummy data...
Compiling & Autotuning (this triggers the 'Slow kernel' logs)...
W1209 23:47:13.613217    7866 gemm_fusion_autotuner.cc:1312] Slow kernel for %gemm_fusion_do
t_general.1_computation (parameter_0: f32[16000,16000], parameter_1: f32[16000,16000]) -> f3
2[16000,16000] {
  %parameter_0 = f32[16000,16000]{1,0} parameter(0)
  %parameter_1 = f32[16000,16000]{1,0} parameter(1)
  ROOT %dot_general.0 = f32[16000,16000]{1,0} dot(%parameter_0, %parameter_1), lhs_contracti
ng_dims={1}, rhs_contracting_dims={0}, metadata={op_name="jit(matmul)/dot_general" source_fi
le="/mnt/c/repos/pinn/jax_test_4.py" source_line=33 source_end_line=33 source_column=18 sour
ce_end_column=34}
} took: 3.112700927s. {block_m:16,block_n:16,block_k:64,split_k:1,num_stages:4,num_warps:2,n
```

Example 1 – Matrix multiplication – Warmup phase

Cmd : 0 | WSL : 15 | + | default : WSL:Ubuntu-24.04 | 08:09:10 | Dec 09 15:41:30

```
W1209 23:47:20.047026    7866 gemm_fusion_autotuner.cc:1312] Slow kernel for %gemm_fusion_do
t_general.1_computation (parameter_0: f32[16000,16000], parameter_1: f32[16000,16000]) -> f3
2[16000,16000] {
  %parameter_0 = f32[16000,16000]{1,0} parameter(0)
  %parameter_1 = f32[16000,16000]{1,0} parameter(1)
  ROOT %dot_general.0 = f32[16000,16000]{1,0} dot(%parameter_0, %parameter_1), lhs_contracti
ng_dims={1}, rhs_contracting_dims={0}, metadata={op_name="jit(matmul)/dot_general" source_fi
le="/mnt/c/repos/pinn/jax_test_4.py" source_line=33 source_end_line=33 source_column=18 sour
ce_end_column=34}
} took: 3.152536621s. {block_m:16,block_n:16,block_k:128,split_k:1,num_stages:1,num_warps:2,
num_ctas:1,is_tma_allowed:0,is_warp_specialization_allowed:0}
W1209 23:47:26.366991    7866 gemm_fusion_autotuner.cc:1312] Slow kernel for %gemm_fusion_do
t_general.1_computation (parameter_0: f32[16000,16000], parameter_1: f32[16000,16000]) -> f3
2[16000,16000] {
  %parameter_0 = f32[16000,16000]{1,0} parameter(0)
  %parameter_1 = f32[16000,16000]{1,0} parameter(1)
  ROOT %dot_general.0 = f32[16000,16000]{1,0} dot(%parameter_0, %parameter_1), lhs_contracti
ng_dims={1}, rhs_contracting_dims={0}, metadata={op_name="jit(matmul)/dot_general" source_fi
le="/mnt/c/repos/pinn/jax_test_4.py" source_line=33 source_end_line=33 source_column=18 sour
ce_end_column=34}
} took: 3.109759277s. {block_m:16,block_n:16,block_k:128,split_k:1,num_stages:4,num_warps:2,
num_ctas:1,is_tma_allowed:0,is_warp_specialization_allowed:0}
Warmup complete. Kernel is now cached.
```

Example 1 – Matrix multiplication – Warmup phase



Example 1 – Matrix multiplication – Warmup phase

```
} took: 3.109759277s. {block_m:16,block_n:1
num_ctas:1,is_tma_allowed:0,is_warp_special
Warmup complete. Kernel is now cached.

--- BENCHMAR
Allocating N
```

XLA

SEND FEEDBACK

XLA (Accelerated Linear Algebra) is an open-source compiler for machine learning. The XLA compiler takes models from popular frameworks such as PyTorch, TensorFlow, and JAX, and optimizes the models for high-performance execution across different hardware platforms including GPUs, CPUs, and ML accelerators.

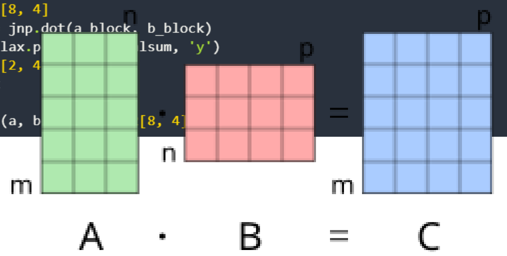
As a part of the OpenXLA project, XLA is built collaboratively by industry-leading ML hardware and software companies, including Alibaba, Amazon Web Services, AMD, Apple, Arm, Google, Intel, Meta, and NVIDIA.

https://openxla.org/xla

Example 1 – Warmup Phase

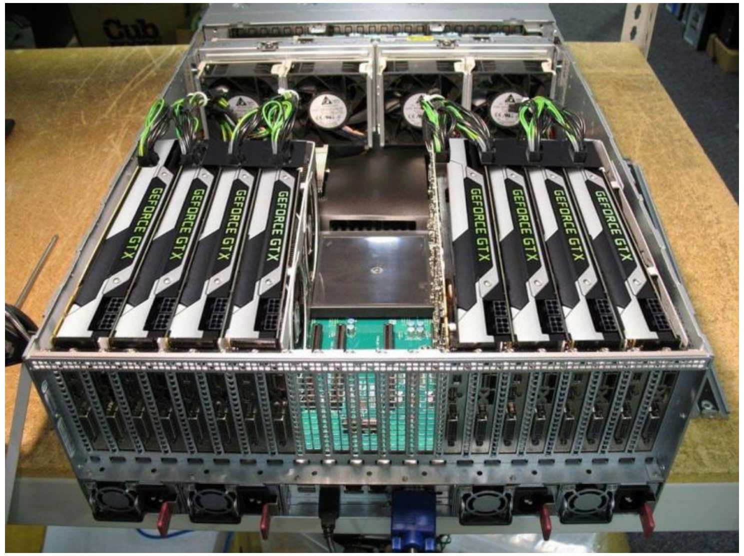
Computational structure (code)

```
from functools import partial

import jax
import jax.numpy as jnp

from jax.sharding import Mesh, PartitionSpec as P
from jax.experimental.shard_map import shard_map

mesh = jax.make_mesh((4, 2), ('x', 'y'))

a = jnp.arange( 8 * 16.).reshape(8, 16)
b = jnp.arange(16 *  4.).reshape(16, 4)

@partial(shard_map, mesh=mesh, in_specs=(P('x', 'y'), P('y', None)),
         out_specs=P('x', None))
def matmul_basic(a_block, b_block):
  # a_block: f32[2, 8]
  # b_block: f32[8, 4]
  c_partialsum = jnp.dot(a_block, b_block)
  c_block = jax.lax.p        lsum, 'y')
  # c_block: f32[2, 4
  return c_block

c = matmul_basic(a, b        [8, 4]
```



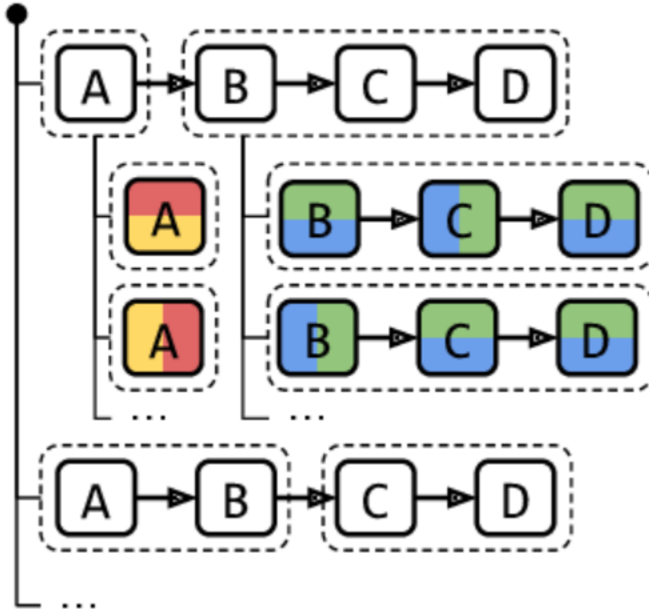
+

Hardware (10,000 GPUs!)



=

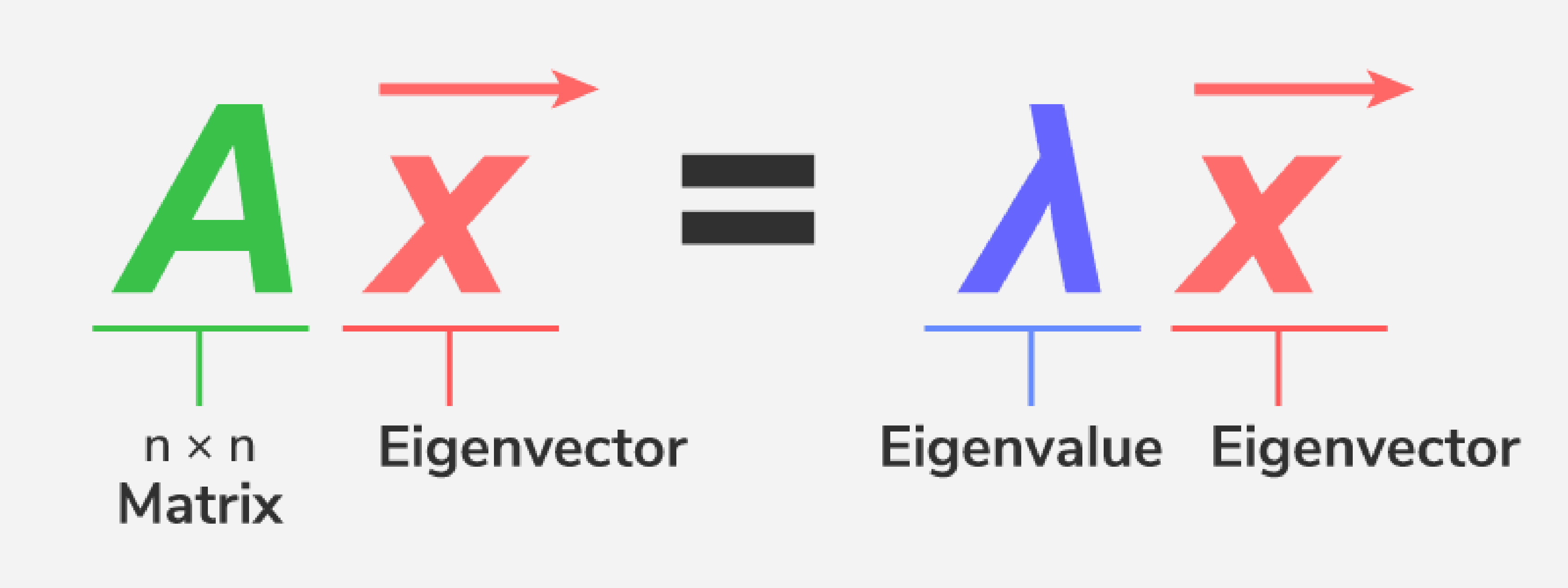
Computational Plan



Declarative! No need for low-level kernels.

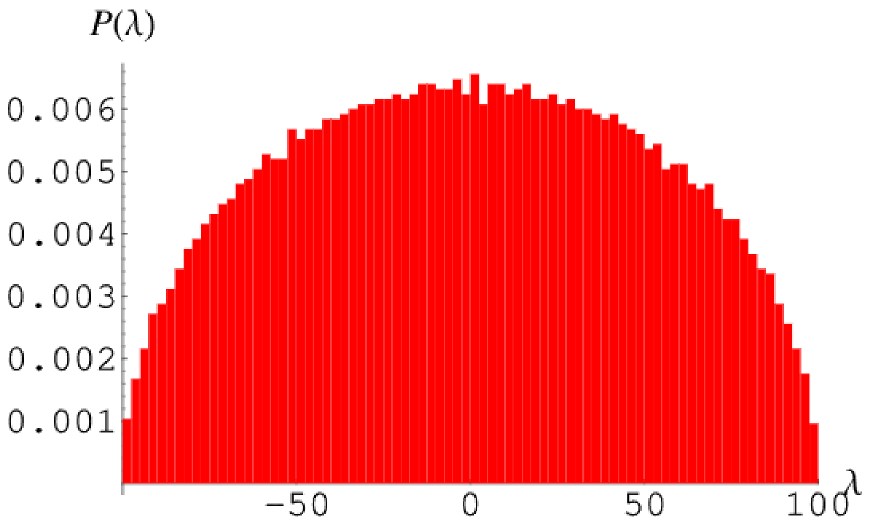
High performance reruns for different inputs.

Example 2 – Matrix eigenvalues



Example 2 – Wigner’s theorem

(Wigner 1955, 1958). This law was first observed by Wigner (1955) for certain special classes of random matrices arising in quantum mechanical investigations.



The distribution of eigenvalues of a symmetric random matrix with entries chosen from a standard normal distribution is illustrated above for a random 5000×5000 matrix.

https://mathworld.wolfram.com/WignersSemicircleLaw.html

Example 2 – Wigner’s theorem

matrix element mean = 0, variance = 1/n, std_dev = 1/sqrt(n)

```python
def generate_symmetric_matrix(key, n):
    """
    Generates a symmetric matrix where elements have mean 0 and variance 1/n.
    """
    key, subkey = random.split(key)
    # Generate random normal values with std dev = 1/sqrt(n) -> variance = 1/n
    scale = 1.0 / jnp.sqrt(n)
    A = random.normal(subkey, (n, n)) * scale
    
    # Symmetrize: Use upper triangle for both upper and lower parts
    # M_ij = A_ij for i <= j
    # M_ji = A_ij for i < j
    M = jnp.triu(A) + jnp.triu(A, 1).T
    return M
```

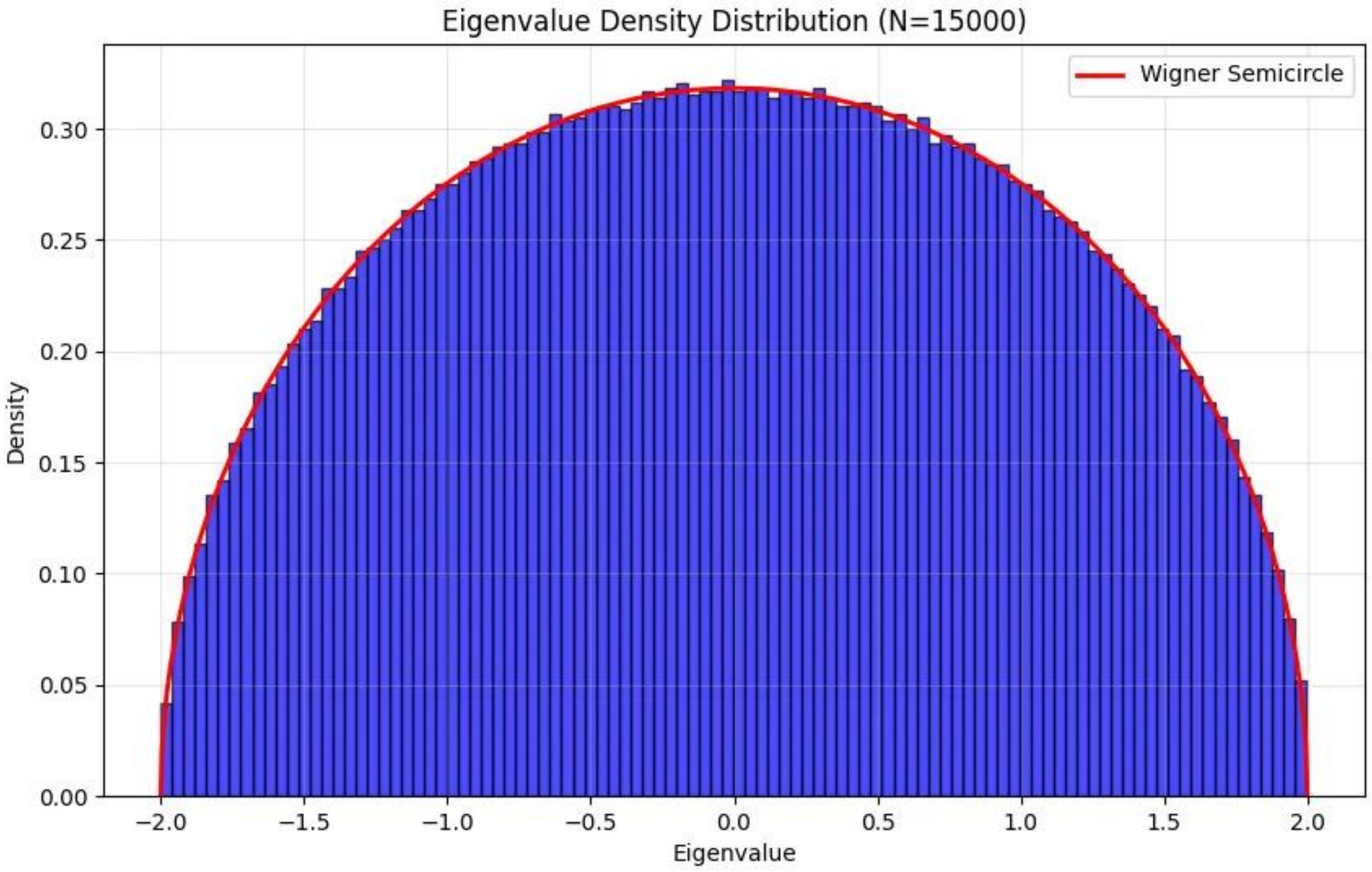
Example 2 – Wigner’s theorem

```python
print("Calculating eigenvalues...")
start_time = time.time()

# eigh is optimized for Hermitian/symmetric matrices
eigens_jax = jnp.linalg.eigh(M)[0]

# Block to ensure calculation is done
eigens_jax.block_until_ready()
end_time = time.time()
```

Example 2 – Wigner's theorem, N=15000 in 30s

```
Cmd : 0   WSL : 15   +        default : WSL:Ubuntu-24.04   C Z N   08:20:43   Dec 09 15:41:30

c/repos/pinn → python3 jax_test_7.py                                    at 00:01:03   took 30s154ms
W1210 00:01:23.110434    8463 cuda_executor.cc:1802] GPU interconnect information not availa
ble: INTERNAL: NVML doesn't support extracting fabric info or NVLink is not used by the devi
ce.
W1210 00:01:23.113145    8364 cuda_executor.cc:1802] GPU interconnect information not availa
ble: INTERNAL: NVML doesn't support extracting fabric info or NVLink is not used by the devi
ce.
JAX Default Backend: gpu
JAX Devices: [CudaDevice(id=0)]
Generating 15000x15000 symmetric matrix...
Generation took: 0.9170 seconds
--- Symmetric Matrix Statistics ---
Mean: 1.744398e-07 (Expected: 0.0)
Var : 6.665583e-05 (Expected: 6.666667e-05)
Calculating eigenvalues...
Eigenvalue calculation took: 25.6521 seconds
First 5 eigenvalues: [-2.000454471200796, -1.9960383140235098, -1.9925514431270979, -1.99139
45334543062, -1.9900501617860438]
Plotting eigenvalue distribution...
Plot saved to /mnt/c/repos/pinn/eigenvalue_dist.webp

c/repos/pinn → ▯                                                        at 00:01:51   took 30s262ms
```

Example 2 – Matrix eigenvalues, Wigner's theorem

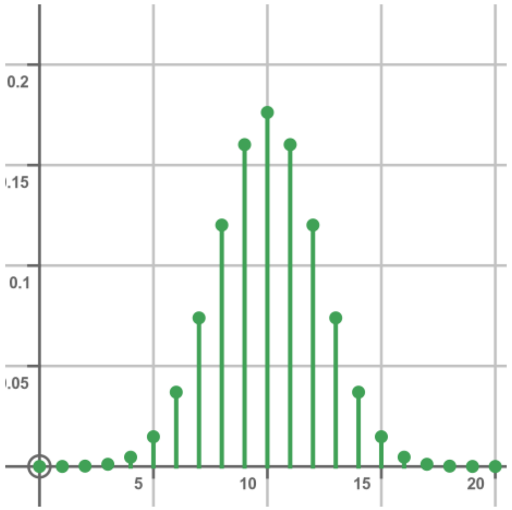


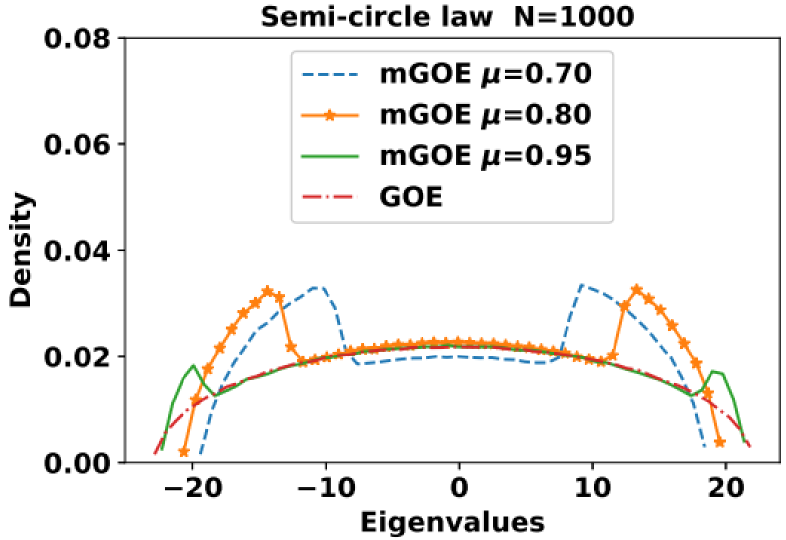
Example 3 – Wigner's Batman

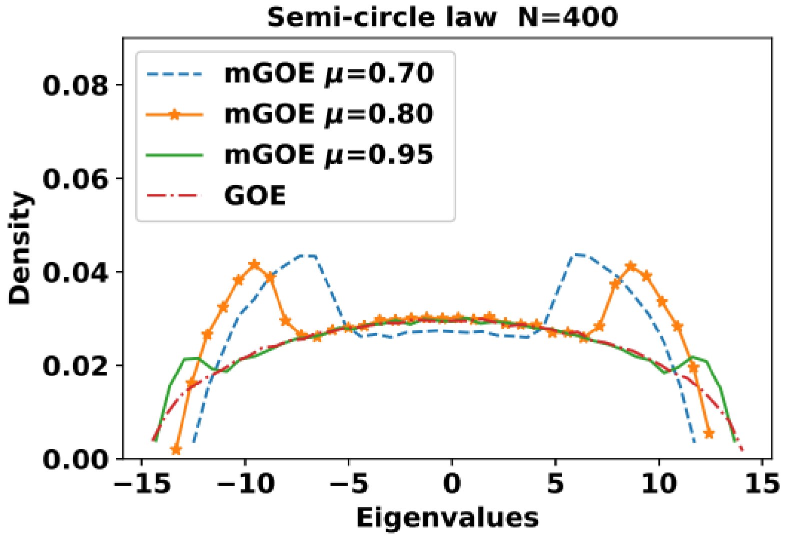
Empirical deviations of semicircle law in mixed-matrix ensembles

Mehmet Süzen*
(Dated: December 3, 2021)

An algorithm is introduced for sampling a set of matrices from mixed orders random matrix ensembles, i.e., Mixed Matrix Ensemble Sampling (MMES). The concept of *the degree of mixture* of the matrix ensemble provides a balanced sampling of the mixed matrix ensemble. As an application of MMES, we have shown how the semicircle law deviates from the conventional behaviour in mixed Gaussian Orthogonal Ensemble (mGOE) as a novel finding.







https://hal.science/hal-03464130v1/document

Example 3 – Wigner’s Batman

```
# 1. Sample Matrix Orders
matrix_orders = random.binomial(key_orders, N_MAX, MU, shape=(N_MATRICES,))
matrix_orders = jnp.clip(matrix_orders, a_min=2, a_max=N_MAX)
orders_np = np.array(matrix_orders)
```

```
print("Generating mixed ensemble...")
start_time = time.time()

for i, order in enumerate(orders_np):
    eigs = get_periodic_eigenvalues_sorted(keys[i], int(order), N_MAX)
    eigs.block_until_ready()
    all_eigenvalues.append(eigs)

raw_eigens = jnp.concatenate(all_eigenvalues).tolist()

# 2. Symmetrization
symmetric_eigens = raw_eigens + [-x for x in raw_eigens]

print(f"Done in {time.time() - start_time:.4f}s")
```

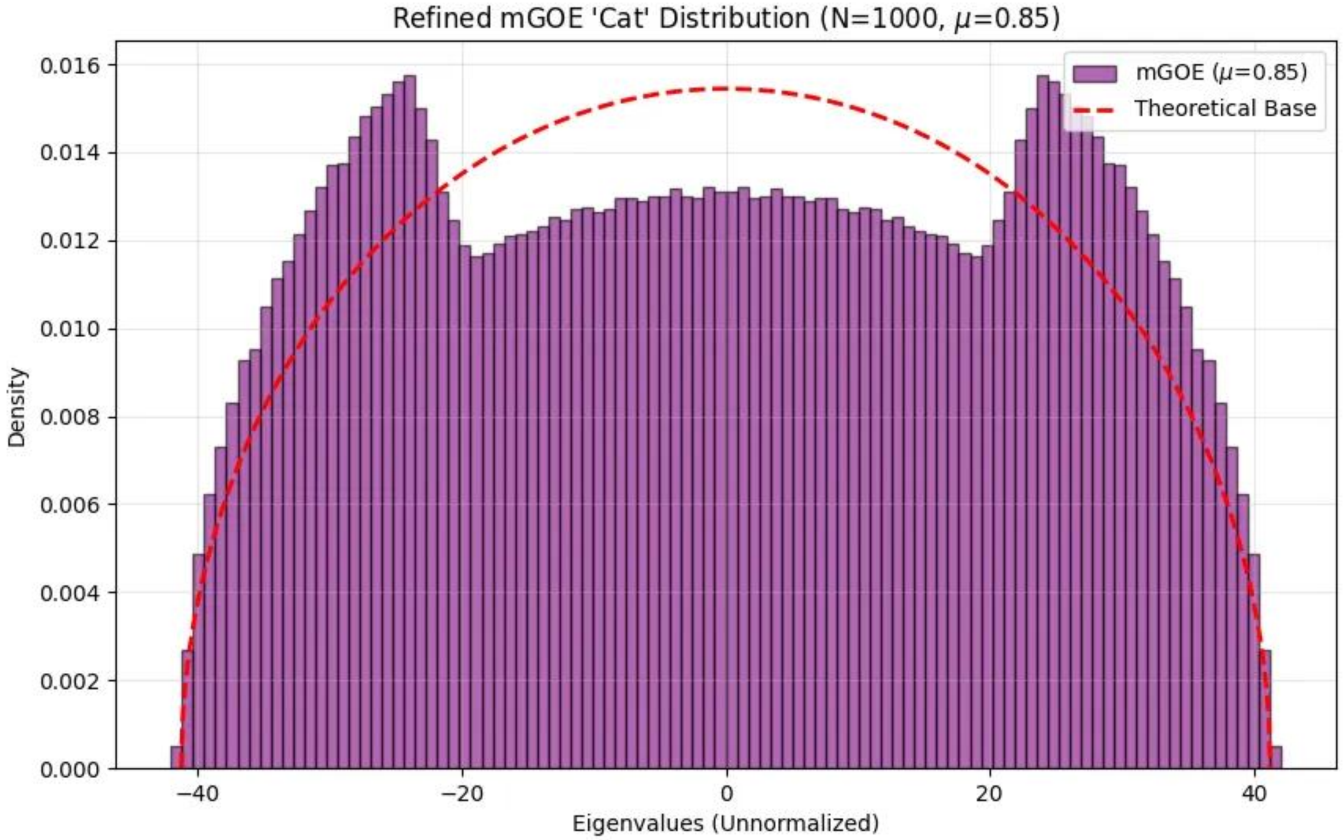
Example 3 – Wigner's Batman, 100 matrices ensemble in 29s

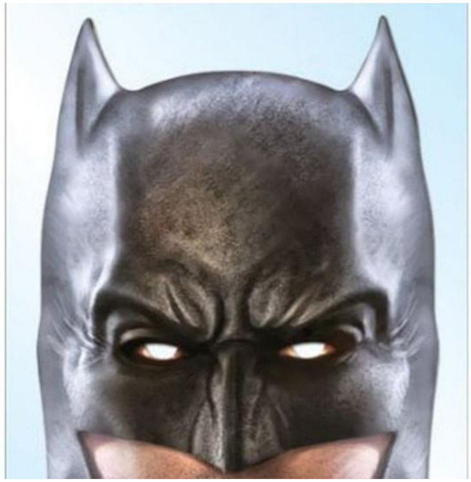
```
N_MAX = 1000
N_MATRICES = 100
MU = 0.9
```

```
c/repos/pinn ➜ python3 jax_test_Wigner_cat.py                     at 00:35:34   took 28s892ms
--- mGOE 'Cat' Final (Scaled) ---
N=1000, mu=0.9
W1210 00:37:24.564644    9536 cuda_executor.cc:1802] GPU interconnect information not availa
ble: INTERNAL: NVML doesn't support extracting fabric info or NVLink is not used by the devi
ce.
W1210 00:37:24.566749    9437 cuda_executor.cc:1802] GPU interconnect information not availa
ble: INTERNAL: NVML doesn't support extracting fabric info or NVLink is not used by the devi
ce.
Generating mixed ensemble...
Done in 24.3746s
Plot saved to /mnt/c/repos/pinn/wigner_cat_final.webp

c/repos/pinn ➜                                                     at 00:37:51   took 28s829ms
```

Example 3 – Wigner's Batman

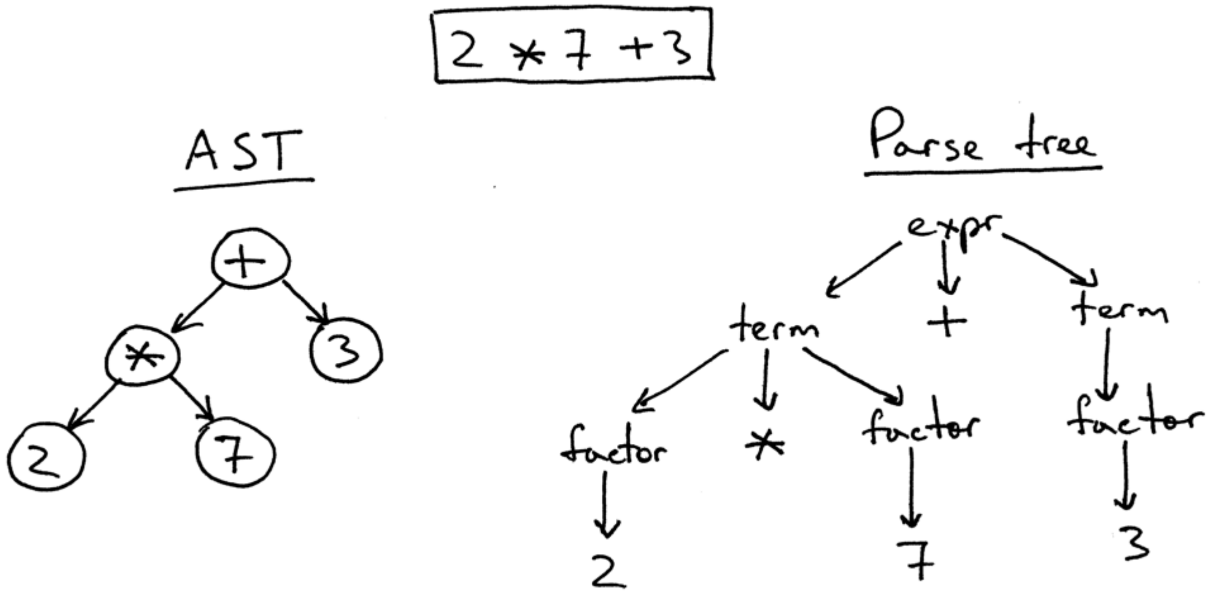
Combined distribution of eigenvalues for binomially distributed size matrix ensemble:





Example 4 – Compound computation, GPU-focused

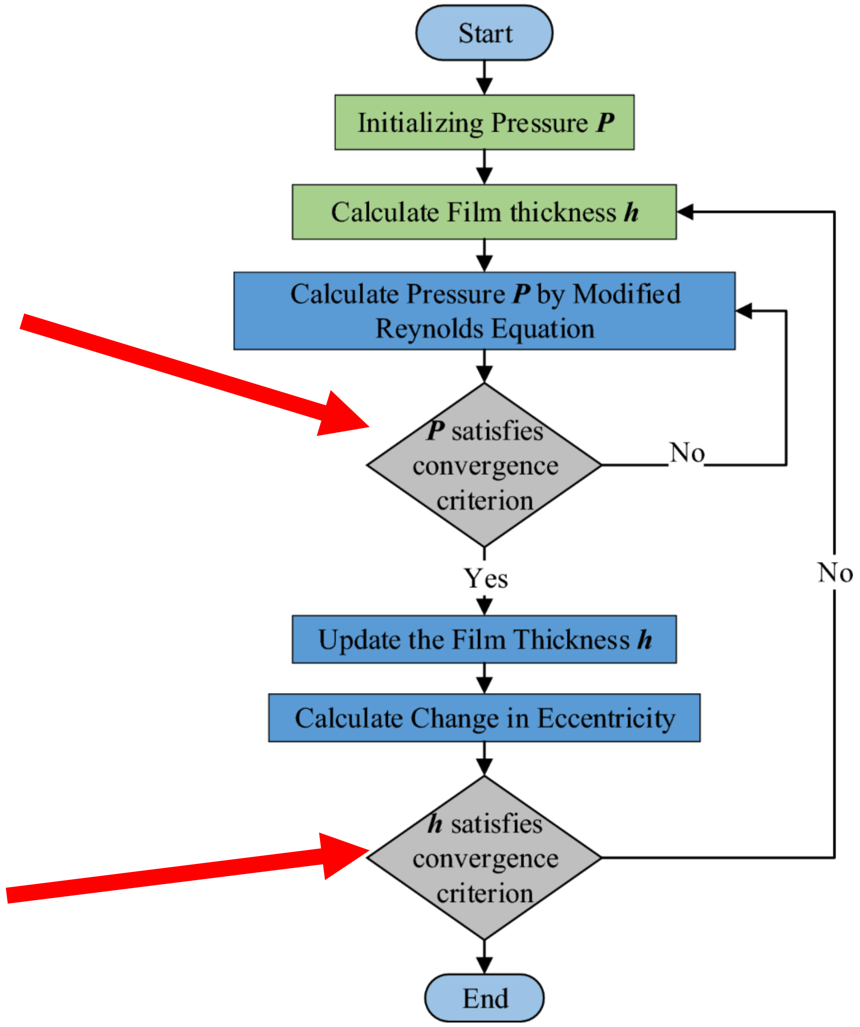
The Task: $x_{t+1} = \tanh(W \cdot x_t + b) + \alpha \cdot x_t$ **Steps:** 100,000 **Matrix Size:** 2048×2048



Code is converted into a computational graph which is than transformed and optimized for target platform and final output!!!

Loops, derivatives, summation…

Example 5 – Dynamic loops, conditional stop, GPU-focused



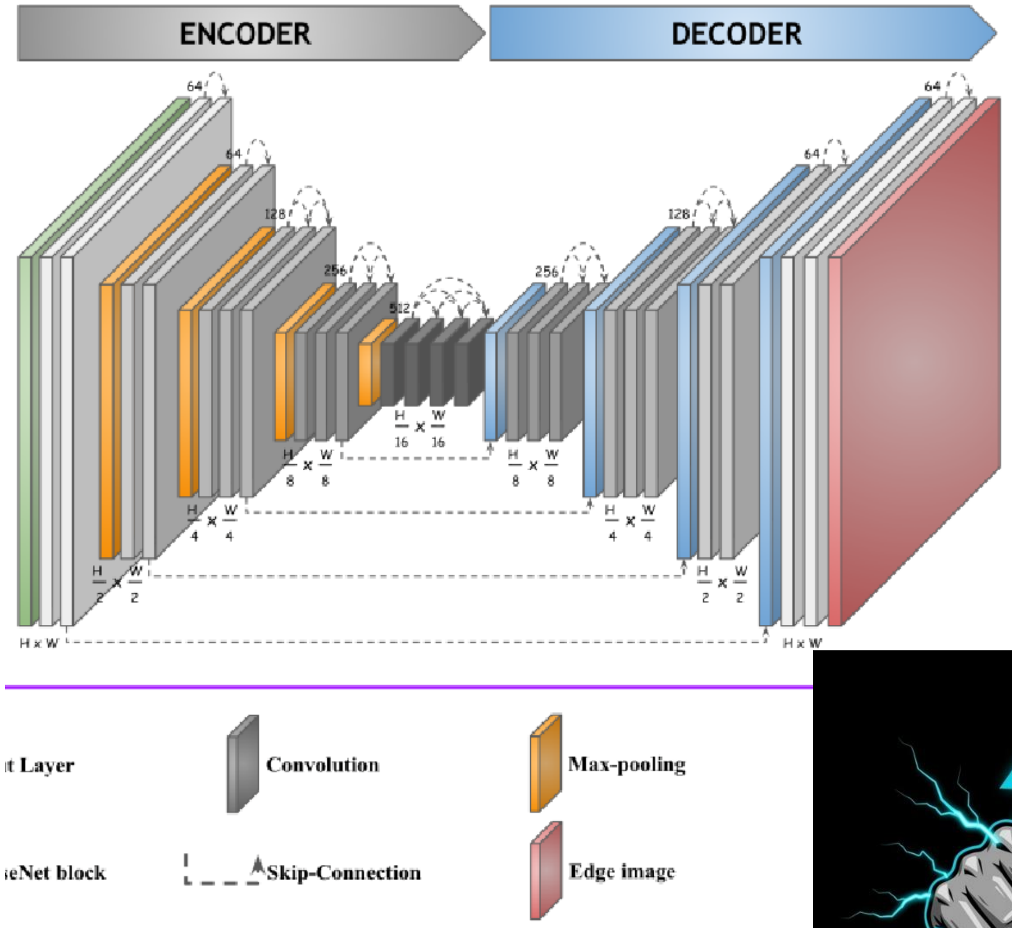
- Use: solvers, training…

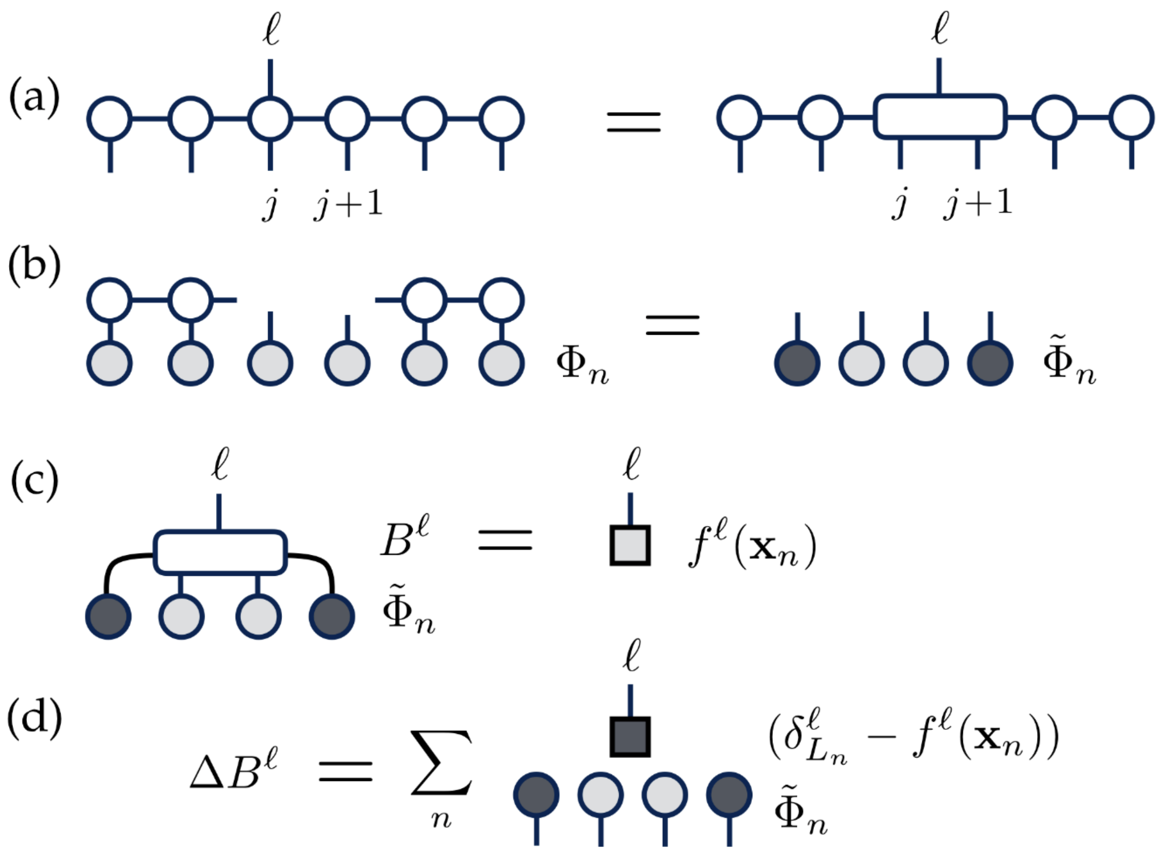
JAX vs PyTorch

- Problem for high-level pytorch
  - CPU-bound, or:
  - Need to go into low-level *Triton*

- Possible via high-level JAX

Example 6 – Derivatives, Hessian, AI, Tensor Networks, QC





Thank you

AVL

www.avl.com