To understand more on the set of all Turing machines, we define the language
L_{TM} = {<M, w> | M is a Turing machine and M accepts w}
Here w is a string over the input alphabet A.
We will prove that L_{TM} is a Turing-recognisable language, but L_{TM} is **not** Turing-decidable.

Proposition: L_{TM} is a Turing-recognisable language.

Proof: We define a Turing machine U that recognises L_{TM}:
U = on input <M, w>, where M is a Turing machine and w is a string.
1. Simulate M on string w.
2. If M ever enters its accept state then accept.
   If M ever enters its reject state then reject.

U loops on input <M, w> if M loops on w ⇒ U is a recogniser but not a decider. (*q.e.d*)

Remark:
The Turing machine U is an example of the **universal Turing machine** first proposed by Turing in 1936. This idea of a universal Turing machine led to the development of stored-program computers.

NB: Philosophically, the universal Turing machine we just constructed runs into the following big issues:
1. U itself is a Turing machine. What happens when U is given an input <U, w>?
2. The encoding of a Turing machine is a string. What happens when we input <M, <M>> or even worse <U, <U>>?

We are getting very close to Russell's paradox, the set $\Gamma = \{D \mid D \notin D\}$ which showed the axioms of naive set theory were inconsistent and led to more complicated axioms.
In one case, these issues lead to showing the language L_{TM} cannot possibly be Turing-decidable.

Proposition: L_{TM} is not Turing-decidable.

Proof: Assume L_{TM} is Turing-decidable and obtain a contradiction.
If L_{TM} is Turing-decidable, then $\exists$ decider H for L_{TM}.
Given input <M, w>, H:

1. Accepts if M accept w.
2. Rejects if M does not accept w.

We now construct another Turing machine D with H as a subroutine, which belongs like the set Γ defined by Russell
D = on input <M>, where M is a Turing machine:

1. Run H on input <M, <M>>
2. Output the opposite of what H outputs.
   If H accepts, then reject.
   If H rejects, then accept.

Now, let us run D on its own encoding <D>:
D on input <D>:

1. Accepts if D does not accept <D>
2. Rejects if D accepts <D>

⇒⇐ D cannot exist, hence H cannot exist. The language L_{TM} has no decider. (*q.e.d*)

Example of a language that is not Turing-recognisable:
Task: Use what we know about L_{TM} to build an example of a language that is not Turing-recognisable.

Definition:
GIven an alphabet A that is finite, ($A^* = A^0 \cup A^1 \cup \ldots \cup A^\infty$), and then a language $L \subset A^*$, we define the complement ~L of L as $\sim L = A^* \setminus L$, i.e. all words over A that are not in L.

Definition:
A language L is called co-Turing-recognisable if its complement ~L is Turing-recognisable.

Theorem: A language L is decidable ⇔ L is Turing-recognisable and co-Turing-recognisable

Proof:
"⇒" If L is decidable ⇒ L is Turing-recognisable. Note that if L is decidable ⇒ $\exists$ a Turing machine M that decides L.
Build a Turing machine ꝏ that reverses the output of M, i.e. if M accepts a string w, then ꝏ rejects the same string w. If M rejects w then ꝏ accepts w.
M is therefore a decider for ~L ⇒ ~L is Turing-decidable ⇒ ~L is Turing-recognisable, so L is Turing-recognisable and co-Turing-recognisable.

"$\Leftarrow$" If both L and ~L are Turing-recognisable $\Rightarrow$ $\exists$ M_1 that recognises L and $\exists$ M_2 that recognises ~L. We use Turing machines M_1 and M_2 to build a decider M for L as follows:
M = on input w, where w is a string:

1. Run both M_1 and M_2 on input w in parallel.
2. If M_1 accepts, then accept.
   If M_2 accepts, then reject.

Running M_1 and M_2 in parallel simply means the M has two tapes: one for simulating M_1 and one for M_2.
Note that for any string w, either $w \in L$ or $w \in {\sim}L$, which means either M_1 or M_2 accepts w $\Rightarrow$ M either accepts or rejects any string.
In fact, M accepts w $\Leftrightarrow$ $w \in L$ by construction $\Rightarrow$ M is a decider for L.
$\Rightarrow$ L is Turing-decidable. (*q.e.d*)

Corollary: $\sim(L_{TM})$ Is **not** Turing-recognisable.

Proof:
We proved L_{TM} is Turing-recognisable. If $\sim(L_{TM})$ were Turing-recognisable, then L_{TM} would be both Turing-recognisable and co-Turing-recognisable.
$\Rightarrow$ By the previous theorem, L_{TM} would be Turing-decidable $\Rightarrow\Leftarrow$ as we proved the contrary
$\Rightarrow$ $\sim(L_{TM})$ is not Turing-recognisable, and we have constructed our example of a non Turing-recognisable language. (*q.e.d*)