Département des Technologies de l'information
et de la communication (TIC)

Informatique et systèmes de communication

High Performance Coding

HPC

Laboratoire 1

DTMF

Etudiant André Costa
Professeur Alberto Dassatti
Assistant Bruno Da Rocha Carvalho
Année 2025

Yverdon-les-Bains, 13.03.2025

Introduction

Ce laboratoire a comme objectif la familiarisation avec le langage `C`.

Pour cela, il est demandé d'implémenter un système d'encodage et décodage `DTMF` (Dual-Tone Multi-Frequency).

Pour l'implémentation de ce système, il nous est proposé de commencer d'abord avec l'encodage avant d'implémenter deux décodeurs différents.

Une fois l'implémentation complète, il nous est demandé de mesurer le temps d'exécution du système.

Guide d'utilisation rapide

1. Compilez le code en utilisant le `CMakeLists.txt` à la racine du répertoire

```
mkdir -p build
cd build
cmake ..
make -j$(nproc)
cd ..
```

2. Exportez votre message sur un fichier text

```
echo "je suis un bg" > je_suis_un_bg.txt
```

3. Encodez votre message

```
./build/dtmf_encdec encode je_suis_un_bg.txt je_suis_un_bg.wav
Encoding alone took 0.014179 seconds
```

4. Décodez votre message en utilisant votre décodeur de préférence

   - Décodeur Fréquentiel

   ```
   ./build/dtmf_encdec decode je_suis_un_bg.wav
     Using 0.539966 as silence amplitude threshold
     Decoding alone took 0.017162 seconds
     Decoded: je suis un bg
   ```

   - Décodeur Temporel

   ```
   ./build/dtmf_encdec decode_time_domain je_suis_un_bg.wav
     Using 0.539966 as silence amplitude threshold
     Decoding alone took 0.001462 seconds
     Decoded: je suis un bg
   ```

Encodage

L'encodage DTMF consiste à lire un message depuis un fichier texte et à générer un fichier audio au format wave.

L'algorithme fonctionne comme suit :

1. Identifier le bouton correspondant à chaque lettre du message.
2. Déterminer le nombre de pressions nécessaires pour sélectionner la lettre.
3. Générer la séquence de tonalités DTMF correspondante, en insérant une pause de `0.05` secondes entre chaque pression.
4. Ajouter une pause de `0.2` secondes entre les lettres.

Tonalité

La tonalité est déterminé par la formule suivante :

$$s(t) = A \times (\sin(2\pi f_{\text{row}} t) + \sin(2\pi f_{\text{col}} t))$$

où :

- A est l'amplitude du signal,
- f_{row} et f_{col} sont les fréquences DTMF associées au caractère,
- t est le temps.

t peut être determiné avec :

$$t = \frac{\text{sample\_number}}{\text{sample\_rate}}$$

Fréquence d'échantillonage (Sample Rate)

Pour l'encodage, la fréquence d'échantillonage est fixée à `44.1 kHz`, ce qui est un habituel pour des applications audio.

La fréquence minimale est toute fois plus basse. En effet, selon le théorème de Nyquist, la fréquence minimale doit être au moins deux fois plus élevé que la fréquence maximale que nous allons utiliser ce qui donne la relation suivante:

$$\text{fs} \geq 2 * 1477 = 2954 \text{ Hz}$$

Avec la fréquence choisie de `44.1 kHz` nous respectons largement cette contrainte.

Notons qu'une autre fréquence qui est très utilisé pour la téléphonie est la fréquence de `8kHz`.

Autres paramètres

Vu le challenge de partager des fichier encodés entre la classe les paramètres suivants peuvent être modifiés pour générer des fichiers plus ou moins difficiles à décoder.

```
#define SILENCE_F1    0
#define SILENCE_F2    0
```

Les paramètres `SILENCE_FX` permettent d'indiquer les deux fréquences utilisés lors de la génération du silence. Par défaut à `0`, ces valeurs peuvent être modifiées pour générer du bruit lorsqu'il devrait y avoir du silence

```
#define EXTRA_PRESSES    0
```

Le paramètre `EXTRA_PRESSES` permet d'indiquer combien de tours supplémentaires il faudra encoder pour chaque lettre. Par exemple, la valeur de `1` encodera la valeur `2` avec `5` pressions sur le deuxième bouton à la place de `1` pression.

Decodage

Contraintes du Système

Les 3 contraintes du système sont:

1. Une durée d'un son de 0.2 secondes par caractère.
2. Une pause de 0.2 secondes entre deux caractères.
3. Une pause de 0.05 secondes entre plusieurs pressions pour un même caractère.

En profitant de ces contraintes il est possible de déterminer que, une fois trouvé une pression de touche, la prochaine touche viendra soit :

1. 0.25 secondes plus tard
   - Il sera donc la même touche qui aurait été pressée
2. 0.40 secondes
   - Pour une nouvelle lettre

De plus, la période maximale des signaux générés par notre système correspond à l'envers de fréquence minimale:

$$T_{\{\min\}} = \frac{1}{F_{\min}} = \frac{1}{697} \sim= 1.56 \text{ ms}$$

Cela veut dire qu'en prenant une fenêtre de `0.05s` il est possible de capturer n'importe quel signal généré par notre système.

Ici, l'utilisation de `0.05s` correspond au temps de pause minimale entre chaque touche et permet de facilement rester alignés avec le signal à décoder. Avec cette analyse, des optimisations additionneles seraient possibles mais elles n'ont pas été exploités dans le cadre de ce laboratoire.

Algorithme Général

Alignement avec première touche

Tout d'abord, il faut s'aligner avec le début du signal. Pour cela, la transformé de fourier a été utilisé, l'algorithme est le suivant:

1. Sélectionner une fenêtre de `0.05s`.
2. Identifier les deux fréquences dominantes dans cette fenêtre.
3. Si ces fréquences appartiennent à l'intervalle $[650\text{Hz}; 1500\text{Hz}]$, considérer cette fenêtre comme le début de l'appui sur une touche.
4. Sinon, avancer de `0.05s` et répéter l'opération.

Cet alignement est important car le fichier `wave` pourrait démarrer avec du silence.

Décodage - Algorithme Général

Une fois aligné avec le signal, les contraintes du système garantissent que l'alignement est maintenu. L'algorithme fonctionne comme suit :

1. Sélectionner une fenêtre de `0.05 secondes`.
2. Vérifier la présence de silence :
   - Une fenêtre est considérée comme silencieuse si son amplitude est inférieure à 90% de celle détectée lors de la première touche.
   - Si du silence est détecté, avancer de `0.15 secondes` et revenir à l'étape `1`.
3. Décoder le bouton pressé :

- Si l'identification du bouton n'est pas possible, considérer cette fenêtre comme invalide, avancer de `0.15` secondes, puis retourner à l'étape `1`.
- Si le bouton a été identifié, comptabiliser la pression, avancer de `0.25` secondes, puis retourner à l'étape `1`.

En cas de détection d'une fenêtre invalide ou silencieuse, on retient le dernier bouton pressé ainsi que le nombre d'appuis consécutifs pour déterminer la lettre correcte.

Cet algorithme général est utilisé pour deux méthodes de décodage distinctes :

- Une analyse fréquentielle via la Transformée de Fourier.
- Une analyse temporelle par corrélation.

Analyse Fréquentielle

L'analyse fréquentielle repose sur l'utilisation de la Transformée de Fourier sur une fenêtre de `0.05 secondes`. On en extrait les deux fréquences dominantes et, si celles-ci appartiennent à l'intervalle [650Hz; 1500Hz], elles sont considérées comme valides. On identifie alors les fréquences les plus proches parmi les fréquences DTMF afin de déterminer le bouton correspondant.

Analyse Temporelle

L'analyse temporelle consiste à générer des signaux de référence pour chaque touche, puis à calculer la corrélation entre la fenêtre de `0.05 secondes` et ces signaux de référence. Le bouton associé au signal de référence présentant la meilleure corrélation est alors considéré comme celui ayant été pressé.

La formule de la corrélation est la suivante :

$$R = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Avec :

- x: Le signal à corréler
- $\bar{x}$: La moyenne du signal à corréler
- y: Le signal de référencce
- $\bar{y}$: La moyenne du signal de référencce

Comme dit précedemment, l'analyse se fait par des fenêtres de `0.05 secondes` mais une fenêtre plus petite est suffisamment large pour faire cette corrélation.

Le nombre d'échantillons minimale pour la période maximale est donné par :

$$N_{\text{samples}_{\text{min}}} = \frac{\text{sample\_rate}}{F_{\text{min}}}$$

Avec:

- $F_{\text{min}} = 697\ \text{Hz}$

Et le nombre d'échantillons pris pour calculer la corrélation :

$$N_{\text{samples}} = 5 * N_{\text{samples}_{\text{min}}}$$

Le nombre `5` est arbitraire et a été choisi pour avoir une meilleure précision.

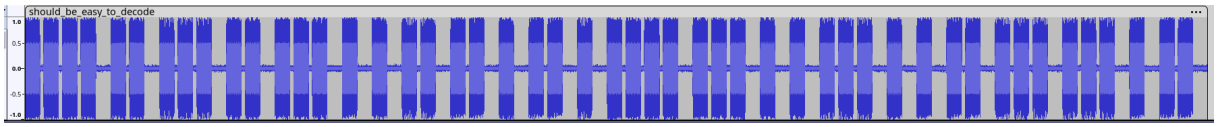
Voici la différence entre le nombre d'échantillons en utilisant cette formule vs l'utilisation de la fenêtre complète de `0.05 secondes`.

sample_rate	$N_{\text{samples}} = 5 * N_{\text{samples}_{\text{min}}}$	Fenêtre de `0.05 secondes`	Rapport
8kHz	57	400	14%
44.1kHz	316	22050	1.4%

Résultats

Should Be Easy To Decode

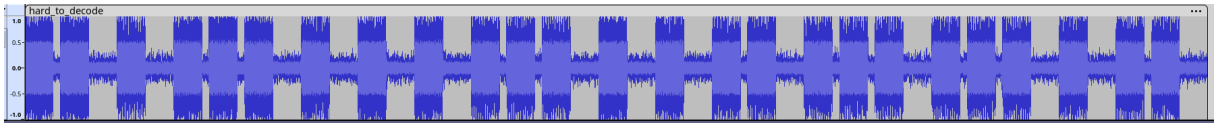
- Contient un peu de bruit




```
Decoding audio/should_be_easy_to_decode.wav
-- Frequency Domain --
Using 0.899973 as silence amplitude threshold
Decoding alone took 0.003061 seconds
Decoded: should be easy to decode
-- Time Domain --
Using 0.899973 as silence amplitude threshold
Decoding alone took 0.000439 seconds
Decoded: should be easy to decode
```

Hard To Decode

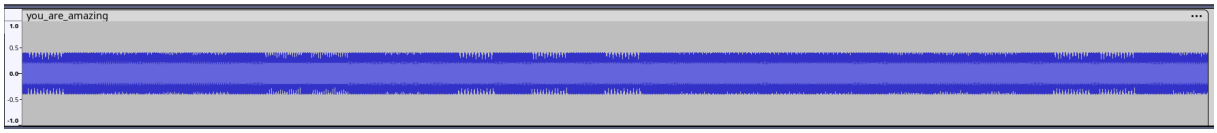
- Contient plus du bruit




```
Decoding audio/hard_to_decode.wav
-- Frequency Domain --
Using 0.899973 as silence amplitude threshold
Decoding alone took 0.001612 seconds
Decoded: hard to decode
-- Time Domain --
Using 0.899973 as silence amplitude threshold
Decoding alone took 0.000341 seconds
Decoded: hard to decode
```

You Are Amazing

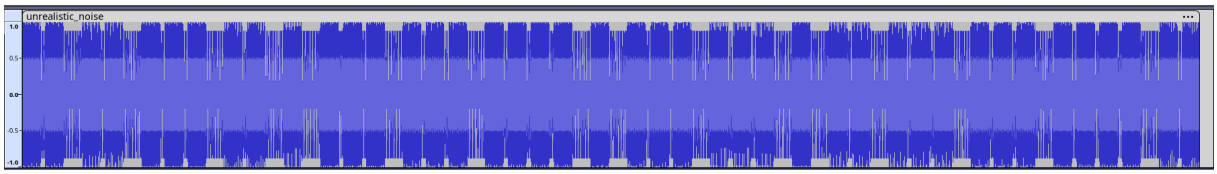
- Contient du bruit seulement dans les moments de silence avec une amplitude égale au moments de pression.




```
Decoding audio/you_are_amazing.wav
-- Frequency Domain --
Using 0.359953 as silence amplitude threshold
Decoding alone took 0.027844 seconds
Decoded: you are amazing
-- Time Domain --
Using 0.359953 as silence amplitude threshold
Decoding alone took 0.00188 seconds
Decoded: you are amazing
```

Unrealistic Noise

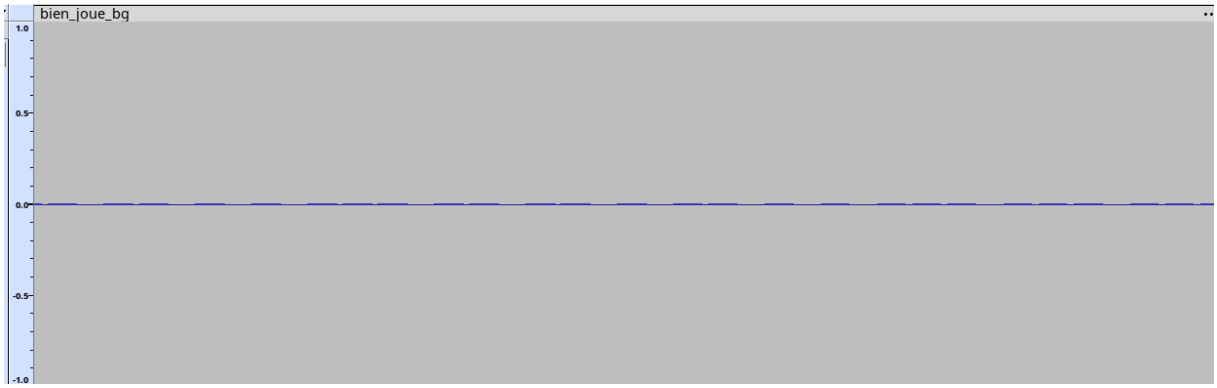
- Contient du bruit seulement dans les moments de silence avec une amplitude égale au moments de pression.




```
Decoding audio/unrealistic_noise.wav
-- Frequency Domain --
Using 0.89976 as silence amplitude threshold
Decoding alone took 0.030596 seconds
Decoded: unrealistic noise
-- Time Domain --
Using 0.89976 as silence amplitude threshold
Decoding alone took 0.001835 seconds
Decoded: unrealistic noise
```

Bien Joue Bg

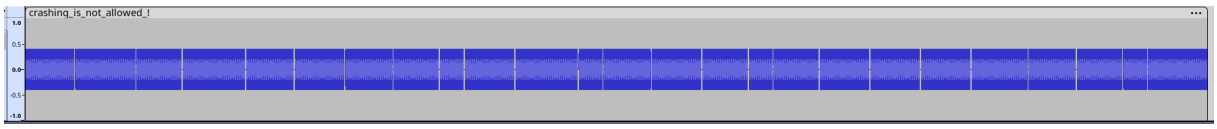
- Amplitude est infiniment petite




```
Decoding audio/bien_joue_bg.wav
-- Frequency Domain --
Using 0.00179916 as silence amplitude threshold
Decoding alone took 0.023035 seconds
Decoded: bien joue bg !!!
-- Time Domain --
Using 0.00179916 as silence amplitude threshold
Decoding alone took 0.001768 seconds
Decoded: bien joue bg !!!
```

Crashing Is Not Allowed

- Encodé avec un nombre de `EXTRA_PRESSES` élevé




```
Decoding audio/crashing_is_not_allowed_!.wav
-- Frequency Domain --
Using 0.359832 as silence amplitude threshold
Decoding alone took 0.34178 seconds
Decoded: crashing is not allowed !
-- Time Domain --
Using 0.359832 as silence amplitude threshold
Decoding alone took 0.013804 seconds
Decoded: crashing is not allowed !
```

Analyse

De manière surprenante, le décodage basé sur l'analyse temporelle par corrélation parvient à identifier correctement les touches, même en présence d'un bruit important ou d'un bruit artificiel peu réaliste.

Cette robustesse s'explique par les contraintes du système qui nous permet de s'aligner avec le signal très facilement, une fois qu'on sait où les pressions peuvent être, la moitié du travail est déjà effectué.

Le décodeur fréquentiel reste supérieur en toutes circonstances. Grâce à l'utilisation de la Transformée de Fourier, il identifie avec précision les fréquences dominantes, offrant un décodage stable et fiable, y compris en présence de bruit. Son efficacité est due à sa capacité à isoler les composantes fréquentielles du signal, ce qui le rend plus robuste face aux variations d'intensité ou aux déformations temporelles du signal d'entrée.

Il est surtout plus fiable lorsqu'il n'est pas possible de s'aligner parfaitement avec le signal. Dans de telles situations, l'analyse fréquentielle permet de retrouver les fréquences des touches même si le signal est légèrement décalé ou perturbé.

Performance

En utilisant l'outil `hyperfine` il est possible de facilement déterminer le temps pris par notre application.

De plus, pour pouvoir savoir le temps qui prend la partie de décodage, cette fonction est *wrappé* sur deux appels de `clock`.

```c
clock_t t;
t = clock();
char *value = decode_fn(&decoder);
t = clock() - t;
const double time_taken = ((double)t) / CLOCKS_PER_SEC;
printf("Decoding alone took %g seconds\n", time_taken);
```

Ordinateur

Le code a été lancé dans un ordinateur avec le CPU et mémoire cache suivantes:

```
CPU: AMD Ryzen 5 3600 (12) @ 4.20 GHz
Caches (sum of all):
  L1d:                    192 KiB (6 instances)
  L1i:                    192 KiB (6 instances)
  L2:                     3 MiB (6 instances)
  L3:                     32 MiB (2 instances)
```

Hard To Decode

```
> hyperfine "./build/dtmf_encdec decode audio/hard_to_decode.wav" --shell=none --
warmup 10
Benchmark 1: ./build/dtmf_encdec decode audio/hard_to_decode.wav
  Time (mean ± σ):       2.9 ms ±   0.1 ms    [User: 1.9 ms, System: 1.0 ms]
  Range (min … max):     2.8 ms …   4.0 ms    975 runs

  Warning: Statistical outliers were detected. Consider re-running this benchmark
on a quiet system without any interferences from other programs. It might help to
use the '--warmup' or '--prepare' options.

> hyperfine "./build/dtmf_encdec decode_time_domain audio/hard_to_decode.wav" --
shell=none --warmup 10
Benchmark 1: ./build/dtmf_encdec decode_time_domain audio/hard_to_decode.wav
  Time (mean ± σ):       1.6 ms ±   0.1 ms    [User: 0.5 ms, System: 1.0 ms]
  Range (min … max):     1.4 ms …   2.0 ms    1844 runs

  Warning: Statistical outliers were detected. Consider re-running this benchmark
on a quiet system without any interferences from other programs. It might help to
use the '--warmup' or '--prepare' options.
```

Mesurer la performance de notre décodeur avec un signal si court n'est pas la meilleure idée, car il est difficile d'avoir des résultats consistents. De plus, la partie plus intéressant - le décodage - est surpassé par des autres tâches comme la lecture du fichier `wave`. Ce qui peut être vu lors que la sortie `stdout` n'est pas consommé:

```
> time ./build/dtmf_encdec decode audio/hard_to_decode.wav
Using 0.899973 as silence amplitude threshold
Decoding alone took 0.001628 seconds
Decoded: hard to decode
./build/dtmf_encdec decode audio/hard_to_decode.wav  0.00s user 0.00s system 88%
cpu 0.004 total
```

Ici, nous pouvons voir que, avec le décodeur en domaine fréquentiel, le décodage prend environ `1.6ms` alors que le programme prend `4ms`.

```
> time ./build/dtmf_encdec decode_time_domain audio/hard_to_decode.wav
Using 0.899973 as silence amplitude threshold
Decoding alone took 0.000246 seconds
Decoded: hard to decode
./build/dtmf_encdec decode_time_domain audio/hard_to_decode.wav  0.00s user 0.00s
system 85% cpu 0.003 total
```

Avec le décodeur en domaine temporel, la différence est encore plus flagrante. Le décodage prend environ `0.2ms` alors que le programme prend `3ms`.

Avec ces deux outils il est possible d'estimer que le décodage avec l'analyse linéaire est environ `6.58` fois plus rapide que le décodage avec l'analyse fréquentielle.

Le programme en soit tourne `1.8` fois plus rapidement.

Crashing is not allowed

Vu la longeur du signal `crashing_is_not_allowed` il est possible d'avoir des résultats plus fiables.

```
> hyperfine "./build/dtmf_encdec decode audio/crashing_is_not_allowed_\\\!.wav" --
shell=none --warmup 10
Benchmark 1: ./build/dtmf_encdec decode audio/crashing_is_not_allowed_\!.wav
  Time (mean ± σ):     364.9 ms ±   7.8 ms    [User: 348.0 ms, System: 15.3 ms]
  Range (min … max):   357.3 ms … 379.7 ms    10 runs

> hyperfine "./build/dtmf_encdec decode_time_domain audio/
crashing_is_not_allowed_\\\!.wav" --shell=none --warmup 10
Benchmark 1: ./build/dtmf_encdec decode_time_domain audio/
crashing_is_not_allowed_\!.wav
  Time (mean ± σ):      28.9 ms ±   0.3 ms    [User: 14.0 ms, System: 14.7 ms]
  Range (min … max):    28.1 ms …  29.7 ms    102 runs
```

```
> time ./build/dtmf_encdec decode audio/crashing_is_not_allowed_\!.wav
Using 0.359832 as silence amplitude threshold
Decoding alone took 0.341306 seconds
Decoded: crashing is not allowed !
./build/dtmf_encdec decode audio/crashing_is_not_allowed_!.wav  0.34s user 0.02s
system 99% cpu 0.360 total
> time ./build/dtmf_encdec decode_time_domain audio/crashing_is_not_allowed_\!.wav
Using 0.359832 as silence amplitude threshold
Decoding alone took 0.013742 seconds
Decoded: crashing is not allowed !
./build/dtmf_encdec decode_time_domain audio/crashing_is_not_allowed_!.wav  0.01s
user 0.01s system 98% cpu 0.030 total
```

Maintenant, pour le décodage en domaine fréquentiel, le programme passe la plupart du temps à decoder, ce qui est attendu. Le décodage dans le domaine temporel est toujours très rapide, sûrement à cause de la taille réduite du nombre d'échantillons.

Ici le décodage linéaire est environ 25 fois plus rapide que le décodage avec l'analyse fréquentielle.

Le programme en soit tourne 12.6 fois plus rapidement.

Graphique

Finalement, voici un graphe avec les informations ressorties des mesures :

Name	Number of Samples	Decoding Time - FFT [ms]	Decoding Time - Correlation [ms]
Crashing is not allowed	608805	341.3	13.7
Hard To Decode	66800	1.6	0.2

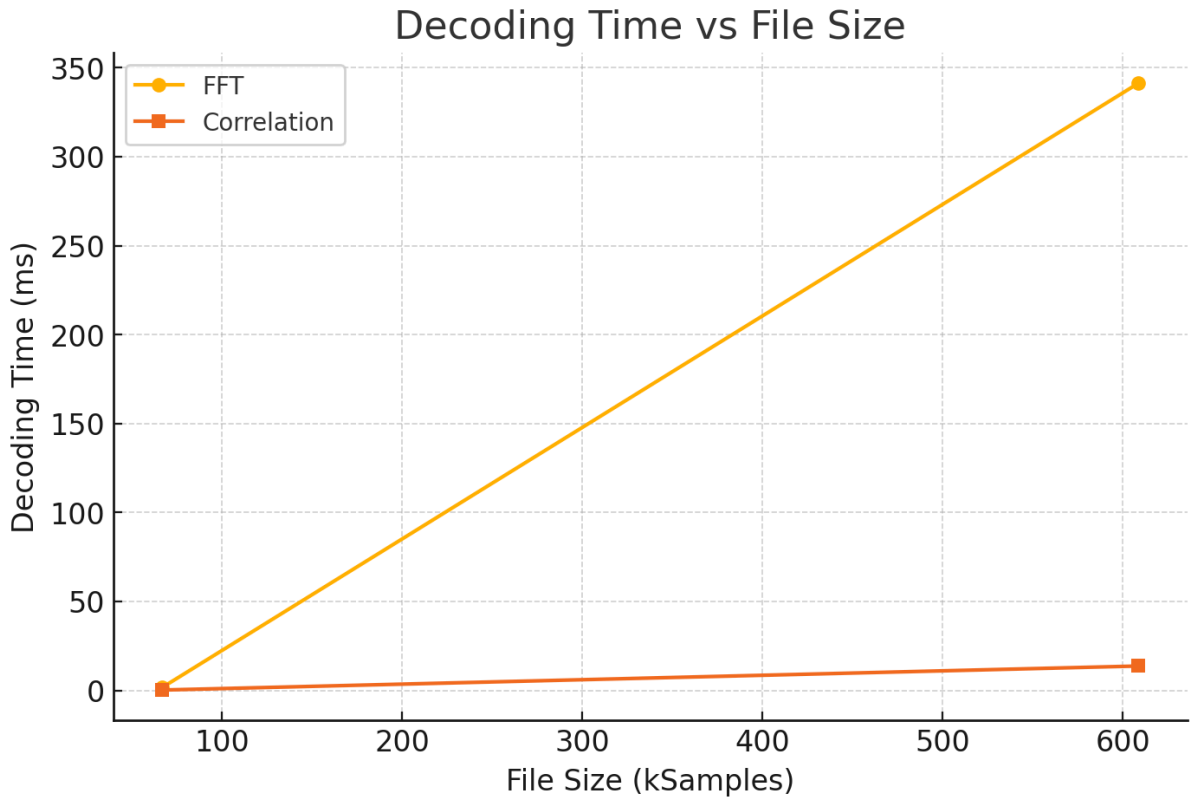


Figure 1: Decoding Time vs File Size Per Algorithm