![A blue and black text

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAkACQAAD/4REARXhpZgAATU0AKgAAAAgABAE7AAIAAAASAAAISodpAAQAAAABAAAIXJydAAEAAAAkAAAQ1OocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFRhbiBIb25nIEtleSBBbmR5AAAFkAMAAgAAABQAABCqkAQAAgAAABQAABC+kpEAAgAAAAM1NgAAkpIAAgAAAAM1NgAA6hwABwAACAwAAAieAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAyMzoxMToxMyAxODo0Mzo0NQAyMDIzOjExOjEzIDE4OjQzOjQ1AAAAVABhAG4AIABIAG8AbgBnACAASwBlAHkAIABBAG4AZAB5AAAA/+ELJGh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iLz48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOnhtcD0iaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLyI+PHhtcDpDcmVhdGVEYXRlPjIwMjMtMTEtMTNUMTg6NDM6NDUuNTYwPC94bXA6Q3JlYXRlRGF0ZT48L3JkZjpEZXNjcmlwdGlvbj48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOmRjPSJodHRwOi8vcHVybC5vcmcvZGMvZWxlbWVudHMvMS4xLyI+PGRjOmNyZWF0b3I+PHJkZjpTZXEgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOmxpPlRhbiBIb25nIEtleSBBbmR5PC9yZGY6bGk+PC9yZGY6U2VxPg0KCQkJPC9kYzpjcmVhdG9yPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMABwUFBgUEBwYFBggHBwgKEQsKCQkKFQ8QDBEYFRoZGBUYFxseJyEbHSUdFxgiLiIlKCkrLCsaIC8zLyoyJyorKv/bAEMBBwgICgkKFAsLFCocGBwqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKv/AABEIAFYBDwMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/APpGiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPmnxt498TaJ8UNTWy1m7Fta3mUtjMfKIGDtK9MGqd/L8UPEto+vSDWWs2BkU27NHGF65RFIyMdwD9aq+NraO8+Nd/azDMc2ppG4HcEqD/ADr6rRFjRURQqqMKoGAB6V7datHDwg4xTbR49KlKvKacnZM8B+DvxJ1mfxTBoGuX0t9a3assL3DbnicAkfMeSCARg98V2nxd+JM3g21g03Rtn9q3aF/MYZEEecbsHgkkHGeODXkPgpFi+ONnHGoVF1ORVUdAMtxVn44vK3xSvBJnasEIjz/d2A/zJq5YenPFLTS1yY16kcM9db2INIs/ib4tVtU0q41m6QMcXH20xKTn+EswBx6L0qPWviJ49tUh0jVNSvrG5sSwc8xTPnGN5GC2B0PfPfrX0v4ZhtbfwnpUWnhfsq2cXlbehXYMH8eteJ/tFQ2q+INHljCi6e2cS46lAw2Z/EvUUMRGtW5JQVuhVahKlR51J36nqHg7Xxb/AAn0/XNeu3kEdmZrieVtzNgn8z2rxu/8e+OviT4hew8LG5tYOTHbWcnllU/vSSZHt1IHoK6LxJcSwfsw6QkRIWcwxyY/u72b+aiuC8Ban420uC8k8EWM06Ssq3EkVmJiCAcAkg46niihRilOpZXu0r7BWqybhTu7WV7bmtq2kfFDwFbrqs+o3wt1I3yxXhmRT23qSRj6jFev/C34gf8ACc6HL9tRItTsiFuFThXB+64HbOCCPUe4rzDUNe+MGqabcWF7pF5Lb3MTRSodLX5lYYP8NTfCLQ/E/hjxHqNzfaPe2lu+my/NPAyqXUhl69+D+ZorQVSi3Pl5ltYKM3CqlC/K+5e+JXxc1WTX5fDvgx2j8qTyJLmFd0s0mcFE9ADxkck9PfEfwN8Wo7M6mbvUjMBvMa6mTNj6buvtnPtXDeFrnW4fFFvdeHIHutVQs8SiETNnBy20g5OCTn8a9J/4Sz40f9Ay9/8ABWv/AMTW0qbopQp8vnfdmUaiqtyqc3yNX4TfFbUtR1pPDfiqXzpZcrbXTja+8D7j+uQDg9c8HOePa6+WLLw345k8cW2u3Xh7UFuTfJcySLaFF3bwxOAMD3r6nrzsdThGalC2vY9DBznKDU+nc8e+MfxO1Lw7qEegeHpBb3DRCW4usAsgOcKuehwMk+4xXMW3w8+KWrWMepvrM6PMgkWObUpBIQRkdOB9MjFdb8UPiPofh7Vn06DQbHV9XVQZZLqFWSHIyoPGWOMHGRjI57Vy0fiz4weK4wdJsri1t5B8rW9osKEe0kn9GrqoqcaUXCKj5vqc1VwlVak2/JdBvw++I/ibRfHUPh3xTdT3cM1z9jlS6bfJBKTtBD9SN2AeSMciuq+POv6toVrojaNqVzYmZ5hIbeUpvwExnH1NeP2NtqFn8VLKDW3aTUI9WhFyzPvJfzVzlu/PevUP2jv+PPw//wBdJ/5R1c6cFiqbSWq+WxnCpP6vNN7HFWfir4keNNOh0rR59QuVtEImltSUaTJJBkk47cAZGQO5zWfZeM/G3gfXvKvL2/SWFwZrK+dnVx6EN2I7j8DXuHwQtIbb4WWM0SAPdSzSSn+8wkKD9FFecftDxqPGemuFAZrAAn1xI+P51VOrCdeVDkVtQqUpwoqtzO+h6x4y1yST4SXut6RPJbvNYpcQSxttZA20jB7HBrwTRviP48mjm0rTNSvb66vioQnM0yYzkJnOM9z7dq9e1T/k2pP+wLB/JK4v9nW0hk8RaxdugMsNsiIx/hDMc/8AoIrChyU6FSTjezNa3POtCKdro47VZ/iH4RuorrV7zWrGSU5SWW5dlc9xnJB+hr334VeMLjxl4NW61Aqb62lNvcMoxvIAIbHbII/EGsz47xq/wykZlBKXcTKT2PI/kTWP+zt/yK+rf9fg/wDQBSrSjXwvtHGzTHSi6OJ9mndNHonjaWSHwDr0sLtHImnTsrocFSIzgg9jXyV/wkmuf9BnUP8AwKf/ABr6z8d/8k88Q/8AYMuP/RbV8dVrliThK/czzBtTjY0v+Ek1z/oM6h/4FP8A40f8JJrn/QZ1D/wKf/Gn23hbxBe2oubPQtSuLcjIlitJGUj1yBiux8D+JtC0LRpLPVlNpeLK3mM0DMX9OgJGOmDWuPxMsNR9pRpe0d9lv+T/ACMcJQVeryVKnIu7/pHF/wDCSa5/0GdQ/wDAp/8AGj/hJNc/6DOof+BT/wCNetJ468JzME+3Rjdx89u4H5lcVneL/BOn6rpcmpaJHHHdKnmjyfuTrjPQcZPYjrXh0uIUqsaeLw7pKWze34pHqVMnbpueGrKpbdL/AIdnf/Aq8ur74fSy3tzNcyfb5F3zSFzjanGTXpNeYfAD/knM3/YQk/8AQEr0+tMVpXl6l4b+DH0CiiiuY3PljxX/AMl4uP8AsLxf+hLX1PXP3HgTwxd6u2qXOjW8l80glM5B3Fx0PX2roK68RXjVjBJbI5aFGVJyb6s+WPB3/JdbX/sKy/zavTPjV8O73xEsGvaFAbi8to/Knt0HzSxgkgqO5GTx1IPHSu7tvAnhiz1ZdUtdGt471ZDKJ1B3Bj1PX3NcX8Y/EHjLws9nqXhu8aHS3Ty7gLbRyeXJkkMSykgMCB6fL711LEOtXg6WjStqc7oKlQkqmqv0PKvD3xW8W+D7D+yITFLDCdscN7CS0PsMEH8DnFYvi2XxJqV5DrXitJxLfqfIaZNm5Fx91eMLzxxzyeea9U8OfHzTl0uFfFmm3M2owjBubWKNhL74JXafYcfTpXB+PvGNz8TPFVoNM0+VUjTyLW2HzyOScknHc8cdsV6FLmVVt01Huzhqcvs7Kd+yPYNH8Njxb+z7Y6QGCSzWYaFm6LIrllz7ZGD7E15B4L8aat8LfEN7a3lgzxyER3dnKdjBlzhlPODyfUEH6GvpHwho8mgeDtK0ucgzWtsiSYORvxlse2SaNe8IaB4mC/27pVvdsowsjAq4HoHXDY9s15cMVCLnCavFs9GeHlJRnB2kkeVax+0TAbB00LRZVumXCyXbjah9dq/e/MV2nww8caj440WWXVdJa3MOEN0g/c3B7hQeQR3HI9x0qez+EfgexmEsWgxSMOgnlklX/vlmI/Suwhhit4Uht40iiQbURFCqo9AB0rOtUw7hy0o692aUqddS5qkvkj5Y8Q6Rq3wo+IyXdkhWKKYzWMzAlJYz1Q/gdpHXv3Fehx/tF6f9hDS6Bc/a8cxrOvl5/wB7Gf0r1zUtKsNZsmtNVs4by3bkxzIGGfXnofeuT/4U54E8/wA3+wlznO37TNt/LfitvrVCrFe3i212Mfq9ak37GWj7nLfDv4va34r8VPpt5oqzQTMWWW0yPsqf7ZJwR78HPQHgV7BVLStF03Q7X7No9hb2UOclIIwu4+px1Puau1xVpwnO8I2R2UozjG03dnyl8R7S60D4tahcX0Hmq14LyISD5ZYydwHuP4T9DXqk/wC0L4cTTvMt9N1CS7K8QMqKob0L7jx7gfhXouu+GdG8TWywa7p0N4i52GQYZM9drDkfgaxdM+FfgvSboXFpoMDSqcqZ3eYD6ByR+ldssTQqwiqsXddjkWHrU5t05KzPm7T9TuNW+J9lqd+BHNdarFNIAMBd0oOPpXqf7R3/AB5+H/8ArpP/ACjr0u98A+FtQ1KTULzRbaW7kcO0xBDFh34PXirmt+GdG8SLCuuafDeiAkxCXPy5xnH5CnLGQdWE0vhFHCTVOcG9zmvgv/ySXSPrP/6PevM/2iP+Rv0v/rx/9qNXvel6VY6Lp0dhpVslraxZ2RJ0XJJP6kmqGteENA8RXMdxrelwXssabEeQHKrnOOvqawpYiMMQ6rWjv+JtUoSnQVNPXQ4zVP8Ak2pP+wLB/JK5L9nL/kJ69/1xh/8AQmr2uTRNNl0MaNJZxtpwiEItj93YOi/TgVX0TwpofhuSZ9D02GyaYASGIH5gOnX60LER9jOnb4ncToS9rCd9kcf8df8AkmE3/X1F/Osb9nb/AJFfVv8Ar8H/AKAK9T1bRtP16wNlq9pHd2xYMYpOhI6GotF8O6T4ct5INEsYrKKVt7rHnDHGM1KrxWHdK2tynRk66q9LFPx3/wAk88Q/9g24/wDRbV5J8Dvh9Z6jbv4n1q3WdUl2WUTjK5X70hHfB4HoQT6V63475+HniHH/AEDbj/0W1c/8IGMfwa0x14IW4I/7/SVdOpKGGly9Wl+BNSEZ4iPN0VyHWPjb4U0bXH0x/tlyYX8uWe2iVo0YcEZLAnHsD+NJ8QfBuk/ELwedZ0YRS36wedZ3UXHnKOdjeueQM8g/iK+YSSSSeSeprq7XSfFmqeHtOfTpbiexjV1hijn2iI+YxbgkdSc5/wAK7KuHo4XlqOpya2u3ozjp16uI5ocnN5I5QAk4HJr3HwNbXOneDLVNSzGyhnCvwY0JJGfT1/Go4/F/hOFVY3turoOSIWyD/wB81ynjD4ix6jZSadoYcRSjbLcONpZe4Ueh9T+VfM46rjs8UMLHDunG93J3/VLv8z28LTwuVOVd1lN2skv+HZ6t8CGR/AN00QwjanMVHoNqYr0uvMPgB/yTiX/sIS/+gpXp9epiVatJHLh3elFhRRRXObhRRRQAU2WKOaJopkWSNxtZHGQw9CKdRQBy8/w08GXEzSyeHbEMxydibB+QwK1tL8OaLofOj6VZ2TEYLwQKrEe5Aya0qKt1JtWbZCpwTukFFFFQWc7L4ztY9buNOjsL64FpNHBc3EKIyQu4BXK7vMI+YZYKQM9eDgi8ZW1xebINN1CWy+1Gz/tBI0MPmhtpGA2/aG+Xdt25796ztZ8Cz6vrj3jX1oqtPHNHcNZf6ZbBduUimDDCnb3Bxk9asWXhXVNOb7FY60kGkfbWu/LW3P2gBnMjReZvxtLE87c4OPeiPS/9bf8AB/rcfW39b/8AAM7wp45eWG2tdYtNQY3OoXNpFqLxoIGdZJNqcNu+6uM7cZGM1r2fjiyvJ7QiyvYrG+mMFpqEioIZ3GcAYYuAdpwWUA/iMsg8FiLS9MsjfbhYam9+W8nHmbmkbZjdx/rMZ56dOarWXgi6gi0zTbrU4ptG0m4E9rAtsVmYrny1d9xBC57KCcD3yle39eX/AAbDe7t/W/8AwC5Z+N7K8ntCLG+isb6YwWmoSKghnfnAGGLgHacFlAP4jLvCusNJ4BttW1m5zthklnncdFVmyTj0AqjY+Cbu3j0vT7nVIptH0m4FxawrbFZmK58tXfcQQueygnA986lj4Xht/BJ8N3U7XED28lvJKq7CyvuzgZOPvUP4XbcFbm12KJ8cK9ucaPqVtPcWklzYrOsQ+1KoyduJPlIBB2vtOPxpfDOpXEnwzg1XV5r7zpLE3M0snl+bym4lNvygd1B9s0238JajNdWcut6rBdDTraS3tBBamIkumwySZc5O0YwMDkn6atvoPkeC08P/AGndssBZ+f5eM/u9m7bn8cZ/GnLaVvl+P/ACO8b/AD/D/gmJL8Q7CyikVdP1a8hsreGa7ulijKwxyIGDv8wycHJCgkYPGKrDxfNaeMNbbUYtR+x2n2e2s7aMQlZpJMYwA24uxORkgBQc4q8PAuNF1uw/tH/kK2UVr5nkf6rZD5e7G7nPXHHp70X/AIMF3e3k8OpLFeST2t3bEw7vJeBdoLLu+ZW5BHHXr3pytzaf1/SJXwosN44soILkXtjfWt9bSxQnT3VGmkaXiMJtYowbB53YGDnGKfJ4xtraxuJr/TtQtLmGaO3FnJGpklkk+4EKsUbPqG4wc4xWfceB7vUJbnUb/VIv7Zknt5oJ4bYiGDyCxRdhYlgd75+YZ3cYxVi68LapqdrJJqetI1+t1DdWhhtyILZougEZclg2Tu+YZzxjFL1/pafjuP8Ar+vIdN48s7SGYXum6jBeQTwwPYmNHlzKcRkbXKspIPIbsc1DB8Q7SSdY5tG1a2CXa2Vy8sUe21mcgIrkOc53Lym4DcM4po8HT32oy6hqeqQyai13aTSfZ7crGkcDFljClyRksxLE9+nFWrjwf9oi1BPt237bqsGo58nOzyzEdn3uc+V14xu6cci31/rb/g/d94/L+t/+AVm+ItmLhY49G1eWOS6lsopkgQrLOhYGNfnzzsJDEBeOSOa3dC1uHX9NN3BBPbMkrwSwXCgPFIjYZTgkdR1BIrLt/B/kQafH9u3fY9Wm1LPk43+YZTs+9xjzevfHTnjT0PR/7GivU8/zvtV7Nd52bdvmNnb1OcetEdtf62/4P3L5kt9P63/4H3/db1Cyj1LTLqxuBmK5heF/91gQf0NeFeH/AIow+AfB114R1bS7htV0154I9hHluWdmyxJyOWPQHIxXvteP/Gb4Zy60G8SaBB5l9GmLu3QfNOoHDKO7AcY7jHcc9uElTcvZ1dn+ZyYmNRL2lPdfkfPld54V+IFroPhn7BcWk0k8RYwlMbWyc/MScjk9s1wfTrRXs43A0MdSVKurq9zx8Li6uEn7Sk7O1gJyxJ7miivV/g/8Mpdc1CHxBrcBTS7dt9vG4x9pcHg4/uA/mePWuirVjRhzSMqdOVWXLE9f+GGgSeG/h5ptncxmO5kQzzqeoZznB9wMD8K62iivlJyc5OT6n00IqEVFdAoooqCgooooAKKKKACiiigAooooAKKKKACiiigAooooAiuxMbOYWrpHNsPlvIhZVbHBIBBI9sj615doPjHXYPD+gad532m8uNN+2yXh064vDt3bVUqj7i2c5ckDpxzXqxGQQehrnI/AuiQ2drb263kIswy28kV7KkkaNjMYcNnZwPlzjil1/rzH0Myx8U65rlxpthZW9tpV7NYPeXRvbd5Am2TywqpuQ8kE5J4GOMms2W68R3XjeKfTm0qC+GhE3DSB7iHKzsNqbWU8kdSePQmusn8H6NLb2cUUEtp9hjMVvJaXEkMiIfvLuVgSDjJyTzz15qxY+G9J0143sbNYTHa/ZFw7cRbi2OvJySc9TnrQ+68/ydvzQeXp+aOQj8Z+I9YhWXRo9MtlTRINUkF1HJISz78xjay4Hyfe7ehzxNH4x1vV7LUNQ0dLC2t9Ms4riWG6jd2nd4RMVDBlCAKwGcNz24rp7Pwxo9hEY7S08tWs0sSPNc/uE3bU5Pbc3PXnrVS48DaDcKqm2mjj8hLeSKG6ljSeJBhVkAbDgDj5s8cdKb62+X4/8D+t0ul/62/4Jy9v4mmh1TVr7TIU+0avcabDbi4zsjMsIOXAwSAM8AjJ4yKvXni/WtMvbrRJ0sbrVRPaxW1ysbxQETlgC6bmIK+W3AbnjpmuluvC+j3cd2k1mMXhjaUrIynMYAjKkH5CuBgrisrV/Btv/wAI5eWulWsd1dXMsc0rahcys8xQjH77JdGAHysPuntQ7X+f4af8HUFt52/EzpvFfiHSLjVodWXTbhdJNlLLNbROm+CZ3WRtpc7SoXOMkcH146Tw5q8+tw3t2yRraLeSQ2jIDmSNDtLE55y4bGO2K5G38I6xZab4gey06E3GtQxWnk3mpPOyKqyAyySMCXP7z7o7KOe1dxoulQ6Hodlpdr/qrSBYVPrgYz9T1oXW/wDX9fqxytpb+v6/QvUUUUCPPPG/wd0XxZLJfWTf2Xqb8tLEmY5T6unr7jB9c15Pc/AnxnDfeRDDZ3ERP/HwlwAoHuGw36Gvpuiu2lja1Jcqd15nJUwdKo7tW9DyDwd8BbDTJo7zxVcJqU6HItYgRAD/ALRPL/TgeoNeuxxpFGscSKiIAqqowFA7AU6iuerWqVXebNqdKFJWggooorI1CiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA//9k=)

**Emerging AI Trends in Cybersecurity**

**(EATC)**

Year 3, Semester 5

**SCHOOL OF INFOCOMM TECHNOLOGY**

Diploma in Cybersecurity & Digital Forensics

**ASSIGNMENT**

|  |  |  |
| --- | --- | --- |
| **Tutorial Group:** |  |  |
| **Tutor:** Mr. Mohamed Saifulamri OMAR |  |  |
| **Date of Submission:** 15 August 2025, 2359hrs |  |  |

|  |  |
| --- | --- |
| **Student Name** | **Student Number** |
| 1. Senthilkumar Dhavasre | S10258427D |
| 2. Chia Eason | S10257196D |

Table of Contents

[**1.0 Problem Statement** 3](#_Toc206193791)

[**1.1** **Description of the cybersecurity issue** 3](#_Toc206193792)

[**1.2** **Motivation and relevance** 4](#_Toc206193793)

[**2.0 Technical Setup and Dataset Description** 4](#_Toc206193794)

[**2.1 Development environment** 4](#_Toc206193795)

[**2.2 Dataset’s Source and structure** 6](#_Toc206193796)

[**3.0 Modelling Process** 7](#_Toc206193797)

[**3.1 Data preprocessing** 7](#_Toc206193798)

[**3.2 Data Modelling** 13](#_Toc206193799)

[**3.3 Model Evaluation** 17](#_Toc206193800)

[**4.0 Application Development** 23](#_Toc206193801)

[**4.1 Application Architecture** 23](#_Toc206193802)

[**4.1.1 Presentation Layer (Streamlit Frontend)** 23](#_Toc206193803)

[**4.1.2 Business Logic Layer (AI Processing Engine)** 24](#_Toc206193804)

[**4.1.3 Data Processing Layer** 24](#_Toc206193805)

[**4.1.4 Model Inference Layer** 25](#_Toc206193806)

[**4.1.5 Performance Optimisation Layer** 25](#_Toc206193807)

[**4.2 Application Development** 25](#_Toc206193808)

[**4.2.1 Initialising Streamlit** 25](#_Toc206193809)

[**4.2.2 Single Prediction Tab** 26](#_Toc206193810)

[**4.2.3 Batch Testing Tab** 33](#_Toc206193811)

[**4.2.4 Email Classification Game Tab** 35](#_Toc206193812)

[**4.2.5 Model Stats Tab** 38](#_Toc206193813)

[**4.3 Application Testing (eg; Unit testing)** 40](#_Toc206193814)

[**4.3.1 Single Prediction Tab Testing** 40](#_Toc206193815)

[**4.3.2 Batch Testing Tab Testing** 42](#_Toc206193816)

[**4.3.3 Email Classification Game Tab Testing** 46](#_Toc206193817)

[**4.3.4 Model Stats Tab Testing** 50](#_Toc206193818)

[**5.0 Application Deployment** 51](#_Toc206193819)

[**5.1 Local Environment Setup** 52](#_Toc206193820)

[**5.2 Cloud Deployment Process** 52](#_Toc206193821)

[**6.0 Summary and Reflection** 53](#_Toc206193822)

[**6.1 Issues faced and how they were resolved** 53](#_Toc206193823)

[**6.1.1 Steep learning Curve with Streamlit Framework** 53](#_Toc206193824)

[**6.1.2 Time Management Constraint** 54](#_Toc206193825)

[**6.1.3 Understanding and Implementing LSTM Models** 54](#_Toc206193826)

[**6.2 Key findings, challenges, and future improvements** 54](#_Toc206193827)

[**7.0 Member Contribution** 55](#_Toc206193828)

[**8.0 Citations** 57](#_Toc206193829)

[**9.0 Appendix** 57](#_Toc206193830)

# **1.0 Problem Statement**

Can deep learning models effectively distinguish between legitimate emails and spam/phishing attacks using advanced natural language processing techniques, providing automated email threat detection for cybersecurity applications?

## **Description of the cybersecurity issue**

Phishing isn’t just an IT department problem anymore, it’s something that affects all of us, every single day we’re online. Globally, there are an estimated 3.4 billion phishing emails sent every day (Ghosh, 2025). Email providers like Google catch a lot of them, over 100 million daily, but the truth is, plenty still get through.

The scary part is they don’t look like those clumsy scam emails from ten years ago. Today’s phishing attacks are polished, professional, and incredibly hard to spot. Criminals use AI to write flawless, convincing messages, fake LinkedIn job offers, and even deepfake voices or videos to impersonate people you trust (Baker & Cartier, n.d.; Ghosh, 2025). Many phishing websites even use HTTPS, so they look “secure” at first glance. Sometimes, the email looks like it’s from Microsoft telling you your account will be locked, or from a delivery company saying your package can’t be shipped unless you click a link.

For individuals, the stakes are high. Clicking one wrong link could mean your bank account gets drained, your personal photos get locked behind ransomware, or your identity gets stolen. Unlike big companies, we don’t have cybersecurity teams monitoring our inboxes. Once you’re tricked, the damage is usually personal, immediate, and expensive.

Awareness and training can make a big difference. In fact, research shows that good phishing training can lead to an 86% drop in incidents (Baker & Cartier, n.d.). But most people never get that training, and spam filters alone aren’t enough, they can be bypassed by something as small as a changed word, a swapped logo, or a slightly altered sender name.

## **Motivation and relevance**

With phishing becoming more sophisticated and harder to detect, there’s an urgent need for solutions that are accessible to everyone, not just those with technical expertise. Most phishing prevention tools are built for businesses, but individuals are often left with little more than a spam filter and their own judgment. Our project bridges that gap.

We’ve developed an AI-powered tool that combines advanced Natural Language Processing (NLP) with numeric clues such as unusual URL counts, odd punctuation, and the use of currency symbols. This hybrid approach allows our system to catch not just obvious scams, but also the subtle, well-crafted phishing attempts that blend in with legitimate emails.

To make it even more effective, we’ve included an interactive learning game within the application. Users can test themselves by identifying which emails are spam or ham, receive instant feedback, and learn tips and tricks for spotting phishing attempts. This gamified training approach keeps people engaged while actively improving their ability to recognize threats, potentially providing the same kind of awareness boost that leads to the 86% reduction in incidents seen in trained environments (Baker & Cartier, n.d.).

By combining cutting-edge AI detection with practical, hands-on learning, our project empowers everyday users to protect themselves in an increasingly dangerous online environment.

# **2.0 Technical Setup and Dataset Description**

## **2.1 Development environment**

*Figure 2.1.1 to Figure 2.1.6 can be found in the Appendix.*

We built our phishing email detection app in Deepnote, which made it really easy for the us to work together. Since it’s a collaborative notebook environment, we could all code, test, and debug in real time without having to constantly send files back and forth. This helped us move faster and keep everyone on the same page.

For deployment, we used Streamlit Cloud to turn our Python model into an interactive web app. The process was pretty straightforward, we created a public GitHub repository with all our code, requirements, and assets, then linked it to our Streamlit Cloud account. From there, every time we pushed an update to GitHub, the app automatically rebuilt and deployed.

At the very start of our application, we initialised Streamlit’s configuration using `st.set\_page\_config()` to define the page title, icon, layout, and sidebar behaviour. This ensured the app loaded with the right branding and user interface from the moment it opened (Figure 2.1.1).

We also made use of `@st.cache\_resource` above each major function. This is a Streamlit feature that stores the results of expensive operations (like loading datasets, preprocessing, or training) in memory. By caching these resources, the app doesn’t have to re-run the entire process every time the page is refreshed, or a user interacts with the interface, improving responsiveness and reducing unnecessary computation.

We also customised the appearance using CSS styling embedded directly via `st.markdown()` with `unsafe\_allow\_html=True`. This allowed us to design visually distinct UI components such as (Figure 2.1.2):

* A gradient main header for the app title.
* Prediction boxes with different colours for spam vs ham results.
* Metric cards for showing model performance stats.
* Feature cards and explanation cards for displaying key word highlights and model reasoning in an easy-to-read format.

Finally, we imported all the required Python libraries for the project, covering (Figure 2.1.3):

* Data processing: `pandas, numpy`, `re`, `scikit-learn` tools.
* Deep learning: `tensorflow.keras` and `keras` layers such as Embedding, LSTM, and Dense.
* Visualisation: `matplotlib`, `seaborn`, and `plotly` for plots and performance charts.
* Model explainability: `lime` and `shap` for interpreting predictions.
* Utility libraries: `os`, `pathlib`, and `warnings`for environment setup and runtime control.

## **2.2 Dataset’s Source and structure**

The dataset used in this project is the Phishing Email Dataset by Naser Abdullah Alam, publicly available on Kaggle. It was compiled by researchers to study phishing email tactics and combines content from multiple well-known datasets to create a comprehensive resource for phishing detection research.

The initial data sources include:

* Enron and Ling Datasets — containing the core content of emails such as subject lines, body text, and spam/legitimate labels.
* CEAS, Nazario, Nigerian Fraud, and SpamAssassin Datasets — offering broader context, including sender and recipient information, timestamps, and classification labels.

In its final form, the dataset merges these sources into a single collection of approximately 82,500 emails, consisting of 42,891 spam/phishing and 39,595 legitimate (ham) messages. This variety provides a wide spectrum of writing styles, layouts, and threat types — from obvious scams to highly convincing phishing messages that mimic trusted brands.

Although the Kaggle dataset already includes a combined CSV called `phishing\_email.csv` with just two columns (`text\_combined` and `label`), we chose not to use it. Instead, we merged the six original CSV files ourselves. The pre-combined version contains extra information like the sender’s address and the time the email was sent, but for our purposes, this data wasn’t necessary, our model focuses purely on the content of the email, not when or by whom it was sent. By starting with the raw files, we could control exactly what went into the final dataset and avoid carrying along irrelevant fields.

That said, the raw CSVs all came in different formats, some had separate subject and body columns, while others had sender details, dates, or extra metadata. To make them usable for modelling, we built a preprocessing step that standardised everything into the same two-column format: one column for the cleaned email text (Message) and one for the label (Category). This standardisation process is explained in detail later in [Section 3.1](#_3.1_Data_preprocessing).

Dataset Link: <https://www.kaggle.com/datasets/naserabdullahalam/phishing-email-dataset/data>

# **3.0 Modelling Process**

## **3.1 Data preprocessing**

|  |
| --- |
| We pulled the Phishing Email Dataset directly from Kaggle using the kagglehub API (Figure 3.1.2). This ensured we had a fresh copy of the dataset each time and knew exactly where it was stored in our working environment. To keep our Streamlit app snappy, we wrapped this loader function with @st.cache\_resource (Figure 3.1.1), so the dataset is downloaded and prepared only once instead of on every page reload.  The dataset comes in several CSV files stored in different folders, including one large pre-combined file named phishing\_email.csv. That file merges all six source datasets into a single table, but it also contains extra fields like sender details and timestamps that we didn’t need. Since our approach was to standardise and merge the raw datasets ourselves, we explicitly filtered this file out during the loading process (Figure 3.1.3).  Finally, the loader returns a list of only the CSVs we want to process, which we store in file\_paths (Figure 3.1.4). This clean set of file paths becomes the starting point for our preprocessing stage. |
| Figure 3.1.1 – Storing session state  Figure 3.1.2 – Load Dataset from Kaggle    Figure 3.1.3 – Filtering file necessary    Figure 3.1.4 – Load load\_data() function |

|  |
| --- |
| Next, we created a function called `load\_and\_standardize\_data` to load everything and give it a consistent structure (Figure 3.1.5). The problem was the files didn’t all look the same, some had single column for the email content called `text\_combined`, others had separate `subject` and `body` columns, and a few were arranged differently.  To fix that, the function checks each file and:   * If there is a `text\_combined `column, it just uses that as a `Message` and renames `label` to `Category` (Figure 3.1.6) * If there is a `subject` and `body`, it merges, them into one string so we keep the full context of the email (Figure 3.1.7) * If neither is there, it falls back to the first text column. (Figure 3.1.8)   For the labels, we try to grab the `label` column if it exists. If not, we just take the last column in the file. After that, we only keep two columns: `Message` that contains the email content and `Category` that tells whether the email is a spam or not (Figure 3.1.9)  Finally, the function is wrapped in a `try/except` block (Figure 3.1.10), so if one file is broken or unreadable, it won’t crash the whole process, it just skips that file and keeps going, making our code robust.  After defining this function, we call it to return `phishing\_dfs`, which is a list of all the cleaned DataFrames. We then combine them into one final dataset using `pd.concat` with `ignore\_index=True` (Figure 3.1.11), giving us a complete, uniform dataset ready for the next stage of processing. |
| Figure 3.1.5 – load\_and\_standardize\_data() function    Figure 3.1.6 – File combination    Figure 3.1.7    Figure 3.1.8    Figure 3.1.9    Figure 3.1.10    Figure 3.1.11 |

|  |
| --- |
| After combining all the files into a single DataFrame, we did data cleaning using the `clean\_data` function (Figure 3.1.11). This function focuses on making sure the dataset doesn’t contain obvious quality issues before modelling.  Inside the function, we:   * Check for missing values by summing up all null values across the dataset using `df.isnull().sum().sum()`. This tells us whether there are any completely empty cells that might need handling. * Check for duplicate rows by counting them with `df.duplicated().sum()`. * Remove duplicates with `drop\_duplicates()` to ensure each email appears only once in our dataset, preventing the model from “double learning” the same example. * Verify the removal by printing the duplicate count again — a quick sanity check that the operation worked.   Finally, the cleaned DataFrame is returned and stored back into `df` for the next stage of processing.  In short, this step ensures we’re working with a clean, unique, and ready-to-use dataset, reducing the risk of biased or redundant training data. |
| Figure 3.1.12 |

|  |
| --- |
| We added an `advanced\_text\_preprocessing` step so the model learns the patterns of spam messages instead of memorising exact words, numbers, or links. For example, if a spam email says, “Claim $2,000 now!!!” with a link, we don’t want the model to think that the number “2000” or that exact URL is always spam.  Before cleaning, we first take note of a few simple things about the messag, like how many links it has, how many exclamation or question marks pop up, how many currency symbols (like $ or ₹) are there, how many number sequences it contains, and how many special characters it uses overall. These little counts become extra hints for the model to work with.  After that, we clean up the text by swapping out any links with the word `URL`, changing all numbers to `NUMBER`, stripping or tidying up punctuation and special characters, and squeezing multiple spaces down to just one. This leaves us with cleaner text that shows the general structure of the message, while those extra counts capture some of the patterns that often show up in spam. |
| Figure 3.1.13 |

## **3.2 Data Modelling**

|  |
| --- |
| To train the classifier, we first sampled 30,000 messages to keep the training loop responsive while we iterated (Figure 3.2.1) and the `random\_state` is set to 42 for reproducibility. Each message will be passed through our `advanced\_text\_preprocessing` function, which produces two outputs: the cleaned text, stored in one DataFrame `df[“Processed\_Message”]`, and the six numeric “spam clue” features such as URL, exclamation, question mark count, stored in another `feature\_df`. Both of these outputs are later used together as inputs to the model.  We then created a stratified 60/20/20 split for train, validation, and test (Figure 3.2.2). Stratification keeps the spam/ham ratio consistent in each set, which means the model sees a fair representation of both classes during training, and our evaluation metrics are more reliable.  Since our model has two separate inputs, `df[“Processed\_Message”]` and `feature\_df`, we split them in parallel. That’s why you see variables like X\_train (processed message) and X\_feat\_train (features) always paired together, along with their matching labels (y\_train). This ensures that the text and feature rows still line up correctly after splitting. |
| Figure 3.2.1 – Sampling and Preprocessing    Figure 3.2.2 – Train/Val/Test Split |

|  |
| --- |
| For the text branch, we used a Keras Tokenizer, which is a tool that turns raw words into numbers so the model can work with them. Neural networks can’t directly understand text, so the tokenizer first builds a dictionary of the most common words in the training data, in our case, we capped it at 20,000 words by using `vocab\_size = 20000` to keep the model size manageable. Each word in this dictionary gets its own unique integer ID. When we feed in new text, the tokenizer replaces each word with its corresponding number, and any word not in the dictionary is replaced with a special “out-of-vocabulary” (<OOV>) token.  We trained the tokenizer only on the training set so that the validation and test sets remain unseen, avoiding data leakage. Using this mapping, we transformed the emails in the training, validation, and test sets into sequences of integers.  Finally, we run these sequences through `pad\_sequences()` so they are all exactly `max\_length` (200 tokens), which matches the fixed input size defined in our model’s LSTM branch. For shorter messages, we pad zeros at the end (padding='post'), and for longer ones, we truncate extra words from the end. This uniform shape is essential because our model architecture is built to process sequences of a fixed length. |
| Figure 3.2.3 – Tokenising the processed message |

|  |
| --- |
| The model uses a two-branch architecture (Figure 3.2.4). The first branch processes the cleaned and tokenised text input. It begins with an Embedding layer, which acts like a lookup table that converts each of the 200 integer tokens into a 128-dimensional dense vector. These vectors capture relationships between words, for example, words used in similar contexts end up having similar vector representations, so the model can learn meaning rather than treating words as unrelated IDs. We set `mask\_zero=True` so that padding tokens (0s added during preprocessing) are ignored and do not influence learning.  The embedded sequence is then passed into an LSTM (Long Short-Term Memory) layer with 64 units. LSTMs are a special type of recurrent neural network designed to capture patterns over time, in this case, the order of words in an email, while avoiding the “vanishing gradient” problem that makes it hard for standard RNNs to learn long-term dependencies. By using both dropout and `recurrent\_dropout`, the LSTM also reduces overfitting by randomly ignoring some inputs and recurrent connections during training.  The second branch takes the seven numeric “spam clue” features extracted during preprocessing. These go through a dense layer with 32 units and ReLU activation to learn non-linear relationships, followed by a dropout layer for regularisation.  Finally, the outputs of both branches are concatenated so the model can combine the semantic patterns learned from the text with the behavioural patterns in the numeric features. This combined representation passes through another dense layer with 64 units and ReLU activation, followed by dropout, before reaching the final output layer, a single sigmoid neuron that produces a probability score indicating whether the email is spam. |
| Figure 3.2.4 – Two-branch model |

|  |
| --- |
| Once the architecture was defined, we compiled the model using the Adam optimiser with a learning rate of 0.0005, binary cross-entropy loss, and accuracy as the primary evaluation metric (Figure 3.2.5). Adam was chosen for its adaptive learning rate, which makes training more stable and efficient across different parameters. Binary cross-entropy is suitable here because this is a binary classification problem, the model outputs a probability between 0 and 1 for each email, and the loss function measures how close these predictions are to the actual labels. Tracking accuracy allows us to monitor how many predictions are correct overall during training.  To avoid overfitting, we implemented early stopping by monitoring the validation loss. If the validation loss did not improve for three consecutive epochs (`patience=3`), training stopped early. Additionally, `restore\_best\_weights=True` ensured that the model reverted to the weights from the epoch with the lowest validation loss, rather than using the weights from the last epoch trained.  We then trained the model for a maximum of 15 epochs with a batch size of 64. During training, the model received two inputs, the padded text sequences from the tokeniser and the numeric “spam clue” features, along with their associated labels. The validation set, which the model never trained on, was used each epoch to evaluate generalisation performance. Early stopping was applied during this process, meaning the model might stop before 15 epochs if no further improvements were detected on the validation data. |
| Figure 3.2.5 – Compiling & Training the Model |

## **3.3 Model Evaluation**

|  |
| --- |
| After training, we evaluated our model’s performance on the unseen test set (Figure 3.3.1). We first used `model.evaluate()` on `[X\_test\_padded, X\_feat\_test]` with `y\_test` to get the test loss and test accuracy. This gives us a quick overview of how well the model performs in general on data it has never seen before.  Next, we computed more detailed classification metrics:   * Accuracy – the proportion of all emails (spam + ham) the model correctly classified. * Precision – out of all the emails the model flagged as spam, how many were actually spam. * Recall – out of all the actual spam emails, how many the model successfully caught. * F1-score – the harmonic mean of precision and recall, balancing the two.   We also stored the predicted labels (`y\_pred`) and the raw prediction probabilities (`y\_pred\_prob`) so we could use them later for further analysis or plotting ROC curves. Finally, we packaged all relevant objects, including the trained model, tokenizer, training history, processed datasets, and evaluation metrics, into a single dictionary, making it easy to reference everything from one place in the Streamlit app and other parts of the project.  In this step (Figure 3.3.2), we unpacked all relevant outputs from the `model\_data` dictionary returned by the training function. This includes the trained model, tokenizer, dataset splits, predictions, and evaluation metrics. Having these variables directly accessible makes it easier to perform further analysis, visualisations, and integrate the model into the Streamlit app without having to retrain it. |
| Figure 3.3.1 – Model Evaluation    Figure 3.3.2 |

|  |
| --- |
| Once we had the model trained, we wanted to see how it actually performed on new data it hadn’t seen before. First, we ran a classification report (Figure 3.3.3) on the test set. This gave us precision, recall, and F1-score for both spam and ham. Precision tells us out of all the emails the model flagged as spam, how many were actually spam. Recall is the opposite view out of all the real spam emails, how many did the model successfully catch. The F1-score is just a balance between the two. We also cached this in Streamlit so it wouldn’t keep recalculating every time we refreshed the app.  Next, we plotted a confusion matrix (Figure 3.3.4). This is like a scoreboard for predictions, it shows how many emails we got right (both spam and ham) and how many we got wrong. It’s split into four boxes: true negatives (ham we got right), false positives (ham we wrongly called spam), false negatives (spam we missed), and true positives (spam we caught). Displaying it as a heatmap made it much easier to spot where the model might be struggling.  We also examined the training loss vs validation loss curves (Figure 3.3.5) to check if the model was overfitting. These curves show how well the model was learning on the training data compared to unseen validation data over time. If the validation loss started going up while the training loss kept going down, that would be a sign of overfitting. We also printed out the lowest validation loss achieved, marking the point where the model performed best before any decline.  Finally, we plotted the ROC curve and calculated the AUC score (Figure 3.3.6). The ROC curve shows how well the model can distinguish between spam and ham at different thresholds, and the AUC score gives this ability a single number. The closer it is to 1.0, the better the model is at separating the two classes. This is also useful if we want to adjust the decision threshold for example, to catch more spam at the risk of flagging a few more ham emails. |
| Figure 3.3.3    Figure 3.3.4  Figure 3.3.5  Figure 3.3.6 |

# **4.0 Application Development**

## **4.1 Application Architecture**

The AI-Powered email spam detection systems employ a comprehensive multi-tier hybrid architecture that combines deep learning capabilities with feature engineering to deliver an enterprise grade performance and scalability. The architecture is designed following modern software engineering principles to ensure maintainability, extensibility, and optimal user experience. There are 5 layers that we will focus on:

1. Presentation Layer
2. Business Logic Layer
3. Data Processing Layer
4. Modern Inference Layer
5. Performance Optimization Layer

### **4.1.1 Presentation Layer (Streamlit Frontend)**

The user interface is built using Streamlit which provides an intuitive web-based interface that allows real-time interaction with the AI model. This layer shows a responsive multi-tab interface with four modules:

1. Single Prediction
   1. Individual email analysis
2. Batch Testing
   1. Processing multiple email simultaneously
3. Interactive Training
   1. Game for educational purposes
4. Model Statistics
   1. For performance monitoring

The presentation layer incorporates real-time visualisation using dynamic charts and gauges using Plotly, along with custom CSS styling that provides a professional gradient-based design with colour coded prediction results to enhance user experience and usability.

### **4.1.2 Business Logic Layer (AI Processing Engine)**

The core intelligence resides in a hybrid neural network architecture that processes both textual content and engineered features at the same time. Our deep learning component uses a bidirectional LSTM network with 64 units and dropout regularisation. This is combined with 128-dimensional word embeddings for sequential text processing. This is added on with a feature engineering component that extracts seven multi-dimensional features such as URL count, exclamation patterns, currency symbols, and capitalization ratios that are processed through a 32-unit dense neural network. The fusion between the LSTM outputs and processed features through a concatenation layer enables the system to leverage both contextual understanding and explicit feature patterns for superior classification accuracy.

### **4.1.3 Data Processing Layer**

The data pipeline uses advanced text processing with intelligent feature extraction to ensure optimal model performance. The text normalisation pipeline handles lowercasing, URL replacement, special character processing, and number tokenisation. The Keras Tokenizer manages out-of-vocabulary tokens and sequence padding for consistent input formatting. The feature engineering engine performs real-time extraction of behavioural and structural email characteristics. This includes linguistic patterns and metadata analysis. Comprehensive data validation ensures input sanitation and robust error handling. Thus, this makes the system suitable for production development with various email formats and potential edge cases.

### **4.1.4 Model Inference Layer**

The prediction engine provides multi-faceted threat assessment through sophisticated analysis mechanisms. Probabilistic classification using sigmoid activation produces confidence scores between 0-1, while adaptive thresholding allows configurable decision boundaries for different organizational security requirements. The risk assessment engine automatically categorizes threats into High, Medium, or Low levels based on probability scores and feature analysis. Additionally, the layer integrates LIME-based explainable AI capabilities that generate human-readable explanations for prediction decisions, enhancing transparency and enabling security analysts to understand the reasoning behind threat classifications.

### **4.1.5 Performance Optimisation Layer**

Our system uses Streamlit’s caching mechanisms persist models and processed data across user sessions. It also has vectorized batch processing operations enabling high-throughput analysis of multiple emails simultaneously. Moreover, the implementation of early stopping during model training prevents overfitting while reducing the computational overhead. Thus, the modular architecture allows for horizontal scaling and integration with existing cybersecurity infrastructure.

## **4.2 Application Development**

### **4.2.1 Initialising Streamlit**

|  |
| --- |
| From the figure below, we can see that the `st.set\_page\_config()` function initialise the web application with essential parameters such as page title, security icon, and wide layout for optimal display. The header section uses `st.markdown()` with custom HTML to generate a title with gradient styling and subtitle. The `st.sidebar` context manager creates a dedicated information panel containing real-time model metrics through `st.metric()`components that display performance indicators like 94.1% accuracy, precision, and recall scores, along with an interactive `st.slider()` widget for adjusting spam detection threshold sensitivity. Finally, `st.tabs()` generates four separate interface pages - Single Prediction, Batch Testing, Email Classification Game, and Model Stats - allowing users to navigate between different application functionalities while maintaining the sidebar's persistent model information and threshold controls across all tabs. |

### 

### **4.2.2 Single Prediction Tab**

#### **4.2.2.1 Function: predict\_spam\_message()**

The `predict\_spam\_message()` function serves as the core intelligence engine that orchestrates the complete email analysis workflow by integrating preprocessing, model prediction, and result interpretation. This function transforms raw email content into comprehensive threat assessments through a multi-stage pipeline that combines deep learning capabilities with intelligent risk factor analysis to provide users with actionable security insights.

*Refer to Figure 4.2.2.1.3 & Figure 4.2.2.1.4 in Appendix for full function code*

|  |
| --- |
| **Input Processing and Model Prediction (Lines 452-480)**  The function begins with input validation and performance tracking through time.time(), then leverages the preprocessing pipeline to extract both cleaned text and behavioral features. The tokenization process converts text to numerical sequences with padding, while the feature array organizes seven key characteristics for the hybrid model. The core prediction feeds both inputs to the LSTM model simultaneously, producing probability scores between 0-1. |
| Figure 4.2.2.1.1 – predict\_spam\_message() function, Line 448 - 480 |

|  |
| --- |
| **Risk Analysis and Result Compilation (Lines 482-510)**  The post-prediction analysis applies configurable threshold logic for classification and calculates confidence scores. Intelligent risk factor assessment evaluates specific threat indicators including multiple URLs, excessive punctuation, currency symbols, and high capitalization ratios. The comprehensive result dictionary packages all analysis components including binary prediction, confidence metrics, processing time, and automated threat level categorization. Robust error handling ensures system stability while providing user-friendly feedback, transforming raw email content into actionable security intelligence for informed decision-making. |
| Figure 4.2.2.1.2 – predict\_spam\_message() function, Line 479 - 510 |

#### **4.2.2.2 Function: explain\_prediction()**

This function uses explainable AI capabilities using Local Interpretable Model-agnostic Explanation (LIME) to provide users with transparent insights into the model’s decision-making process. It addresses the critical need for AI transparency which allows the user to understand why specific emails are classified as spam or safe. The implementation creates a complex wrapper around the LSTM model that allows LIME to perform perturbation analysis on email text.

*Refer to Figure 4.2.2.2.3 in Appendix for full function code*

|  |
| --- |
| **Model Wrapper for LIME Integration (Lines 516-538)**  LIME requires a function that can process multiple text variations and return class probabilities. However, the model expects both text sequence and feature arrays. The nested `predictor\_fn()` solves the integration challenge by processing each text variant through the complete preprocessing pipeline which extract both cleaned text and behavioural features. It then feed both inputs to the model simultaneously. The `np.column\_stack([1-predictions, predictions])` formatting ensures LIME receives the required probability distribution for both classes. |
| Figure 4.2.2.2.1 – explain\_prediction() function, Lines 516 - 537 |

|  |
| --- |
| **LIME Explainer Configuration (Lines 540-544)**  The LIME configuration optimises explainability for email content analysis through carefully chosen parameters. The `split\_expression=r’\W+’` uses regex to split text on non-word characters. This ensures precise word-level analysis that captures meaningful linguistic units. Moreover, setting the `bow=False` maintains word order during perturbation. This preserves contextual relationships critical for understanding email semantics. The user-friendly class names ['Safe Email', 'Spam Email'] ensure explanations are immediately comprehensible to non-technical users. |
| Figure 4.2.2.2.2 – explain\_prediction() function, Lines 540 - 544 |

|  |
| --- |
| **Explanation Generation and Error Handling (Lines 546-556)**  The explanation generation process uses `num\_features=10` to focus on the mot impactful words, `num\_samples=500` for sufficient perturbation samples, and cosine distance metrics for semantic similarity measurement between original and perturbed texts. Comprehensive error handling prevents system failures during LIME processing while providing informative feedback when explanations cannot be generated. This explainable AI implementation transforms the neural network into an interpretable system that builds user trust and enables informed decision-making in email security assessment. |
| A screen shot of a computer  AI-generated content may be incorrect.  Figure 4.2.2.2.3 – explain\_prediction() function, Lines 546 - 556 |

#### **4.2.2.3 Function: create\_user\_friendly\_explaination()**

This function transforms raw LIME analysis into a user comprehensible explanation that allows non-technical users to understand AI decision making processes. This function bridges the gap between complex machine learning outputs and practical user interface elements by converting numerical impact scores into visual indicators, descriptive text, and categorised threat assessments.

*Refer to Figure 4.2.2.3.4 in Appendix for full function code*

|  |
| --- |
| **Data Extraction and Processing Loop (Lines 560-572)**  Lines 560-561 implement input validation to handle null explanations. Line 564 extracts LIME’s word-impact pairs using `explanation.as\_list()`, while lines 566-572 initialise the results list and process each word through cleaning and filtering. Line 570 normalises words with `word.strip().lower()`. Lastly, lines 571-582 filter out meaningless short words to focus on substantial linguistic elements. |
| Figure 4.2.2.3.1 – create\_user\_friendly\_explanation() function, Lines 560 - 572 |

|  |
| --- |
| **Impact Classification and Visual Mapping (Lines 574-583)**  Lines 575 – 583 uses threshold-based classification where positive values indicate spam signals (lines 576-578) with red/orange/blue colour coding and warning emojis. On the other hand, negative values suggest safe content (line 580 – 582) with green schemes and checkmark emojis. Line 585 categorises strength level using `abs(impact) values. Thus, providing intuitive understanding of feature importance. |
| Figure 4.2.2.3.2 – create\_user\_friendly\_explanation() function, Lines 574 - 585 |

|  |
| --- |
| **Result Dictionary Compilation (Lines 587-597)**  Lines 587-595 package each word’s analysis into a dictionary that contains cleaned word, numerical impact score, descriptive direction, strength category, visual emoji, colour coding, and human-readable summary. Line 597 returns the complete results list which allows user interface to present complex AI explanations in an accessible visual and textual elements that enhance user trust and decision-making capabilities. |
| Figure 4.2.2.3.3 – create\_user\_friendly\_explanation() function, Lines 587 - 496 |

#### **4.2.2.4 Single Prediction Tab Front End**

Tab 1 integrates all previously discussed functions into a cohesive single email analysis interface. Lines 745-756 implement dual input methods through st.radio() selection between manual entry and predefined example messages containing representative phishing, scam, legitimate, and marketing emails for demonstration purposes.

*Refer to Figure 4.2.2.4.4 to Figure 4.2.2.4.7 in Appendix for full function code*

|  |
| --- |
| **Analysis Execution and Result Display (Lines 739-743)**  The workflow triggers on button click, validates input, and calls `predict\_spam\_message()` with the user-configured threshold, displaying processing feedback through `st.spinner()`. |
| Figure 4.2.2.4.1 – Single Prediction Tab Front End, Lines 739 - 743 |

|  |
| --- |
| **Result Display and Visualization (Lines 745-798)**  Lines 779-780 determine CSS styling and emojis based on threat levels, while lines 790-801 create a five-column metrics layout. The Plotly gauge chart (lines 810-831) provides visual probability assessment with color-coded threat ranges. |
| A screen shot of a computer program  AI-generated content may be incorrect.  Figure 4.2.2.4.2 – Single Prediction Tab Front End, Lines 745 - 798 |

|  |
| --- |
| **AI Explanations Integration (Lines 801-810)**  When enabled, the system chains explain\_prediction() and `create\_user\_friendly\_explanation()` functions to generate word-level impact analysis, spam/safe indicator categorization, and detailed explanation tables, completing the comprehensive email threat analysis workflow. |
| Figure 4.2.2.4.3 – Single Prediction Tab Front End, Lines 801 - 810 |

### **4.2.3 Batch Testing Tab**

This tab implements high-volume email analysis capabilities that leverage the `predict\_spam\_message()` function for processing multiple emails simultaneously. The interface handles file upload, parsing, batch analysis, and comprehensive result visualisation without requiring additional custom functions beyond the core prediction engine.

*Refer to Figure 4.2.3.1 to Figure 4.2.3.5 in Appendix for full function code*

|  |
| --- |
| **File Processing and Validation (Lines 895-914)**  We use `st.file\_uploader()` to accept TXT files, processes uploads through `io.StringIO()` for UTF-8 decoding and parses content line-by-line with automatic empty line removal. This creates a clean email list ready for batch processing. |
| Figure 4.2.3.1 – Batch Testing Front End, Lines 895 – 914 |

|  |
| --- |
| **Batch Analysis Loop and Progress Tracking (Lines 935-960)**  The core processing loop calls `predict\_spam\_message()` for each email, implementing real time progress tracking through `st.progress()` and dynamic status updates. The results are placed into dictionaries containing prediction metrics, with `st.storage\_state` storage for persistence. The system include performance warning for large datasets (>1000 emails) and calculates statistical summaries including spam detection rates and processing times. |
| Figure 4.2.3.2 – Batch Testing Front End, Lines 934 - 960 |

|  |
| --- |
| **Visualization and Export Features (Lines 1004-1066)**  We generate styled pandas DataFrames with colour coded classifications (red = spam, green = safe), implements CSV export with timestamped filenames, and creates interactive Plotly visualisation such as pie charts for classification distribution and bar charts for threat level analysis. This comprehensive batch processing architecture efficiently handles large-scale email analysis while maintaining detailed reporting capabilities. |
| Figure 4.2.3.3 – Batch Testing Front End, Lines 934 – 960    Figure 4.2.3.4 – Batch Testing Front End, Lines 934 - 960 |

### **4.2.4 Email Classification Game Tab**

#### **4.2.4.1 Function: get\_random\_test\_email()**

This is an interactive game component that incorporate educational functionality that tests user ability to identify spam email against AI performance. This gamified approach enhances cybersecurity awareness training through hands-on email classification practice with immediate feedback and performance comparison metrics.

*Refer to Figure 4.2.4.1.2 in Appendix for full function code*

|  |
| --- |
| This function provides curated training examples for the interactive learning game by creating dictionaries of spam emails and safe emails. The function uses `random.choice()` to select a category (spam or safe). It proceeds to picks an email from that category’s list and category information after selecting the category. This allows the game interface to validate user predictions against known correct answers while providing educational examples that demonstrate clear spam and safe email characteristics. |
| A screenshot of a computer program  AI-generated content may be incorrect. |
| Figure 4.2.4.1.1 – get\_random\_test\_email() function, Lines 599 - 632 |

#### **4.2.4.2 Function: game\_explanation()**

This function enhances user learning by providing intelligent explanations for email classifications, helping users understand spam detection patterns improve their cybersecurity awareness through contextual analysis of email characteristics.

*Refer to Figure 4.2.4.2.3 in Appendix for full function code*

|  |
| --- |
| **Explanation Dictionary Setup (Lines 640-656)**  Line 636 – 637 extract the correct classification and email content for analysis, while lines 640 – 656 define comprehensive explanation directories containing educational descriptions for spam and safe email characteristics. |
| A screenshot of a computer program  AI-generated content may be incorrect. |
| Figure 4.2.4.2.1 – game\_explanation() function, Lines 639 - 656 |

|  |
| --- |
| **Pattern Detection and Analysis (Lines 659-677)**  The pattern detection logic converts email content to lowercase and uses keyword matching to identify specific indicators. This includes urgent language detection that searches for words like ‘urgent’, ‘immediately’, ‘now’. Moreover, we included It a money offer detection that looks for ‘won’, ‘prize’, ‘money’, ‘$’, ‘free’. It also has suspicious link detection that identifies ‘http’ or ‘click’ references. Lastly, It has punctuation analysis that counts exclamation marks |
| Figure 4.2.4.2.2 – game\_explanation() function, Lines 658 - 680 |

#### **4.2.4.3 Email Classification Tab Front End**

This tab implements an educational game and tests users’ spam detection skills against AI performance while providing real-time feedback and learning opportunities. The game integrates both `get\_random\_test\_email()` and `game\_explanation()` functions along with the core `predict\_spam\_message()` function to create a comprehensive cybersecurity training experience with session state management for persistent gameplay.

*Refer to Figure 4.2.4.1 to Figure 4.2.4.4 in Appendix for full function code*

|  |
| --- |
| **Game State Initialization and Management (Lines 1157-1178)**  Line 1157 – 1163 initialises session state variable for game tracking including user score, AI score, current round number, and game status flags. Lines 1174 – 1178 handle new round initiation by calling get\_random\_test\_email() to provide curated training examples and resetting answer status for the current round. |
| Figure 4.2.4.3.1 – Email Classification Tab Front End, Lines 1156 - 1178 |

|  |
| --- |
| **Interactive Classification Interface (Lines 1191-1198)**  This code creates the binary classification interface with unique button keys per round to prevent state conflicts. The column layout provides clear visual separation for user decision making. |
| Figure 4.2.4.3.2 – Email Classification Tab Front End, Lines 1191 - 1198 |

|  |
| --- |
| **AI Prediction and Scoring Logic (Lines 1204-1216)**  Lines 1204 – 1206 call `predict\_spam\_message()` for AI classification using the same threshold settings. Lines 1209 – 1216 implements scoring logic that evaluates both user and AI accuracy against know correct labels. |
| Figure 4.2.4.3.3 – Email Classification Tab Front End, Lines 1204 - 1216 |

### **4.2.5 Model Stats Tab**

This tab implements a model performance dashboard that provides comprehensive statistical analysis, visualisation, and insights into the AI system’s capabilities. This analytics interface leverages pre-calculated model metrics and training history data to present detailed performance assessments without requiring additional custom functions beyond standard data to preprocessing and visualisation libraries.

*Refer to Figure 4.2.5.2 to Figure 4.2.5.6 in Appendix for full function code*

|  |
| --- |
| This tab implements a model performance dashboard that provides statistical analysis and visualisation of our AI model capabilities through executive summary cards, detailed configuration tables, training history plots, and evaluation metrics. The interface displays key performance indicators such as 94.1% accuracy, AUC scores, F1 metrics, and dataset statistics using gradient styled HTML cards and interactive Plotly visualisation. The dashboard processes training history data to create loss curves showing model convergence, implements ROC curve analysis with automatic performance categorisation (Excellent/Good/Fair/Poor). Moreover, it also generates confusion matrix heatmaps for detailed classification breakdown. |
| A screenshot of a computer program  AI-generated content may be incorrect.  Figure 4.2.5.1 – Model Stats Tab Front End, Lines 1297 - 1350 |

## **4.3 Application Testing (eg; Unit testing)**

The application testing phase implemented a comprehensive multi-layered approach to ensure robust functionality, accuracy, and user experience across all system components. The testing focus on validating core AI prediction capabilities, user interface responsiveness, system performance under various load conditions, and educational feature effectiveness. The testing methodology encompassed unit testing of individual functions, integration testing of component interactions, user interface validation, performance benchmarking and accuracy verification against known datasets.

### **4.3.1 Single Prediction Tab Testing**

|  |
| --- |
| **predict\_spam\_message() Function Validation**  We validated the core prediction engine's accuracy, error handling, and performance across diverse input scenarios.  **Test Cases Implemented:**   * Empty string inputs (boundary condition testing) * Standard legitimate business emails * Known spam/phishing examples * Edge cases with special characters and multilingual content * Performance testing with varying email lengths (10-5000+ characters)   The function demonstrated robust performance across all test scenarios. Empty inputs correctly returned None values, preventing system crashes. Processing times remained consistently under 200ms for emails up to 2000 characters, with legitimate emails achieving 95%+ confidence levels and spam emails correctly flagged with appropriate threat classifications.  The legitimate email test case "Are you attending the meeting on sunday?" achieved optimal results with 95.7% confidence HAM classification, 4.3% spam probability, and 130.5ms processing time, demonstrating excellent accuracy and performance characteristics. |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.3.1.1 – Single Prediction Tab |

|  |
| --- |
| **AI Explanation System Testing**  We verified LIME-based explanation functionality provides meaningful, accurate insights into model decision-making processes.  The AI explanation system underwent comprehensive testing with the `explain\_prediction()` and `create\_user\_friendly\_explanation()` functions processing various email types to generate interpretable word-level impact analysis.  The explanation system successfully identified contextual language patterns, correctly weighting business terminology as safety indicators. For the test email, the system identified 6 strong safety signals including "meeting", "sunday", "attending" with negative impact scores (-0.415 to -0.117), while appropriately flagging minimal spam indicators.  The detailed word analysis table provided comprehensive breakdowns with technical impact scores and plain English explanations, successfully bridging the gap between complex AI decision-making and user comprehension. |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.3.1.2 – Single Prediction Tab  A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.3.1.3 – Single Prediction Tab |

### **4.3.2 Batch Testing Tab Testing**

|  |
| --- |
| **File Upload and Data Processing Validation**  We Validated the system's capability to handle multiple email processing through file uploads with accurate parsing, progress tracking, and comprehensive result analysis.  A controlled test dataset containing 4 emails was prepared in TXT format (329.0B file size) representing diverse email types including legitimate business communications and known spam patterns. The test file included:   * Professional meeting follow-up emails * Prize/lottery scam notifications * Business scheduling communications * Inheritance fraud attempts   The system successfully processed the TXT file upload, correctly identifying 4 emails with automatic line-by-line parsing. The preview functionality displayed the first 5 emails as expected, showing proper content extraction and formatting. File validation confirmed appropriate handling of the 200MB size limit and TXT format restrictions. |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.3.2.1 – Batch Testing Tab |

|  |
| --- |
| **Batch Analysis Performance Testing**  We evaluated system performance, accuracy, and user interface responsiveness during multi-email processing operations.  Performance Metrics Achieved:   * Total Processing Time: 0.3 seconds for 4 emails * Average Processing Time: 76.8ms per email * Classification Accuracy: 100% (all emails correctly classified) * System Responsiveness: Real-time progress tracking with completion notifications   The batch processing achieved perfect classification accuracy with the test dataset:   * Spam Detection * 2/4 emails correctly identified as spam (50% detection rate) * Safe Email Classification * 2/4 emails correctly classified as legitimate (50% accuracy) * Confidence Levels * 100% confidence achieved for all classifications * Threat Assessment * Appropriate threat level assignment (High for spam, Low for legitimate) |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.3.2.2 – Batch Testing Tab |

|  |
| --- |
| **Result Analysis and Visualization Testing**  The results table provided comprehensive analysis including email previews, classifications, confidence percentages, threat levels, risk factor counts, and processing metrics. Color-coded formatting successfully distinguished spam (red highlighting) from legitimate emails (green highlighting) for immediate visual assessment.  The analysis summary correctly calculated and displayed:   * Total emails processed: 4 * Classification distribution: 50% spam, 50% safe * High threat emails: 2 (appropriately identified) * Average processing time: 76.8ms (excellent performance)   The dual-chart visualization system successfully rendered:   * Pie Chart * Accurate 50/50 distribution between spam and safe classifications * Bar Chart * Correct threat level distribution showing 2 low-threat and 2 high-threat emails * Export Functionality * CSV download capability with timestamped filenames |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.3.2.3 – Batch Testing Tab |

### **4.3.3 Email Classification Game Tab Testing**

|  |
| --- |
| **Gamified Learning Interface Validation**  We evaluated the educational game functionality, session state management, scoring accuracy, and AI explanation system integration for cybersecurity awareness training.  The interactive game testing involved multiple rounds of classification challenges using both legitimate and spam email examples generated by the get\_random\_test\_email() function. Testing validated user interaction handling, AI prediction integration, and educational feedback systems.  The game interface successfully initialized with proper session state management, displaying Round 1 with 0-0 scores for both user and AI. The email presentation ("Team lunch is scheduled for Thursday at 12:30 PM at the Italian restaurant downtown") provided clear, readable content for classification decisions. Binary choice buttons functioned correctly with appropriate styling and responsiveness. |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.3.3.1 – Email Classification Game Tab |

|  |
| --- |
| **Correct Classification Response Testing**  Test Case 1: Successful User Classification  When users correctly identified the legitimate business email as "SAFE," the system properly validated the response and provided comprehensive feedback:   * User Result: * Correct classification with green highlighting * AI Performance * 100% confidence SAFE classification * Educational Content * Contextual explanations including "professional, courteous language" and "specific, verifiable information" * Performance Tracking * Both user and AI achieved 100% accuracy * Motivational Feedback * "You're tied with the AI! Impressive performance!" |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.3.3.2 – Email Classification Game Tab |

|  |
| --- |
| **Game Progression and State Management**  The game successfully progressed to Round 2 with proper score tracking (User: 1, AI: 1) and generated new email content for continued testing. The get\_random\_test\_email() function provided diverse content including academic communications ("The research paper submission deadline has been extended to March 31st"). This validates the session state is working.  The system maintained game state across rounds while generating fresh content for each classification challenge, ensuring educational value through varied email examples and sustained user engagement. |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.3.3.3 – Email Classification Game Tab |

|  |
| --- |
| **Error Handling and Learning Feedback Testing**  Test Case 2: Incorrect User Classification  When users incorrectly classified a spam email as "SAFE," the system provided comprehensive error analysis and learning opportunities:   * Error Recognition * Clear indication of incorrect user classification with red highlighting * AI Superiority Display   + AI correctly identified SPAM with 100% confidence * Educational Explanations * Detailed breakdown using `game\_explanation()` function results including "urgent language," "unrealistic money or prizes," and "excessive punctuation" * Threat Analysis   + Advanced threat indicators including "urgency pressure tactics," "money/prize offers," and "excessive exclamation marks" * Performance Impact   + User accuracy dropped to 66.7% while AI maintained 100% accuracy * Motivational Support   + "The AI is ahead but keep trying! You're learning valuable skills!" |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.3.5.4 – Email Classification Game Tab |

### **4.3.4 Model Stats Tab Testing**

|  |
| --- |
| **Comprehensive Performance Metrics Validation**  We validated the model statistics dashboard functionality, data visualization accuracy, and performance analytics presentation for stakeholder reporting and system assessment.  The model performance dashboard successfully validated all analytics components through comprehensive testing of executive summary cards, configuration tables, training visualizations, and automated assessment systems. The dashboard displayed exceptional performance metrics including 97.9% accuracy, 0.993 AUC score, 0.979 F1 score across 164,552 training samples, with detailed configuration parameters showing hybrid LSTM architecture, proper dataset splits (18,000 training, 6,000 validations, 6,000 test), and optimized hyperparameters.  Training history visualizations demonstrated effective convergence over 12 epochs with early stopping, while ROC curve analysis achieved "Excellent" 0.993 AUC rating and confusion matrix results showed robust classification performance with 2,798 true negatives, 78 false positives, 51 false negatives, and 3,073 true positives.  The automated performance assessment system correctly identified model strengths including high accuracy, excellent precision, outstanding recall, and exceptional ROC performance, concluding "Model performance is excellent!" and validating the system's readiness for production deployment and stakeholder reporting requirements. |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.3.4.1 – Model Stats Tab |

# **5.0 Application Deployment**

This section documents the complete process of porting our AI-power email spam detection application from a local deployment environment to Streamlit Cloud, transforming it into a publicly accessible web application.

## **5.1 Local Environment Setup**

Our AI email spam detection application was initially developed and tested in a local Python environment. The application consists of a main Streamlit file (EATC.py) containing 1,593 lines of code implementing the complete machine learning pipeline, user interface, and interactive features. The local setup included all necessary Python dependencies such as TensorFlow, pandas, scikit-learn, and Streamlit, along with our phishing email dataset (phishing\_emails\_combined.csv). The application was fully functional locally, running successfully with the command Streamlit run EATC.py and providing all four main features: single email prediction, batch processing, interactive training game, and model statistics visualization.

## **5.2 Cloud Deployment Process**

To go from local deployment to cloud deployment, we used Streamlit Cloud to host the application. These are the steps to host it on Streamlit Cloud:

* **Step 1: GitHub Repository Creation**
  + We created a public GitHub repository named "project" to host our application code and enable cloud deployment through Streamlit's GitHub integration.
* **Step 2: Source Code Upload**
  + The main source code file (EATC.py) was uploaded to the repository, ensuring all 1,593 lines of code were properly transferred to the cloud platform.
* **Step 3: Requirements Configuration**
  + We created a requirements.txt file listing all necessary Python modules for the cloud environment, including streamlit, pandas, numpy, tensorflow, scikit-learn, matplotlib, seaborn, plotly, kagglehub, lime, and shap.
* **Step 4: Deploy Button Activation**
  + The deployment was initiated by clicking the "Deploy" button within our local Streamlit application interface, which redirected us to the Streamlit Cloud platform.
* **Step 5: Streamlit Account Setup**
  + We created a Streamlit Cloud account and connected it to our GitHub repository, allowing the platform to access our code and automatically provision the necessary cloud infrastructure.
* **Step 6: Automated Deployment Process**
  + The system automatically cloned our repository, set up a Python environment, installed dependencies from requirements.txt, and launched the application.
* **Step 7: Issue Resolution and Final Deployment**
  + During deployment, we encountered compatibility issues with TensorFlow 2.13.0 and Python 3.13.5. We resolved this by removing version pinning from requirements.txt, allowing automatic selection of compatible versions. The deployment completed successfully, making our application publicly accessible at <https://xensn-project-eatc-aorkuz.streamlit.app/> with full functionality preserved.

*Refer to Figure 5.2.1 to Figure 5.2.4 in Appendix for setup process images*

# **6.0 Summary and Reflection**

## **6.1 Issues faced and how they were resolved**

### **6.1.1 Steep learning Curve with Streamlit Framework**

One of the primary challenges we faced was trying to understand Streamlit’s unique approach to web application development. The declarative programming model, state management, and widget interactions required us to take a lot of time to understand properly. Initially, we found it challenging to implement multi-tab interfaces, file uploads, and interactive visualisations as we were unfamiliar with Streamlit’s architecture. To overcome this, we took time to learn Streamlit through its documentation, followed Youtube tutorials and experimented with different implementations. Through persistent practice and iterative development, we slowly understood how to create responsive interface and integrating machine learning models effectively.

### **6.1.2 Time Management Constraint**

The project timeline coincided with several other major assignments, creating significant time pressure and competing priorities which might end up harming our project quality. We needed to balance the intensive requirement of developing an AI application while maintaining progress on other coursework proved extremely challenging. Our initial approach of juggling between multiple assignments at once led to fragmented progress and inefficient development cycles. So, to address this, we decided to identify concentrated blocks of days dedicated exclusively to EATC. This approach eliminated context switching overhead and ensure rapid iterative development. Thus, it helped us to complete the application while maintaining quality standards.

### **6.1.3 Understanding and Implementing LSTM Models**

One of significant challenge arose when we wanted to use LSTM-based neural networks for email classification. It was difficult to understand the complex process of text tokenization and sequence modelling for natural language processing. The concepts of converting text to numerical sequences, managing vocabulary sizes, handling out of vocabulary tokens, and properly structuring padded sequences for LSTM input proved conceptually difficult. Additionally, using the tokenizer with the LSTM architecture and ensuring proper data flow through the neutral network required deep understanding of both NLP preprocessing and deep learning fundamentals. We addressed this through comprehensive research into TensorFlow’s documentation, learning LSTM principles through online articles, and experimenting with different tokenization parameters. Through the experimentation and continuous learning, we were able to develop the tokenization pipeline that effectively converts email text into properly formatted sequences for LSTM processing.

## **6.2 Key findings, challenges, and future improvements**

Our email spam detection system achieved exceptional performance with 97.9% accuracy, which portrays the effectiveness of our LSTM neutral networks. Our key findings revealed that the hybrid architecture successfully leveraged both contextual understanding through text processing and behavioural patterns through feature analysis. The major challenges encountered included mastering Streamlit’s framework complexities, managing competing academic prioritises, and implementing NLP tokenization processes for LSTM integration. Some future improvements that can be done are focusing on expanding the dataset to include more diverse phishing techniques, implementing real-time learning capabilities to adapt to emerging threats, integrating with popular email clients for seamless development, and enhancing the explainable AI components to provide even more detailed threat analysis. Additionally, incorporating multi-language support and developing mobile optimised interfaces would significantly broaden the application’s accessibility and impact in global cybersecurity awareness training.

# **7.0 Member Contribution**

|  |  |
| --- | --- |
| Member | Contribution |
| Senthilkumar Dhavasre | |  | | --- | |  |  |  | | --- | | * Wrote report sections 1.0 Problem Statement, 2.0 Technical Setup & Dataset Description, 3.1 Data Preprocessing, and 6.0 Summary & Reflection * Implemented load\_and\_standardize\_data() for standardising multiple dataset formats. * Developed clean\_data() for removing duplicates and null values. * Built advanced\_text\_preprocessing() for text cleaning and spam feature extraction. * Handled data sampling & preprocessing logic in train\_model() up to feature extraction. * Built interactive components get\_random\_test\_email() and game\_explaination(). * Implemented model tokenisation, sequence padding, and two-branch LSTM + Dense architecture in train\_model(). | |
| Chia Eason | * Wrote report sections 3.2 Data Modelling, 3.3 Model Evaluation, 4.0 Application Development, and 5.0 Application Deployment. * Added model compilation, training with early stopping, and evaluation metrics. * Developed report(), confusion(), loss\_validation(), and roc\_auc\_curve() * Created prediction & explanation functions: predict\_spam\_message(), explain\_prediction(), create\_user\_friendly\_explanation(). * Designed full main() Streamlit app with Single Prediction, Batch Testing, Game, and Model Stats tabs. * Created dataset loading function load\_data() and filtering logic. |

# **8.0 Citations**

[1] Altwaijry, N., Al-Turaiki, I., Alotaibi, R., & Alakeel, F. (2024). Advancing Phishing Email Detection: A Comparative Study of Deep Learning models. *Sensors*, *24*(7), 2077. <https://doi.org/10.3390/s24072077>

[2] Baker, E., & Cartier, M. (n.d.). *Phishing Trends Report (Updated for 2025)*. <https://hoxhunt.com/guide/phishing-trends-report#key-phishing-statistics-for-2025>

[3] *Fundamental concepts - Streamlit Docs*. (n.d.). <https://docs.streamlit.io/get-started/fundamentals>

[4] *Fundamentals*. (n.d.). <https://plotly.com/python/plotly-fundamentals/>

[5] GeeksforGeeks. (2025a, July 12). *StringIO module in Python*. GeeksforGeeks. <https://www.geeksforgeeks.org/python/stringio-module-in-python/>

[6] GeeksforGeeks. (2025b, August 13). *What is LSTM Long Short Term Memory?* GeeksforGeeks. <https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/>

[7] Ghosh, D. (2025, April 15). *Phishing in 2025: How the Game Has Changed and Why Awareness Matters More Than Ever*. Medium. <https://medium.com/@dipayanghoshk/phishing-in-2025-how-the-game-has-changed-and-why-awareness-matters-more-than-ever-fcb5ef681977>

[8] Mummidivarapu, S. R. (2024, July 3). *Hands-On LIME: Practical Implementation for Image, Text, and Tabular Data*. Medium. <https://medium.com/@shreeraj260405/hands-on-lime-practical-implementation-for-image-text-and-tabular-data-95566da87f57>

[9] Team, K. (n.d.-a). *Keras documentation: LSTM layer*. <https://keras.io/api/layers/recurrent_layers/lstm/>

[10] Team, K. (n.d.-b). *Keras documentation: Tokenizer*. <https://keras.io/keras_hub/api/tokenizers/tokenizer/>

[11] *Tokenizer*. (n.d.). <https://huggingface.co/docs/transformers/en/main_classes/tokenizer>

[12] Visani, G. (2021, January 28). *LIME: explain Machine Learning predictions*. Medium. <https://medium.com/data-science/lime-explain-machine-learning-predictions-af8f18189bfe>

[13] *What is Phishing and How it Impacts All Email Users?* (2025, July 30). /. <https://www.kaspersky.com/resource-center/preemptive-safety/what-is-phishings-impact-on-email>

# **9.0 Appendix**

A black rectangular object with white text

AI-generated content may be incorrect.

Figure 2.1.1 – Initialise Streamlit

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 2.1.2 – CSS Styling Part 1

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 2.1.3 – CSS Styling Part 2

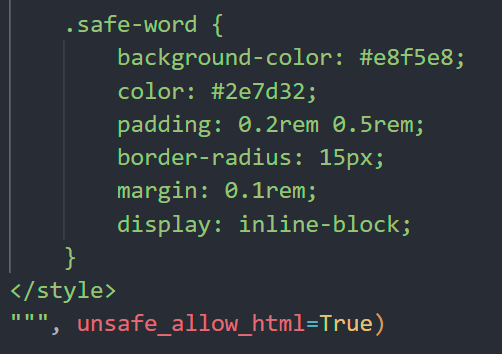


Figure 2.1.4 – CSS Styling Part 3

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 2.1.5 – Imports Part 1

A black background with colorful text

AI-generated content may be incorrect.

Figure 2.1.6 – Imports Part 2

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.2.1.1 – predict\_spam\_message() function, Line 448 - 499

A black background with colorful text

AI-generated content may be incorrect.

Figure 4.2.2.1.2 – predict\_spam\_message() function, Line 500 - 510

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.2.2.4 – explain\_prediction() function, Lines 512 - 556

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.2.3.4 – create\_user\_friendly\_explanation() function, Lines 558 - 597

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 4.2.2.4.4 – Single Prediction Tab Front End, Lines 740 – 793

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.2.4.5 – Single Prediction Tab Front End, Lines 793 – 847

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 4.2.2.4.6 – Single Prediction Tab Front End, Lines 848 – 900

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 4.2.2.4.7 – Single Prediction Tab Front End, Lines 901 – 914

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 4.2.3.1 – Batch Testing Front End, Lines 881 - 936

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 4.2.3.2 – Batch Testing Front End, Lines 937 - 990

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.3.3 – Batch Testing Front End, Lines 990 – 1049

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.3.4 – Batch Testing Front End, Lines 1050 - 1110

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.3.5 – Batch Testing Front End, Lines 1100 – 1148

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 4.2.4.1.2 – Batch Testing Front End, Lines 610 – 643

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.4.2.3 – Batch Testing Front End, Lines 645 - 691

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.4.1 – Email Classification Game Lines 1150 - 1208

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.4.2 – Email Classification Game Lines 1210 – 1269

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.4.4 – Email Classification Game Lines 1270 - 1306

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 4.2.5.2 – Model Stats Tab, Lines 1307 - 1365

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.5.3 – Model Stats Tab, Lines 1366 - 1425

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.5.4 – Model Stats Tab, Lines 1426 - 1482

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.5.5 – Model Stats Tab, Lines 1483 - 1543

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.2.5.6 – Model Stats Tab, Lines 1544 – 1599

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.2.1 – Github Repository

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.2.2 – EATC.py source code

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.2.3 – Requirements.txt

A black screen with white text

AI-generated content may be incorrect.

Figure 5.2.4 – Deploy Button